

axiomTM



The 30 Year Horizon

<i>Manuel Bronstein</i>	<i>William Burge</i>	<i>Timothy Daly</i>
<i>James Davenport</i>	<i>Michael Dewar</i>	<i>Martin Dunstan</i>
<i>Albrecht Fortenbacher</i>	<i>Patrizia Gianni</i>	<i>Johannes Grabmeier</i>
<i>Jocelyn Guidry</i>	<i>Richard Jenks</i>	<i>Larry Lambe</i>
<i>Michael Monagan</i>	<i>Scott Morrison</i>	<i>William Sit</i>
<i>Jonathan Steinbach</i>	<i>Robert Sutor</i>	<i>Barry Trager</i>
<i>Stephen Watt</i>	<i>Jim Wen</i>	<i>Clifton Williamson</i>

Volume 9: Axiom Compiler

Portions Copyright (c) 2005 Timothy Daly

The Blue Bayou image Copyright (c) 2004 Jocelyn Guidry

Portions Copyright (c) 2004 Martin Dunstan

Portions Copyright (c) 1991-2002,
The Numerical Algorithms Group Ltd.
All rights reserved.

This book and the Axiom software is licensed as follows:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical Algorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

Cyril Alberga	Roy Adler	Richard Anderson
George Andrews	Henry Baker	Stephen Balzac
Yuriy Baransky	David R. Barton	Gerald Baumgartner
Gilbert Baumsлаг	Fred Blair	Vladimir Bondarenko
Mark Botch	Alexandre Bouyer	Peter A. Broadbery
Martin Brock	Manuel Bronstein	Florian Bundschuh
William Burge	Quentin Carpent	Bob Caviness
Bruce Char	Cheekai Chin	David V. Chudnovsky
Gregory V. Chudnovsky	Josh Cohen	Christophe Conil
Don Coppersmith	George Corliss	Robert Corless
Gary Cornell	Meino Cramer	Claire Di Crescenzo
Timothy Daly Sr.	Timothy Daly Jr.	James H. Davenport
Jean Della Dora	Gabriel Dos Reis	Michael Dewar
Claire DiCrescendo	Sam Dooley	Lionel Ducos
Martin Dunstan	Brian Dupee	Dominique Duval
Robert Edwards	Heow Eide-Goodman	Lars Erickson
Richard Fateman	Bertfried Fauser	Stuart Feldman
Brian Ford	Albrecht Fortenbacher	George Frances
Constantine Frangos	Timothy Freeman	Korrinn Fu
Marc Gaetano	Rudiger Gebauer	Kathy Gerber
Patricia Gianni	Holger Gollan	Teresa Gomez-Diaz
Laureano Gonzalez-Vega	Stephen Gortler	Johannes Grabmeier
Matt Grayson	James Griesmer	Vladimir Grinberg
Oswald Gschnitzer	Jocelyn Guidry	Steve Hague
Vilya Harvey	Satoshi Hamaguchi	Martin Hassner
Ralf Hemmecke	Henderson	Antoine Hersen
Pietro Iglio	Richard Jenks	Kai Kaminski
Grant Keady	Tony Kennedy	Paul Kosinski
Klaus Kusche	Bernhard Kutzler	Larry Lambe
Frederic Lehouby	Michel Levaud	Howard Levy
Rudiger Loos	Michael Lucks	Richard Luczak
Camm Maguire	Bob McElrath	Michael McGettrick
Ian Meikle	David Mentre	Victor S. Miller
Gerard Milmeister	Mohammed Mobarak	H. Michael Moeller
Michael Monagan	Marc Moreno-Maza	Scott Morrison
Mark Murray	William Naylor	C. Andrew Neff
John Nelder	Godfrey Nolan	Arthur Norman
Jinzhong Niu	Michael O'Connor	Kostas Oikonomou
Julian A. Padget	Bill Page	Jaap Weel
Susan Pelzel	Michel Petitot	Didier Pinchon
Claude Quitte	Norman Ramsey	Michael Richardson
Renaud Rioboo	Jean Rivlin	Nicolas Robidoux
Simon Robinson	Michael Rothstein	Martin Rubey
Philip Santas	Alfred Scheerhorn	William Schelter
Gerhard Schneider	Martin Schoenert	Marshall Schor
Fritz Schwarz	Nick Simicich	William Sit
Elena Smirnova	Jonathan Steinbach	Christine Sundaresan
Robert Sutor	Moss E. Sweedler	Eugene Surowitz
James Thatcher	Baldir Thomas	Mike Thomas
Dylan Thurston	Barry Trager	Themos T. Tsikas
Gregory Vanuxem	Bernhard Wall	Stephen Watt
Juergen Weiss	M. Weller	Mark Wegman
James Wen	Thorsten Werther	Michael Wester
John M. Wiley	Berhard Will	Clifton J. Williamson
Stephen Wilson	Shmuel Winograd	Robert Wisbauer
Sandra Wityak	Waldemar Wiwianka	Knut Wolf
Clifford Yapp	David Yun	Richard Zippel
Evelyn Zoernack	Bruno Zuercher	Dan Zwillinger

Contents

0.1	Makefile	1
1	Overview	3
1.1	The Input	4
1.2	The Output, the EQ.nrlib directory	8
1.3	The code.lsp and EQ.lsp files	9
1.4	The code.o file	23
1.5	The info file	23
1.6	The EQ.fn file	26
1.7	The index.kaf file	31
1.7.1	The index offset byte	33
1.7.2	The “loadTimeStuff”	33
1.7.3	The “compilerInfo”	35
1.7.4	The “constructorForm”	42
1.7.5	The “constructorKind”	42
1.7.6	The “constructorModemap”	42
1.7.7	The “constructorCategory”	44
1.7.8	The “sourceFile”	45
1.7.9	The “modemaps”	45
1.7.10	The “operationAlist”	47
1.7.11	The “superDomain”	49
1.7.12	The “signaturesAndLocals”	49
1.7.13	The “attributes”	49
1.7.14	The “predicates”	49
1.7.15	The “abbreviation”	50
1.7.16	The “parents”	50
1.7.17	The “ancestors”	51
1.7.18	The “documentation”	51
1.7.19	The “slotInfo”	53
1.7.20	The “index”	55
2	Compiler top level	57
2.1	Global Data Structures	57
2.2	Pratt Parsing	57
2.3)compile	58

2.3.1	Spad compiler	61
2.4	Operator Precedence Table Initialization	62
2.4.1	LED and NUD Tables	62
2.5	Gliph Table	65
2.5.1	Rename Token Table	65
2.5.2	Generic function table	66
2.6	Giant steps, Baby steps	66
3	The Parser	67
3.1	EQ.spad	67
3.2	preparse	71
3.2.1	defvar \$index	72
3.2.2	defvar \$linelist	72
3.2.3	defvar \$echolinestack	72
3.2.4	defvar \$preparse-last-line	72
3.3	Parsing routines	72
3.3.1	defun initialize-preparse	73
3.3.2	defun preparse	76
3.3.3	defun Build the lines from the input for piles	81
3.3.4	defun parsepiles	84
3.3.5	defun add-parens-and-semis-to-line	84
3.3.6	defun preparseReadLine	85
3.3.7	defun skip-ifblock	86
3.3.8	defun preparseReadLine1	87
3.4	I/O Handling	88
3.4.1	defun preparse-echo	88
3.4.2	Parsing stack	88
3.4.3	defstruct \$stack	88
3.4.4	defun stack-load	89
3.4.5	defun stack-clear	89
3.4.6	defmacro stack-/empty	89
3.4.7	defun stack-push	89
3.4.8	defun stack-pop	90
3.4.9	Parsing token	90
3.4.10	defstruct \$token	90
3.4.11	defvar \$prior-token	90
3.4.12	defvar \$nonblank	91
3.4.13	defvar \$current-token	91
3.4.14	defvar \$next-token	91
3.4.15	defvar \$valid-tokens	91
3.4.16	defun token-install	92
3.4.17	defun token-print	92
3.4.18	Parsing reduction	92
3.4.19	defstruct \$reduction	92

4	Parse Transformers	93
4.1	Direct called parse routines	93
4.1.1	defun parseTransform	93
4.1.2	defun parseTran	93
4.1.3	defun parseAtom	94
4.1.4	defun parseTranList	95
4.1.5	defplist parseConstruct	95
4.1.6	defun parseConstruct	95
4.2	Indirect called parse routines	96
4.2.1	defplist parseAnd	97
4.2.2	defun parseAnd	97
4.2.3	defplist parseAtSign	97
4.2.4	defun parseAtSign	98
4.2.5	defun parseType	98
4.2.6	defplist parseCategory	98
4.2.7	defun parseCategory	99
4.2.8	defun parseDropAssertions	99
4.2.9	defplist parseCoerce	99
4.2.10	defun parseCoerce	100
4.2.11	defplist parseColon	100
4.2.12	defun parseColon	100
4.2.13	defplist parseDEF	101
4.2.14	defun parseDEF	101
4.2.15	defun parseLhs	102
4.2.16	defun transIs	102
4.2.17	defun transIs1	102
4.2.18	defun isListConstructor	103
4.2.19	defplist parseDollarGreaterthan	104
4.2.20	defun parseDollarGreaterthan	104
4.2.21	defplist parseDollarGreaterEqual	104
4.2.22	defun parseDollarGreaterEqual	104
4.2.23	defun parseDollarLessEqual	105
4.2.24	defplist parseDollarNotEqual	105
4.2.25	defun parseDollarNotEqual	105
4.2.26	defplist parseEquivalence	106
4.2.27	defun parseEquivalence	106
4.2.28	defplist parseExit	106
4.2.29	defun parseExit	107
4.2.30	defplist parseGreaterEqual	107
4.2.31	defun parseGreaterEqual	107
4.2.32	defplist parseGreaterthan	108
4.2.33	defun parseGreaterthan	108
4.2.34	defplist parseHas	108
4.2.35	defun parseHas	108
4.2.36	defun parseHasRhs	110
4.2.37	defun loadIfNecessary	111

4.2.38	defun loadLibIfNecessary	111
4.2.39	defun updateCategoryFrameForConstructor	112
4.2.40	defun convertOpAlist2compilerInfo	112
4.2.41	defun updateCategoryFrameForCategory	113
4.2.42	defplist parseIf	113
4.2.43	defun parseIf	114
4.2.44	defun parseIf,ifTran	114
4.2.45	defplist parseImplies	116
4.2.46	defun parseImplies	116
4.2.47	defplist parseIn	117
4.2.48	defun parseIn	117
4.2.49	defplist parseInBy	118
4.2.50	defun parseInBy	118
4.2.51	defplist parseIs	119
4.2.52	defun parseIs	119
4.2.53	defplist parseIsnt	119
4.2.54	defun parseIsnt	120
4.2.55	defplist parseJoin	120
4.2.56	defun parseJoin	120
4.2.57	defplist parseLeave	121
4.2.58	defun parseLeave	121
4.2.59	defplist parseLessEqual	121
4.2.60	defun parseLessEqual	122
4.2.61	defplist parseLET	122
4.2.62	defun parseLET	122
4.2.63	defplist parseLETD	123
4.2.64	defun parseLETD	123
4.2.65	defplist parseMDEF	123
4.2.66	defun parseMDEF	123
4.2.67	defplist parseNot	124
4.2.68	defun parseNot	124
4.2.69	defun parseNot	124
4.2.70	defplist parseNotEqual	125
4.2.71	defun parseNotEqual	125
4.2.72	defplist parseOr	125
4.2.73	defun parseOr	125
4.2.74	defplist parsePretend	126
4.2.75	defun parsePretend	126
4.2.76	defplist parseReturn	127
4.2.77	defun parseReturn	127
4.2.78	defplist parseSegment	127
4.2.79	defun parseSegment	128
4.2.80	defplist parseSeq	128
4.2.81	defun parseSeq	128
4.2.82	defplist parseVCONS	129
4.2.83	defun parseVCONS	129

4.2.84	defplist parseWhere	129
4.2.85	defun parseWhere	129
5	Compile Transformers	131
5.0.86	defvar \$NoValueMode	131
5.0.87	defvar \$EmptyMode	131
5.1	Routines for handling forms	131
5.2	Functions which handle == statements	133
5.2.1	defun compDefineAddSignature	133
5.2.2	defun hasFullSignature	134
5.2.3	defun addEmptyCapsuleIfNecessary	134
5.2.4	defun getTargetFromRhs	135
5.2.5	defun giveFormalParametersValues	135
5.2.6	defun macroExpandInPlace	136
5.2.7	defun macroExpand	136
5.2.8	defun macroExpandList	137
5.2.9	defun compDefineCategory1	137
5.2.10	defun makeCategoryPredicates	138
5.2.11	defun mkCategoryPackage	139
5.2.12	defun mkEvalableCategoryForm	140
5.2.13	defun compDefineCategory2	142
5.2.14	defun compile	145
5.2.15	defun encodeFunctionName	148
5.2.16	defun mkRepetitionAssoc	149
5.2.17	defun splitEncodedFunctionName	149
5.2.18	defun encodeItem	150
5.2.19	defun getCaps	150
5.2.20	defun constructMacro	151
5.2.21	defun spadCompileOrSetq	151
5.2.22	defun compileConstructor	153
5.2.23	defun compileConstructor1	153
5.2.24	defun putInLocalDomainReferences	154
5.2.25	defun getArgumentModeOrMoan	154
5.2.26	defun augLisplibModemapsFromCategory	155
5.2.27	defun mkAlistOfExplicitCategoryOps	156
5.2.28	defun flattenSignatureList	158
5.2.29	defun interactiveModemapForm	158
5.2.30	defun replaceVars	159
5.2.31	defun fixUpPredicate	160
5.2.32	defun orderPredicateItems	161
5.2.33	defun signatureTran	161
5.2.34	defun orderPredTran	162
5.2.35	defun isDomainSubst	164
5.2.36	defun moveORsOutside	165
5.2.37	defun substVars	166
5.2.38	defun modemapPattern	167

5.2.39	defun evalAndRwriteLispForm	168
5.2.40	defun rwriteLispForm	168
5.2.41	defun mkConstructor	168
5.2.42	defun compDefineCategory	169
5.2.43	defun compDefineLisplib	169
5.2.44	defun compileDocumentation	172
5.2.45	defun lisplibDoRename	172
5.2.46	defun initializeLisplib	173
5.2.47	defun writeLib1	174
5.2.48	defun finalizeLisplib	174
5.2.49	defun getConstructorOpsAndAtts	176
5.2.50	defun getCategoryOpsAndAtts	177
5.2.51	defun getSlotFromCategoryForm	177
5.2.52	defun transformOperationAlist	177
5.2.53	defun getFunctorOpsAndAtts	179
5.2.54	defun getSlotFromFunctor	179
5.2.55	defun compMakeCategoryObject	180
5.2.56	defun mergeSignatureAndLocalVarAlists	180
5.2.57	defun lisplibWrite	180
5.2.58	defun compDefineFunctor	181
5.2.59	defun compDefineFunctor1	181
5.2.60	defun augmentLisplibModemapsFromFunctor	189
5.2.61	defun allLASSOCs	190
5.2.62	defun formal2Pattern	190
5.2.63	defun mkDatabasePred	191
5.2.64	defun disallowNilAttribute	191
5.2.65	defun compFunctorBody	191
5.2.66	defun bootStrapError	192
5.2.67	defun reportOnFunctorCompilation	192
5.2.68	defun displayMissingFunctions	193
5.2.69	defun makeFunctorArgumentParameters	194
5.2.70	defun genDomainViewList0	196
5.2.71	defun genDomainViewList	196
5.2.72	defun genDomainView	197
5.2.73	defun genDomainOps	198
5.2.74	defun mkOpVec	199
5.2.75	defun AssocBarGensym	200
5.2.76	defun compDefWhereClause	200
5.2.77	defun orderByDependency	202
5.3	Code optimization routines	203
5.3.1	defun optimizeFunctionDef	203
5.3.2	defun optimize	205
5.3.3	defun optXLAMCond	206
5.3.4	defun optCONDtail	206
5.3.5	defvar \$BasicPredicates	207
5.3.6	defun optPredicateIfTrue	207

5.3.7	defun optIF2COND	207
5.3.8	defun subrname	208
5.3.9	Special case optimizers	208
5.3.10	defplist optCall	209
5.3.11	defun Optimize “call” expressions	209
5.3.12	defun optPackageCall	210
5.3.13	defun optCallSpecially	211
5.3.14	defun optSpecialCall	212
5.3.15	defun compileTimeBindingOf	213
5.3.16	defun optCallEval	213
5.3.17	defplist optSEQ	214
5.3.18	defun optSEQ	214
5.3.19	defplist optEQ	215
5.3.20	defun optEQ	216
5.3.21	defplist optMINUS	216
5.3.22	defun optMINUS	216
5.3.23	defplist optQSMINUS	217
5.3.24	defun optQSMINUS	217
5.3.25	defplist opt-	217
5.3.26	defun opt-	218
5.3.27	defplist optLESSP	218
5.3.28	defun optLESSP	218
5.3.29	defplist optSPADCALL	219
5.3.30	defun optSPADCALL	219
5.3.31	defplist optSuchthat	220
5.3.32	defun optSuchthat	220
5.3.33	defplist optCatch	220
5.3.34	defun optCatch	220
5.3.35	defplist optCond	222
5.3.36	defun optCond	222
5.3.37	defun EqualBarGensym	224
5.3.38	defplist optMkRecord	225
5.3.39	defun optMkRecord	225
5.3.40	defplist optRECORDELT	225
5.3.41	defun optRECORDELT	225
5.3.42	defplist optSETRECORDELT	226
5.3.43	defun optSETRECORDELT	226
5.3.44	defplist optRECORDCOPY	227
5.3.45	defun optRECORDCOPY	227
5.4	Functions to manipulate modemaps	228
5.4.1	defun addDomain	228
5.4.2	defun unknownTypeError	229
5.4.3	defun isFunctor	229
5.4.4	defun getDomainsInScope	230
5.4.5	defun putDomainsInScope	230
5.4.6	defun isSuperDomain	231

5.4.7	defun addNewDomain	231
5.4.8	defun augModemapsFromDomain	232
5.4.9	defun augModemapsFromDomain1	232
5.4.10	defun substituteCategoryArguments	233
5.4.11	defun addConstructorModemaps	234
5.4.12	defun getModemap	234
5.4.13	defun getUniqueSignature	235
5.4.14	defun getUniqueModemap	235
5.4.15	defun getModemapList	235
5.4.16	defun getModemapListFromDomain	236
5.4.17	defun domainMember	236
5.4.18	defun augModemapsFromCategory	237
5.4.19	defun addEltModemap	237
5.4.20	defun mkNewModemapList	238
5.4.21	defun insertModemap	239
5.4.22	defun mergeModemap	239
5.4.23	defun TruthP	241
5.4.24	defun evalAndSub	241
5.4.25	defun getOperationAlist	242
5.4.26	defvar \$FormalMapVariableList	242
5.4.27	defun substNames	243
5.4.28	defun augModemapsFromCategoryRep	243
5.5	Maintaining Modemaps	245
5.5.1	defun addModemapKnown	245
5.5.2	defun addModemap	245
5.5.3	defun addModemap0	246
5.5.4	defun addModemap1	246
5.6	Indirect called comp routines	247
5.6.1	defplist compAdd plist	247
5.6.2	defun compAdd	247
5.6.3	defun compTuple2Record	249
5.6.4	defplist compCapsule plist	250
5.6.5	defun compCapsule	250
5.6.6	defun compCapsuleInner	250
5.6.7	defun processFunctor	251
5.6.8	defun compCapsuleItems	252
5.6.9	defun compSingleCapsuleItem	252
5.6.10	defun doIt	253
5.6.11	defun doItIf	257
5.6.12	defun isMacro	259
5.6.13	defplist compCase plist	259
5.6.14	defun compCase	259
5.6.15	defun compCase1	260
5.6.16	defplist compCat plist	261
5.6.17	defplist compCat plist	261
5.6.18	defplist compCat plist	261

5.6.19	defun compCat	261
5.6.20	defplist compCategory plist	262
5.6.21	defun compCategory	262
5.6.22	defun compCategoryItem	263
5.6.23	defun mkExplicitCategoryFunction	265
5.6.24	defun mustInstantiate	266
5.6.25	defun wrapDomainSub	266
5.6.26	defplist compColon plist	266
5.6.27	defun compColon	267
5.6.28	defun makeCategoryForm	270
5.6.29	defplist compCons plist	270
5.6.30	defun compCons	270
5.6.31	defun compCons1	271
5.6.32	defplist compConstruct plist	272
5.6.33	defun compConstruct	272
5.6.34	defplist compConstructorCategory plist	273
5.6.35	defplist compConstructorCategory plist	273
5.6.36	defplist compConstructorCategory plist	273
5.6.37	defplist compConstructorCategory plist	273
5.6.38	defun compConstructorCategory	274
5.6.39	defplist compDefine plist	274
5.6.40	defun compDefine	274
5.6.41	defun compDefine1	275
5.6.42	defun getAbbreviation	277
5.6.43	defun mkAbbrev	277
5.6.44	defun addSuffix	278
5.6.45	defun alistSize	278
5.6.46	defun getSignatureFromMode	278
5.6.47	defun compInternalFunction	279
5.6.48	defun compDefineCapsuleFunction	280
5.6.49	defun compileCases	283
5.6.50	defun getSpecialCaseAssoc	285
5.6.51	defun addArgumentConditions	285
5.6.52	defun compArgumentConditions	286
5.6.53	defun stripOffSubdomainConditions	287
5.6.54	defun stripOffArgumentConditions	287
5.6.55	defun getSignature	288
5.6.56	defun checkAndDeclare	289
5.6.57	defun hasSigInTargetCategory	290
5.6.58	defun getArgumentMode	291
5.6.59	defplist compElt plist	291
5.6.60	defun compElt	292
5.6.61	defplist compExit plist	293
5.6.62	defun compExit	293
5.6.63	defplist compHas plist	294
5.6.64	defun compHas	294

5.6.65	defun compHasFormat	295
5.6.66	defun mkList	296
5.6.67	defplist compIf plist	296
5.6.68	defun compIf	296
5.6.69	defun compFromIf	297
5.6.70	defun canReturn	298
5.6.71	defun compBoolean	300
5.6.72	defun getSuccessEnvironment	300
5.6.73	defun getInverseEnvironment	301
5.6.74	defun getUnionMode	303
5.6.75	defun isUnionMode	303
5.6.76	defplist compImport plist	303
5.6.77	defun compImport	304
5.6.78	defplist compIs plist	304
5.6.79	defun compIs	304
5.6.80	defplist compJoin plist	305
5.6.81	defun compJoin	305
5.6.82	defun compForMode	307
5.6.83	defplist compLambda plist	307
5.6.84	defun compLambda	307
5.6.85	defplist compLeave plist	308
5.6.86	defun compLeave	308
5.6.87	defplist compMacro plist	309
5.6.88	defun compMacro	309
5.6.89	defplist compPretend plist	310
5.6.90	defun compPretend	310
5.6.91	defplist compQuote plist	311
5.6.92	defun compQuote	311
5.6.93	defplist compReduce plist	312
5.6.94	defun compReduce	312
5.6.95	defun compReduce1	312
5.6.96	defplist compRepeatOrCollect plist	314
5.6.97	defplist compRepeatOrCollect plist	314
5.6.98	defun compRepeatOrCollect	314
5.6.99	defplist compReturn plist	316
5.6.100	defun compReturn	317
5.6.101	defplist compSeq plist	318
5.6.102	defun compSeq	318
5.6.103	defun compSeq1	318
5.6.104	defun replaceExitEtc	319
5.6.105	defun convertOrCroak	320
5.6.106	defun compSeqItem	320
5.6.107	defplist compSetq plist	320
5.6.108	defplist compSetq plist	321
5.6.109	defun compSetq	321
5.6.110	defun compSetq1	321

5.6.111	defun uncons	322
5.6.112	defun setqMultiple	322
5.6.113	defun setqMultipleExplicit	324
5.6.114	defun setqSetelt	326
5.6.115	defun setqSingle	326
5.6.116	defun assignError	328
5.6.117	defun outputComp	328
5.6.118	defun maxSuperType	329
5.6.119	defun isDomainForm	329
5.6.120	defun isDomainConstructorForm	330
5.6.121	defplist compString plist	330
5.6.122	defun compString	331
5.6.123	defplist compSubDomain plist	331
5.6.124	defun compSubDomain	331
5.6.125	defun compSubDomain1	332
5.6.126	defun lispize	333
5.6.127	defplist compSubsetCategory plist	333
5.6.128	defun compSubsetCategory	333
5.6.129	defplist compSuchthat plist	334
5.6.130	defun compSuchthat	334
5.6.131	defplist compVector plist	335
5.6.132	defun compVector	335
5.6.133	defplist compWhere plist	336
5.6.134	defun compWhere	336
5.7	Functions for coercion	337
5.7.1	defun coerce	337
5.7.2	defun coerceEasy	338
5.7.3	defun coerceSubset	338
5.7.4	defun coerceHard	339
5.7.5	defun coerceExtraHard	340
5.7.6	defun hasType	341
5.7.7	defun coerceable	341
5.7.8	defun coerceExit	342
5.7.9	defplist compAtSign plist	342
5.7.10	defun compAtSign	343
5.7.11	defplist compCoerce plist	343
5.7.12	defun compCoerce	343
5.7.13	defun compCoerce1	344
5.7.14	defun coerceByModemap	345
5.7.15	defun autoCoerceByModemap	345
5.7.16	defun resolve	347
5.7.17	defun mkUnion	347
5.7.18	defun This orders Unions	348
5.7.19	defun modeEqualSubst	348
5.7.20	compilerDoitWithScreenedLisplib	349

6	Post Transformers	351
6.1	Direct called postparse routines	351
6.1.1	defun postTransform	351
6.1.2	defun postTran	352
6.1.3	defun postOp	353
6.1.4	defun postAtom	353
6.1.5	defun postTranList	354
6.1.6	defun postScriptsForm	354
6.1.7	defun postTranScripts	354
6.1.8	defun postTransformCheck	355
6.1.9	defun postcheck	355
6.1.10	defun postError	356
6.1.11	defun postForm	356
6.2	Indirect called postparse routines	357
6.2.1	defplist postAdd plist	358
6.2.2	defun postAdd	358
6.2.3	defun postCapsule	359
6.2.4	defun postBlockItemList	359
6.2.5	defun postBlockItem	360
6.2.6	defplist postAtSign plist	360
6.2.7	defun postAtSign	361
6.2.8	defun postType	361
6.2.9	defplist postBigFloat plist	361
6.2.10	defun postBigFloat	362
6.2.11	defplist postBlock plist	362
6.2.12	defun postBlock	362
6.2.13	defplist postCategory plist	363
6.2.14	defun postCategory	363
6.2.15	defun postCollect,finish	364
6.2.16	defun postMakeCons	364
6.2.17	defplist postCollect plist	365
6.2.18	defun postCollect	365
6.2.19	defun postIteratorList	366
6.2.20	defplist postColon plist	366
6.2.21	defun postColon	367
6.2.22	defplist postColonColon plist	367
6.2.23	defun postColonColon	367
6.2.24	defplist postComma plist	368
6.2.25	defun postComma	368
6.2.26	defun comma2Tuple	368
6.2.27	defun postFlatten	368
6.2.28	defplist postConstruct plist	369
6.2.29	defun postConstruct	369
6.2.30	defun postTranSegment	370
6.2.31	defplist postDef plist	370
6.2.32	defun postDef	370

6.2.33	defun postDefArgs	372
6.2.34	defplist postExit plist	373
6.2.35	defun postExit	373
6.2.36	defplist postIf plist	373
6.2.37	defun postIf	373
6.2.38	defplist postin plist	374
6.2.39	defun postin	374
6.2.40	defun postInSeq	374
6.2.41	defplist postIn plist	375
6.2.42	defun postIn	375
6.2.43	defplist postJoin plist	375
6.2.44	defun postJoin	376
6.2.45	defplist postMapping plist	376
6.2.46	defun postMapping	376
6.2.47	defplist postMDef plist	377
6.2.48	defun postMDef	377
6.2.49	defplist postPretend plist	378
6.2.50	defun postPretend	378
6.2.51	defplist postQUOTE plist	379
6.2.52	defun postQUOTE	379
6.2.53	defplist postReduce plist	379
6.2.54	defun postReduce	379
6.2.55	defplist postRepeat plist	380
6.2.56	defun postRepeat	380
6.2.57	defplist postScripts plist	380
6.2.58	defun postScripts	381
6.2.59	defplist postSemiColon plist	381
6.2.60	defun postSemiColon	381
6.2.61	defun postFlattenLeft	381
6.2.62	defplist postSignature plist	382
6.2.63	defun postSignature	382
6.2.64	defun removeSuperfluousMapping	383
6.2.65	defun killColons	383
6.2.66	defplist postSlash plist	383
6.2.67	defun postSlash	383
6.2.68	defplist postTuple plist	384
6.2.69	defun postTuple	384
6.2.70	defplist postTupleCollect plist	384
6.2.71	defun postTupleCollect	385
6.2.72	defplist postWhere plist	385
6.2.73	defun postWhere	385
6.2.74	defplist postWith plist	386
6.2.75	defun postWith	386
6.3	Support routines	386
6.3.1	defun setDefOp	386
6.3.2	defun aplTran	387

6.3.3	defun aplTran1	387
6.3.4	defun aplTranList	389
6.3.5	defun hasAplExtension	389
6.3.6	defun deepestExpression	390
6.3.7	defun containsBang	390
6.3.8	defun getScriptName	391
6.3.9	defun decodeScripts	391
7	DEF forms	393
7.0.10	defvar \$defstack	393
7.0.11	defvar \$is-spill	393
7.0.12	defvar \$is-spill-list	393
7.0.13	defvar \$vl	394
7.0.14	defvar \$is-gensymlist	394
7.0.15	defvar \$initial-gensym	394
7.0.16	defvar \$is-eqlist	394
7.0.17	defun hackforis	394
7.0.18	defun hackforis1	395
7.0.19	defun unTuple	395
7.0.20	defun errhuh	395
8	PARSE forms	397
8.1	The original meta specification	397
8.2	The PARSE code	402
8.2.1	defvar \$tmptok	402
8.2.2	defvar \$tok	402
8.2.3	defvar \$ParseMode	403
8.2.4	defvar \$definition-name	403
8.2.5	defvar \$lablasoc	403
8.2.6	defun PARSE-NewExpr	403
8.2.7	defun PARSE-Command	404
8.2.8	defun PARSE-SpecialKeyWord	404
8.2.9	defun PARSE-SpecialCommand	405
8.2.10	defun PARSE-TokenCommandTail	405
8.2.11	defun PARSE-TokenOption	406
8.2.12	defun PARSE-TokenList	406
8.2.13	defun PARSE-CommandTail	407
8.2.14	defun PARSE-PrimaryOrQM	407
8.2.15	defun PARSE-Option	408
8.2.16	defun PARSE-Statement	408
8.2.17	defun PARSE-InfixWith	409
8.2.18	defun PARSE-With	409
8.2.19	defun PARSE-Category	409
8.2.20	defun PARSE-Expression	411
8.2.21	defun PARSE-Import	411
8.2.22	defun PARSE-Expr	412

8.2.23	defun PARSE-LedPart	412
8.2.24	defun PARSE-NudPart	412
8.2.25	defun PARSE-Operation	413
8.2.26	defun PARSE-leftBindingPowerOf	413
8.2.27	defun PARSE-rightBindingPowerOf	414
8.2.28	defun PARSE-getSemanticForm	414
8.2.29	defun PARSE-Prefix	414
8.2.30	defun PARSE-Infix	415
8.2.31	defun PARSE-TokTail	416
8.2.32	defun PARSE-Qualification	416
8.2.33	defun PARSE-Reduction	417
8.2.34	defun PARSE-ReductionOp	417
8.2.35	defun PARSE-Form	417
8.2.36	defun PARSE-Application	418
8.2.37	defun PARSE-Label	419
8.2.38	defun PARSE-Selector	419
8.2.39	defun PARSE-PrimaryNoFloat	420
8.2.40	defun PARSE-Primary	420
8.2.41	defun PARSE-Primary1	420
8.2.42	defun PARSE-Float	421
8.2.43	defun PARSE-FloatBase	422
8.2.44	defun PARSE-FloatBasePart	422
8.2.45	defun PARSE-FloatExponent	423
8.2.46	defun PARSE-Enclosure	424
8.2.47	defun PARSE-IntegerTok	424
8.2.48	defun PARSE-FormalParameter	425
8.2.49	defun PARSE-FormalParameterTok	425
8.2.50	defun PARSE-Quad	425
8.2.51	defun PARSE-String	425
8.2.52	defun PARSE-VarForm	426
8.2.53	defun PARSE-Scripts	426
8.2.54	defun PARSE-ScriptItem	427
8.2.55	defun PARSE-Name	427
8.2.56	defun PARSE-Data	428
8.2.57	defun PARSE-Sexpr	428
8.2.58	defun PARSE-Sexpr1	428
8.2.59	defun PARSE-NBGlyphTok	429
8.2.60	defun PARSE-GlyphTok	430
8.2.61	defun PARSE-AnyId	430
8.2.62	defun PARSE-Sequence	431
8.2.63	defun PARSE-Sequence1	431
8.2.64	defun PARSE-OpenBracket	432
8.2.65	defun PARSE-OpenBrace	432
8.2.66	defun PARSE-IteratorTail	433
8.2.67	defun PARSE-Iterator	433
8.2.68	The PARSE implicit routines	434

8.2.69	defun PARSE-Suffix	434
8.2.70	defun PARSE-SemiColon	435
8.2.71	defun PARSE-Return	435
8.2.72	defun PARSE-Exit	435
8.2.73	defun PARSE-Leave	436
8.2.74	defun PARSE-Seg	436
8.2.75	defun PARSE-Conditional	437
8.2.76	defun PARSE-ElseClause	437
8.2.77	defun PARSE-Loop	438
8.2.78	defun PARSE-LabelExpr	438
8.2.79	defun PARSE-FloatTok	439
8.3	The PARSE support routines	439
8.3.1	String grabbing	440
8.3.2	defun match-string	440
8.3.3	defun skip-blanks	440
8.3.4	defun token-lookahead-type	441
8.3.5	defun match-advance-string	441
8.3.6	defun initial-substring-p	442
8.3.7	defun quote-if-string	442
8.3.8	defun escape-keywords	443
8.3.9	defun isTokenDelimiter	443
8.3.10	defun underscore	444
8.3.11	Token Handling	444
8.3.12	defun getToken	444
8.3.13	defun unget-tokens	444
8.3.14	defun match-current-token	445
8.3.15	defun match-token	446
8.3.16	defun match-next-token	446
8.3.17	defun current-symbol	446
8.3.18	defun make-symbol-of	446
8.3.19	defun current-token	447
8.3.20	defun try-get-token	447
8.3.21	defun next-token	448
8.3.22	defun advance-token	448
8.3.23	defvar \$XTokenReader	449
8.3.24	defun get-token	449
8.3.25	Character handling	449
8.3.26	defun current-char	449
8.3.27	defun next-char	449
8.3.28	defun char-eq	450
8.3.29	defun char-ne	450
8.3.30	Error handling	450
8.3.31	defvar \$meta-error-handler	450
8.3.32	defun meta-syntax-error	451
8.3.33	Floating Point Support	451
8.3.34	defun floatexpid	451

8.3.35	Dollar Translation	451
8.3.36	defun dollarTran	451
8.3.37	Applying metagrammatical elements of a production (e.g., Star). . . .	452
8.3.38	defmacro Bang	452
8.3.39	defmacro must	452
8.3.40	defun action	453
8.3.41	defun optional	453
8.3.42	defmacro star	453
8.3.43	Stacking and retrieving reductions of rules.	454
8.3.44	defvar \$reduce-stack	454
8.3.45	defmacro reduce-stack-clear	454
8.3.46	defun push-reduction	454
9	Utility Functions	455
9.0.47	defun translablel	455
9.0.48	defun translablel1	455
9.0.49	defun displayPreCompilationErrors	456
9.0.50	defun bumperrorcount	457
9.0.51	defun parseTranCheckForRecord	457
9.0.52	defun new2OldLisp	458
9.0.53	defun makeSimplePredicateOrNil	458
9.0.54	defun parse-spadstring	458
9.0.55	defun parse-string	459
9.0.56	defun parse-identifier	459
9.0.57	defun parse-number	460
9.0.58	defun parse-keyword	460
9.0.59	defun parse-argument-designator	460
9.0.60	defun print-package	461
9.0.61	defun checkWarning	461
9.0.62	defun tuple2List	461
9.0.63	defmacro pop-stack-1	462
9.0.64	defmacro pop-stack-2	463
9.0.65	defmacro pop-stack-3	463
9.0.66	defmacro pop-stack-4	463
9.0.67	defmacro nth-stack	464
9.0.68	defun Pop-Reduction	464
9.0.69	defun addclose	464
9.0.70	defun blankp	465
9.0.71	defun drop	465
9.0.72	defun escaped	465
9.0.73	defvar \$comblocklist	465
9.0.74	defun fincomblock	466
9.0.75	defun indent-pos	466
9.0.76	defun infixtok	467
9.0.77	defun is-console	467
9.0.78	defun next-tab-loc	467

9.0.79	defun nonblankloc	468
9.0.80	defun parseprint	468
9.0.81	defun skip-to-endif	468
10	The Compiler	469
10.1	Compiling EQ.spad	469
10.1.1	The top level compiler command	472
10.1.2	The Spad compiler top level function	474
10.1.3	defun compilerDoit	478
10.1.4	defun /RQ,LIB	479
10.1.5	defun /rf-1	480
10.1.6	defun spad	489
10.1.7	defun Interpreter interface to the compiler	490
10.1.8	defun print-defun	493
10.1.9	defun def-rename	493
10.1.10	defun def-rename1	494
10.1.11	defun compTopLevel	494
10.1.12	defun compOrCroak	496
10.1.13	defun compOrCroak1	496
10.1.14	defun comp	497
10.1.15	defun compNoStacking	498
10.1.16	defun compNoStacking1	498
10.1.17	defun comp2	499
10.1.18	defun comp3	500
10.1.19	defun compTypeOf	502
10.1.20	defun compColonInside	502
10.1.21	defun compAtom	503
10.1.22	defun convert	504
10.1.23	defun primitiveType	504
10.1.24	defun compSymbol	505
10.1.25	defun compList	506
10.1.26	defun compExpression	507
10.1.27	defun compForm	507
10.1.28	defun compForm1	508
10.1.29	defun getFormModemaps	510
10.1.30	defun eltModemapFilter	511
10.1.31	defun seteltModemapFilter	512
10.1.32	defun compExpressionList	512
10.1.33	defun compForm2	513
10.1.34	defun compForm3	515
10.1.35	defun compFormPartiallyBottomUp	516
10.1.36	defun compFormMatch	516
10.1.37	defun compUniquely	516
10.1.38	defun compArgumentsAndTryAgain	517
10.1.39	defun compWithMappingMode	517
10.1.40	defun compWithMappingMode1	518

10.1.41 defun extractCodeAndConstructTriple	523
10.1.42 defun hasFormalMapVariable	524
10.1.43 defun argsToSig	524
10.1.44 defun compMakeDeclaration	525
10.1.45 defun modifyModeStack	525
10.1.46 defun Create a list of unbound symbols	526
10.1.47 defun compOrCroak1,compactify	527
10.1.48 defun Compiler/Interpreter interface	527
10.1.49 defun compileSpadLispCmd	528
10.1.50 defun recompile-lib-file-if-necessary	529
10.1.51 defun spad-fixed-arg	529
10.1.52 defun compile-lib-file	530
10.1.53 defun compileFileQuietly	530
10.1.54 defvar \$byConstructors	531
10.1.55 defvar \$constructorsSeen	531
11 Level 1	533
11.0.56 defvar \$current-fragment	533
11.0.57 defun read-a-line	533
12 Level 0	535
12.1 Line Handling	535
12.1.1 Line Buffer	535
12.1.2 defstruct \$line	535
12.1.3 defvar \$current-line	536
12.1.4 defmacro line-clear	536
12.1.5 defun line-print	536
12.1.6 defun line-at-end-p	536
12.1.7 defun line-past-end-p	537
12.1.8 defun line-next-char	537
12.1.9 defun line-advance-char	537
12.1.10 defun line-current-segment	538
12.1.11 defun line-new-line	538
12.1.12 defun next-line	538
12.1.13 defun Advance-Char	539
12.1.14 defun storeblanks	539
12.1.15 defun initial-substring	539
12.1.16 defun get-a-line	540
13 The Chunks	541
14 Index	557

New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation's website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we've broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We've also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I'm looking forward to future milestones.

With that in mind I've introduced the theme of the "30 year horizon". We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How can we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The "30 year horizon" is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))

0.1 Makefile

This book is actually a literate program[2] and contains executable source code. In particular, the Makefile for this book is part of the source of the book and is included below. Axiom uses the “noweb” literate programming system by Norman Ramsey[6].

Chapter 1

Overview

The Spad language is a mathematically oriented language intended for writing computational mathematics. It derives its logical structure from abstract algebra. It features ideas that are still not available in general purpose programming languages, such as selecting overloaded procedures based on the return type as well as the types of the arguments.

The Spad language is heavily influenced by Barbara Liskov's work. It features encapsulation (aka objects), inheritance, and overloading. It has categories which are defined by the exports. Categories are parameterized functors that take arguments which define their behavior.

More details on the language and its high level concepts is available in the Programmers Guide, Volume 3.

The Spad compiler accepts the Spad language and generates a set of files used by the interpreter, detailed in Volume 5.

The compiler does not produce stand-alone executable code. It assumes that it will run inside the interpreter and that the code it generates will be loaded into the interpreter.

Some of the routines are common to both the compiler and the interpreter. Where this happens we have favored the interpreter volume (Volume 5) as the official source location. In each case we will make reference to that volume and the code in it. Thus, the compiler volume should be considered as an extension of the interpreter document.

This volume will go into painful detail of every aspect of compiling Spad code. We will start by defining the input to, and output from the compiler so we know what we are trying to achieve.

Next we will look at the top level data structures used by the compiler. Unfortunately, the compiler uses a large number of "global variables" to pass information and alter control flow. Some of these are used by many routines and some of these are very local to a small subset or a recursion. We will cover the minor ones as they arise.

Next we examine the Pratt parser idea and the Led and Nud concepts, which is used to drive the low level parsing.

Following that we journey deep into the code, trying our best not to get lost in the details. The code is introduced based on “motivation” rather than in strict execution order or related concept order. We do this to try to make the compiler a “readable novel” rather than a mud-march through the code. The goal is to keep the reader’s interest while trying to be exact. Sometimes this will require detours to discuss subtopics.

“Motivating” a piece of software is a not-very-well established form of narrative writing so we assume your forgiveness if we get it wrong. Worse yet, some of the pieces of the system are “legacy”, in that they are no longer used and should be removed. Other parts of the system may have very weak descriptions because we simply do not understand them either. Since this is a living document and the code for the system is actually the code you are reading we will expand parts as we go.

1.1 The Input

```
)abbrev domain EQ Equation
--FOR THE BENEFIT OF LIBAXO GENERATION
++ Author: Stephen M. Watt, enhancements by Johannes Grabmeier
++ Date Created: April 1985
++ Date Last Updated: June 3, 1991; September 2, 1992
++ Basic Operations: =
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: equation
++ Examples:
++ References:
++ Description:
++ Equations as mathematical objects. All properties of the basis domain,
++ e.g. being an abelian group are carried over the equation domain, by
++ performing the structural operations on the left and on the
++ right hand side.
-- The interpreter translates "=" to "equation". Otherwise, it will
-- find a modemap for "=" in the domain of the arguments.
```

```
Equation(S: Type): public == private where
  Ex ==> OutputForm
  public ==> Type with
    "=": (S, S) -> $
      ++ a=b creates an equation.
    equation: (S, S) -> $
      ++ equation(a,b) creates an equation.
    swap: $ -> $
      ++ swap(eq) interchanges left and right hand side of equation eq.
    lhs: $ -> S
      ++ lhs(eqn) returns the left hand side of equation eqn.
    rhs: $ -> S
      ++ rhs(eqn) returns the right hand side of equation eqn.
```

```

map: (S -> S, $) -> $
  ++ map(f,eqn) constructs a new equation by applying f to both
  ++ sides of eqn.
if S has InnerEvalable(Symbol,S) then
  InnerEvalable(Symbol,S)
if S has SetCategory then
  SetCategory
  CoercibleTo Boolean
  if S has Evalable(S) then
    eval: ($, $) -> $
      ++ eval(eqn, x=f) replaces x by f in equation eqn.
    eval: ($, List $) -> $
      ++ eval(eqn, [x1=v1, ... xn=vn]) replaces xi by vi in equation eqn.
if S has AbelianSemiGroup then
  AbelianSemiGroup
  "+": (S, $) -> $
    ++ x+eqn produces a new equation by adding x to both sides of
    ++ equation eqn.
  "+": ($, S) -> $
    ++ eqn+x produces a new equation by adding x to both sides of
    ++ equation eqn.
if S has AbelianGroup then
  AbelianGroup
  leftZero : $ -> $
    ++ leftZero(eq) subtracts the left hand side.
  rightZero : $ -> $
    ++ rightZero(eq) subtracts the right hand side.
  "-": (S, $) -> $
    ++ x-eqn produces a new equation by subtracting both sides of
    ++ equation eqn from x.
  "-": ($, S) -> $
    ++ eqn-x produces a new equation by subtracting x from
    ++ both sides of equation eqn.
if S has SemiGroup then
  SemiGroup
  "*": (S, $) -> $
    ++ x*eqn produces a new equation by multiplying both sides of
    ++ equation eqn by x.
  "*": ($, S) -> $
    ++ eqn*x produces a new equation by multiplying both sides of
    ++ equation eqn by x.
if S has Monoid then
  Monoid
  leftOne : $ -> Union($,"failed")
    ++ leftOne(eq) divides by the left hand side, if possible.
  rightOne : $ -> Union($,"failed")
    ++ rightOne(eq) divides by the right hand side, if possible.
if S has Group then
  Group
  leftOne : $ -> Union($,"failed")

```

```

    ++ leftOne(eq) divides by the left hand side.
    rightOne : $ -> Union($,"failed")
    ++ rightOne(eq) divides by the right hand side.
if S has Ring then
  Ring
  BiModule(S,S)
if S has CommutativeRing then
  Module(S)
  --Algebra(S)
if S has IntegralDomain then
  factorAndSplit : $ -> List $
    ++ factorAndSplit(eq) make the right hand side 0 and
    ++ factors the new left hand side. Each factor is equated
    ++ to 0 and put into the resulting list without repetitions.
if S has PartialDifferentialRing(Symbol) then
  PartialDifferentialRing(Symbol)
if S has Field then
  VectorSpace(S)
  "/" : ($, $) -> $
    ++ e1/e2 produces a new equation by dividing the left and right
    ++ hand sides of equations e1 and e2.
  inv : $ -> $
    ++ inv(x) returns the multiplicative inverse of x.
if S has ExpressionSpace then
  subst : ($, $) -> $
    ++ subst(eq1,eq2) substitutes eq2 into both sides of eq1
    ++ the lhs of eq2 should be a kernel

private ==> add
  Rep := Record(lhs: S, rhs: S)
  eq1,eq2: $
  s : S
  if S has IntegralDomain then
    factorAndSplit eq ==
      (S has factor : S -> Factored S) =>
        eq0 := rightZero eq
        [equation(rcf.factor,0) for rcf in factors factor lhs eq0]
        [eq]
  l:S = r:S      == [1, r]
  equation(l, r) == [1, r]    -- hack! See comment above.
  lhs eqn        == eqn.lhs
  rhs eqn        == eqn.rhs
  swap eqn       == [rhs eqn, lhs eqn]
  map(fn, eqn)   == equation(fn(eqn.lhs), fn(eqn.rhs))

if S has InnerEvalable(Symbol,S) then
  s:Symbol
  ls:List Symbol
  x:S
  lx:List S

```

```

eval(eqn,s,x) == eval(eqn.lhs,s,x) = eval(eqn.rhs,s,x)
eval(eqn,ls,lx) == eval(eqn.lhs,ls,lx) = eval(eqn.rhs,ls,lx)
if S has Evaluable(S) then
  eval(eqn1:$, eqn2:$):$ ==
    eval(eqn1.lhs, eqn2 pretend Equation S) =
      eval(eqn1.rhs, eqn2 pretend Equation S)
  eval(eqn1:$, leqn2:List $):$ ==
    eval(eqn1.lhs, leqn2 pretend List Equation S) =
      eval(eqn1.rhs, leqn2 pretend List Equation S)
if S has SetCategory then
  eq1 = eq2 == (eq1.lhs = eq2.lhs)@Boolean and
    (eq1.rhs = eq2.rhs)@Boolean
  coerce(eqn:$):Ex == eqn.lhs::Ex = eqn.rhs::Ex
  coerce(eqn:$):Boolean == eqn.lhs = eqn.rhs
if S has AbelianSemiGroup then
  eq1 + eq2 == eq1.lhs + eq2.lhs = eq1.rhs + eq2.rhs
  s + eq2 == [s,s] + eq2
  eq1 + s == eq1 + [s,s]
if S has AbelianGroup then
  - eq == (- lhs eq) = (-rhs eq)
  s - eq2 == [s,s] - eq2
  eq1 - s == eq1 - [s,s]
  leftZero eq == 0 = rhs eq - lhs eq
  rightZero eq == lhs eq - rhs eq = 0
  0 == equation(0$S,0$S)
  eq1 - eq2 == eq1.lhs - eq2.lhs = eq1.rhs - eq2.rhs
if S has SemiGroup then
  eq1:$ * eq2:$ == eq1.lhs * eq2.lhs = eq1.rhs * eq2.rhs
  1:S * eqn:$ == 1 * eqn.lhs = 1 * eqn.rhs
  1:S * eqn:$ == 1 * eqn.lhs = 1 * eqn.rhs
  eqn:$ * 1:S == eqn.lhs * 1 = eqn.rhs * 1
  -- We have to be a bit careful here: raising to a +ve integer is OK
  -- (since it's the equivalent of repeated multiplication)
  -- but other powers may cause contradictions
  -- Watch what else you add here! JHD 2/Aug 1990
if S has Monoid then
  1 == equation(1$S,1$S)
  recip eq ==
    (lh := recip lhs eq) case "failed" => "failed"
    (rh := recip rhs eq) case "failed" => "failed"
    [lh :: S, rh :: S]
  leftOne eq ==
    (re := recip lhs eq) case "failed" => "failed"
    1 = rhs eq * re
  rightOne eq ==
    (re := recip rhs eq) case "failed" => "failed"
    lhs eq * re = 1
if S has Group then
  inv eq == [inv lhs eq, inv rhs eq]
  leftOne eq == 1 = rhs eq * inv rhs eq

```

```

    rightOne eq == lhs eq * inv rhs eq = 1
  if S has Ring then
    characteristic() == characteristic()$S
    i:Integer * eq:$ == (i::S) * eq
  if S has IntegralDomain then
    factorAndSplit eq ==
      (S has factor : S -> Factored S) =>
        eq0 := rightZero eq
        [equation(rcf.factor,0) for rcf in factors factor lhs eq0]
      (S has Polynomial Integer) =>
        eq0 := rightZero eq
        MF ==> MultivariateFactorize(Symbol, IndexedExponents Symbol, _
          Integer, Polynomial Integer)
        p : Polynomial Integer := (lhs eq0) pretend Polynomial Integer
        [equation((rcf.factor) pretend S,0) for rcf in factors factor(p)$MF]
      [eq]
  if S has PartialDifferentialRing(Symbol) then
    differentiate(eq:$, sym:Symbol):$ ==
      [differentiate(lhs eq, sym), differentiate(rhs eq, sym)]
  if S has Field then
    dimension() == 2 :: CardinalNumber
    eq1:$ / eq2:$ == eq1.lhs / eq2.lhs = eq1.rhs / eq2.rhs
    inv eq == [inv lhs eq, inv rhs eq]
  if S has ExpressionSpace then
    subst(eq1,eq2) ==
      eq3 := eq2 pretend Equation S
      [subst(lhs eq1,eq3),subst(rhs eq1,eq3)]

```

1.2 The Output, the EQ.nrlib directory

The Spad compiler generates several files in a directory named after the input abbreviation. The input file contains an abbreviation line:

```
)abbrev domain EQ Equation
```

for each category, domain, or package. The abbreviation line has 3 parts.

- one of “category”, “domain”, or “package”
- the abbreviation for this domain (8 Uppercase Characters maximum)
- the name of this domain

Since the abbreviation for the Equation domain is EQ, the compiler will put all of its output into a subdirectory called “EQ.nrlib”. The “nrlib” is a port of a very old VMLisp file format, simulated with directories.

For the EQ input file, the compiler will create the following output files, each of which we will explain in detail below.

```
/research/test/int/algebra/EQ.nrlib:
used 216 available 4992900
drwxr-xr-x    2 root root  4096 2010-12-09 11:20 .
drwxr-xr-x 1259 root root 73728 2010-12-09 11:43 ..
-rw-r--r--    1 root root 19228 2010-12-09 11:20 code.lsp
-rw-r--r--    1 root root 34074 2010-12-09 11:20 code.o
-rw-r--r--    1 root root 13543 2010-12-09 11:20 EQ.fn
-rw-r--r--    1 root root 19228 2010-12-09 11:20 EQ.lsp
-rw-r--r--    1 root root 36148 2010-12-09 11:20 index.kaf
-rw-r--r--    1 root root  6236 2010-12-09 11:20 info
```

1.3 The code.lsp and EQ.lsp files

```
(/VERSIONCHECK 2)
```

```
(DEFUN |EQ;factorAndSplit;$L;1| (|eq| $)
  (PROG (|eq0| #:G1403 |rcf| #:G1404)
    (RETURN
      (SEQ (COND
        ((|HasSignature| (QREFELT $ 6)
          (LIST '|factor|
            (LIST (LIST '|Factored|
              (|devaluate| (QREFELT $ 6)))
              (|devaluate| (QREFELT $ 6))))))
          (SEQ (LETT |eq0| (SPADCALL |eq| (QREFELT $ 8))
            |EQ;factorAndSplit;$L;1|)
            (EXIT (PROGN
              (LETT #:G1403 NIL |EQ;factorAndSplit;$L;1|)
              (SEQ (LETT |rcf| NIL
                |EQ;factorAndSplit;$L;1|)
                (LETT #:G1404
                  (SPADCALL
                    (SPADCALL
                      (SPADCALL |eq0| (QREFELT $ 9))
                      (QREFELT $ 11))
                      (QREFELT $ 15))
                    |EQ;factorAndSplit;$L;1|)
                  G190
                  (COND
                    ((OR (ATOM #:G1404)
                      (PROGN
                        (LETT |rcf| (CAR #:G1404)
                          |EQ;factorAndSplit;$L;1|)
                          NIL))
                      (GO G191))))
                (GO G191))))
              (GO G191))))
          (GO G191))))))
```



```

(SEQ (EXIT
      (LETT #:G1403
            (CONS
              (SPADCALL (QCAR |rcf|)
                        (|spadConstant| $ 16)
                        (QREFELT $ 17))
              #:G1403)
      |EQ;factorAndSplit;$L;1|)))
(LETT #:G1404 (CDR #:G1404)
      |EQ;factorAndSplit;$L;1|)
(GO G190) G191
(EXIT (NREVERSEO #:G1403))))))
('T (LIST |eq|))))))

(PUT (QUOTE |EQ;=;2S$;2|) (QUOTE |SPADreplace|) (QUOTE CONS))

(DEFUN |EQ;=;2S$;2| (|l| |r| $) (CONS |l| |r|))

(PUT (QUOTE |EQ;equation;2S$;3|) (QUOTE |SPADreplace|) (QUOTE CONS))

(DEFUN |EQ;equation;2S$;3| (|l| |r| $) (CONS |l| |r|))

(PUT (QUOTE |EQ;lhs;$S;4|) (QUOTE |SPADreplace|) (QUOTE QCAR))

(DEFUN |EQ;lhs;$S;4| (|eqn| $) (QCAR |eqn|))

(PUT (QUOTE |EQ;rhs;$S;5|) (QUOTE |SPADreplace|) (QUOTE QCDR))

(DEFUN |EQ;rhs;$S;5| (|eqn| $) (QCDR |eqn|))

(DEFUN |EQ;swap;2$;6| (|eqn| $) (CONS (SPADCALL |eqn| (QREFELT $ 21))
  (SPADCALL |eqn| (QREFELT $ 9))))

(DEFUN |EQ;map;M2$;7| (|fn| |eqn| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) |fn|)
    (SPADCALL (QCDR |eqn|) |fn|)
    (QREFELT $ 17)))

(DEFUN |EQ;eval;$SS$;8| (|eqn| |s| |x| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) |s| |x| (QREFELT $ 26))
    (SPADCALL (QCDR |eqn|) |s| |x| (QREFELT $ 26))
    (QREFELT $ 20)))

(DEFUN |EQ;eval;$LL$;9| (|eqn| |ls| |lx| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) |ls| |lx| (QREFELT $ 30))
    (SPADCALL (QCDR |eqn|) |ls| |lx| (QREFELT $ 30))
    (QREFELT $ 20)))

```

```

(DEFUN |EQ;eval;3$;10| (|eqn1| |eqn2| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn1|) |eqn2| (QREFELT $ 33))
    (SPADCALL (QCDR |eqn1|) |eqn2| (QREFELT $ 33))
    (QREFELT $ 20)))

(DEFUN |EQ;eval;$L$;11| (|eqn1| |leqn2| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn1|) |leqn2| (QREFELT $ 36))
    (SPADCALL (QCDR |eqn1|) |leqn2| (QREFELT $ 36))
    (QREFELT $ 20)))

(DEFUN |EQ;=;2$B;12| (|eq1| |eq2| $)
  (COND
    ((SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 39))
      (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 39)))
    ((QUOTE T) (QUOTE NIL))))

(DEFUN |EQ;coerce;$Of;13| (|eqn| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) (QREFELT $ 42))
    (SPADCALL (QCDR |eqn|) (QREFELT $ 42))
    (QREFELT $ 43)))

(DEFUN |EQ;coerce;$B;14| (|eqn| $)
  (SPADCALL (QCAR |eqn|) (QCDR |eqn|) (QREFELT $ 39)))

(DEFUN |EQ;+;3$;15| (|eq1| |eq2| $)
  (SPADCALL
    (SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 46))
    (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 46))
    (QREFELT $ 20)))

(DEFUN |EQ;+;S2$;16| (|s| |eq2| $)
  (SPADCALL (CONS |s| |s|) |eq2| (QREFELT $ 47)))

(DEFUN |EQ;+;$S$;17| (|eq1| |s| $)
  (SPADCALL |eq1| (CONS |s| |s|) (QREFELT $ 47)))

(DEFUN |EQ;-;2$;18| (|eq| $)
  (SPADCALL
    (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 50))
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 50))
    (QREFELT $ 20)))

(DEFUN |EQ;-;S2$;19| (|s| |eq2| $)
  (SPADCALL (CONS |s| |s|) |eq2| (QREFELT $ 52)))

(DEFUN |EQ;-;$S$;20| (|eq1| |s| $)

```

```

(SPADCALL |eq1| (CONS |s| |s|) (QREFELT $ 52)))

(DEFUN |EQ;leftZero;2$;21| (|eq| $)
  (SPADCALL
    (|spadConstant| $ 16)
    (SPADCALL
      (SPADCALL |eq| (QREFELT $ 21))
      (SPADCALL |eq| (QREFELT $ 9))
      (QREFELT $ 56))
    (QREFELT $ 20)))

(DEFUN |EQ;rightZero;2$;22| (|eq| $)
  (SPADCALL
    (SPADCALL
      (SPADCALL |eq| (QREFELT $ 9))
      (SPADCALL |eq| (QREFELT $ 21))
      (QREFELT $ 56))
    (|spadConstant| $ 16)
    (QREFELT $ 20)))

(DEFUN |EQ;Zero;$;23| ($)
  (SPADCALL (|spadConstant| $ 16) (|spadConstant| $ 16) (QREFELT $ 17)))

(DEFUN |EQ;-;3$;24| (|eq1| |eq2| $)
  (SPADCALL
    (SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 56))
    (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 56))
    (QREFELT $ 20)))

(DEFUN |EQ;*;3$;25| (|eq1| |eq2| $)
  (SPADCALL
    (SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 58))
    (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 58))
    (QREFELT $ 20)))

(DEFUN |EQ;*;S2$;26| (|l| |eqn| $)
  (SPADCALL
    (SPADCALL |l| (QCAR |eqn|) (QREFELT $ 58))
    (SPADCALL |l| (QCDR |eqn|) (QREFELT $ 58))
    (QREFELT $ 20)))

(DEFUN |EQ;*;S2$;27| (|l| |eqn| $)
  (SPADCALL
    (SPADCALL |l| (QCAR |eqn|) (QREFELT $ 58))
    (SPADCALL |l| (QCDR |eqn|) (QREFELT $ 58))
    (QREFELT $ 20)))

(DEFUN |EQ;*;$S$;28| (|eqn| |l| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) |l| (QREFELT $ 58))

```

```

(SPADCALL (QCDR |eqn|) |l| (QREFELT $ 58))
(QREFELT $ 20)))

(DEFUN |EQ;One;$;29| ($)
  (SPADCALL (|spadConstant| $ 62) (|spadConstant| $ 62) (QREFELT $ 17)))

(DEFUN |EQ;recip;$U;30| (|eq| $)
  (PROG (|lh| |rh|)
    (RETURN
      (SEQ
        (LETT |lh|
          (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 65))
          |EQ;recip;$U;30|)
        (EXIT
          (COND
            ((QEQCAR |lh| 1) (CONS 1 "failed"))
            ('T
              (SEQ
                (LETT |rh|
                  (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 65))
                  |EQ;recip;$U;30|)
                (EXIT
                  (COND
                    ((QEQCAR |rh| 1) (CONS 1 "failed"))
                    ('T
                      (CONS 0
                        (CONS (QCDR |lh|) (QCDR |rh|))))))))))))))

(DEFUN |EQ;leftOne;$U;31| (|eq| $)
  (PROG (|re|)
    (RETURN
      (SEQ
        (LETT |re|
          (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 65))
          |EQ;leftOne;$U;31|)
        (EXIT
          (COND
            ((QEQCAR |re| 1) (CONS 1 "failed"))
            ('T
              (CONS 0
                (SPADCALL
                  (|spadConstant| $ 62)
                  (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QCDR |re|) (QREFELT $ 58))
                  (QREFELT $ 20))))))))))

(DEFUN |EQ;rightOne;$U;32| (|eq| $)
  (PROG (|re|)
    (RETURN

```

```

(SEQ
  (LETT |re|
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 65))
    |EQ;rightOne;$U;32|)
  (EXIT
    (COND
      ((QEQCAR |re| 1) (CONS 1 "failed"))
      ('T
        (CONS 0
          (SPADCALL
            (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QCDR |re|) (QREFELT $ 58))
            (|spadConstant| $ 62)
            (QREFELT $ 20))))))))))

(DEFUN |EQ;inv;2$;33| (|eq| $)
  (CONS (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 69))
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 69))))

(DEFUN |EQ;leftOne;$U;34| (|eq| $)
  (CONS 0
    (SPADCALL (|spadConstant| $ 62)
      (SPADCALL (SPADCALL |eq| (QREFELT $ 21))
        (SPADCALL (SPADCALL |eq| (QREFELT $ 21))
          (QREFELT $ 69))
          (QREFELT $ 58))
        (QREFELT $ 20))))))

(DEFUN |EQ;rightOne;$U;35| (|eq| $)
  (CONS 0
    (SPADCALL
      (SPADCALL (SPADCALL |eq| (QREFELT $ 9))
        (SPADCALL (SPADCALL |eq| (QREFELT $ 21))
          (QREFELT $ 69))
          (QREFELT $ 58))
        (|spadConstant| $ 62) (QREFELT $ 20))))))

(DEFUN |EQ;characteristic;Nni;36| ($) (SPADCALL (QREFELT $ 72)))

(DEFUN |EQ;*;I2$;37| (|i| |eq| $)
  (SPADCALL (SPADCALL |i| (QREFELT $ 75)) |eq| (QREFELT $ 60)))

(DEFUN |EQ;factorAndSplit;$L;38| (|eq| $)
  (PROG (#:G1488 #:G1489 |eq0| |p| #:G1490 |rcf| #:G1491)
    (RETURN
      (SEQ (COND
        ((|HasSignature| (QREFELT $ 6)
          (LIST ' |factor|
            (LIST (LIST ' |Factored|
              (|devaluate| (QREFELT $ 6))))

```

```

(|devaluate| (QREFELT $ 6))))))
(SEQ (LETT |eq0| (SPADCALL |eq| (QREFELT $ 8))
|EQ;factorAndSplit;$L;38|)
(EXIT (PROGN
(LETT #:G1488 NIL |EQ;factorAndSplit;$L;38|)
(SEQ (LETT |rcf| NIL
|EQ;factorAndSplit;$L;38|)
(LETT #:G1489
(SPADCALL
(SPADCALL
(SPADCALL |eq0| (QREFELT $ 9))
(QREFELT $ 11))
(QREFELT $ 15))
|EQ;factorAndSplit;$L;38|)
G190
(COND
((OR (ATOM #:G1489)
(PROGN
(LETT |rcf| (CAR #:G1489)
|EQ;factorAndSplit;$L;38|)
NIL))
(GO G191))))
(SEQ (EXIT
(LETT #:G1488
(CONS
(SPADCALL (QCAR |rcf|)
(|spadConstant| $ 16)
(QREFELT $ 17))
#:G1488)
|EQ;factorAndSplit;$L;38|)))
(LETT #:G1489 (CDR #:G1489)
|EQ;factorAndSplit;$L;38|)
(GO G190) G191
(EXIT (NREVERSEO #:G1488))))))
((EQUAL (QREFELT $ 6) (|Polynomial| (|Integer|)))
(SEQ (LETT |eq0| (SPADCALL |eq| (QREFELT $ 8))
|EQ;factorAndSplit;$L;38|)
(LETT |p| (SPADCALL |eq0| (QREFELT $ 9))
|EQ;factorAndSplit;$L;38|)
(EXIT (PROGN
(LETT #:G1490 NIL |EQ;factorAndSplit;$L;38|)
(SEQ (LETT |rcf| NIL
|EQ;factorAndSplit;$L;38|)
(LETT #:G1491
(SPADCALL
(SPADCALL |p| (QREFELT $ 80))
(QREFELT $ 83))
|EQ;factorAndSplit;$L;38|)
G190
(COND

```

```

((OR (ATOM #:G1491)
      (PROGN
        (LETT |rcf| (CAR #:G1491)
          |EQ;factorAndSplit;$L;38|)
          NIL))
      (GO G191)))
(SEQ (EXIT
      (LETT #:G1490
        (CONS
          (SPADCALL (QCAR |rcf|)
            (|spadConstant| $ 16)
            (QREFELT $ 17))
          #:G1490)
        |EQ;factorAndSplit;$L;38|)))
      (LETT #:G1491 (CDR #:G1491)
        |EQ;factorAndSplit;$L;38|)
      (GO G190) G191
      (EXIT (NREVERSEO #:G1490))))))
('T (LIST |eq|))))))

(DEFUN |EQ;differentiate;$S$;39| (|eq| |sym| $)
  (CONS (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) |sym| (QREFELT $ 84))
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) |sym| (QREFELT $ 84))))

(DEFUN |EQ;dimension;Cn;40| ($) (SPADCALL 2 (QREFELT $ 87)))

(DEFUN |EQ;/;3$;41| (|eq1| |eq2| $)
  (SPADCALL (SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 89))
    (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 89))
    (QREFELT $ 20)))

(DEFUN |EQ;inv;2$;42| (|eq| $)
  (CONS (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 69))
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 69))))

(DEFUN |EQ;subst;3$;43| (|eq1| |eq2| $)
  (PROG (|eq3|)
    (RETURN
      (SEQ (LETT |eq3| |eq2| |EQ;subst;3$;43|)
        (EXIT (CONS (SPADCALL (SPADCALL |eq1| (QREFELT $ 9)) |eq3|
          (QREFELT $ 92))
          (SPADCALL (SPADCALL |eq1| (QREFELT $ 21)) |eq3|
            (QREFELT $ 92)))))))

(DEFUN |Equation| (#:G1503)
  (PROG ()
    (RETURN
      (PROG (#:G1504)
        (RETURN

```

```

(COND
  ((LETT #:G1504
    (|lassocShiftWithFunction|
      (LIST (|devaluate| #:G1503))
      (HGET |$ConstructorCache| '|Equation|)
      '|domainEqualList|)
    |Equation|)
  (|CDRwithIncrement| #:G1504))
('T
  (UNWIND-PROTECT
    (PROG1 (|Equation;| #:G1503)
      (LETT #:G1504 T |Equation|))
    (COND
      ((NOT #:G1504) (HREM |$ConstructorCache| '|Equation|)))))))))

(DEFUN |Equation;| (#1|)
  (PROG (DV$1 |dv$| $ #:G1502 #:G1501 #:G1500 #:G1499 #:G1498 |pv$|)
    (RETURN
      (PROGN
        (LETT DV$1 (|devaluate| |#1|) |Equation|)
        (LETT |dv$| (LIST '|Equation| DV$1) |Equation|)
        (LETT $ (GETREFV 98) |Equation|)
        (QSETREFV $ 0 |dv$|)
        (QSETREFV $ 3
          (LETT |pv$|
            (|buildPredVector| 0 0
              (LIST (|HasCategory| |#1| '|(Field|)')
                (|HasCategory| |#1| '|(SetCategory|)')
                (|HasCategory| |#1| '|(Ring|)')
                (|HasCategory| |#1|
                  '|(PartialDifferentialRing| (|Symbol|))')
                (OR (|HasCategory| |#1|
                  '|(PartialDifferentialRing|
                    (|Symbol|))')
                  (|HasCategory| |#1| '|(Ring|)'))
                (|HasCategory| |#1| '|(Group|)')
                (|HasCategory| |#1|
                  (LIST '|InnerEvalable| '|(Symbol|)
                    (|devaluate| |#1|))')
                (AND (|HasCategory| |#1|
                  (LIST '|Evalable|
                    (|devaluate| |#1|))')
                  (|HasCategory| |#1| '|(SetCategory|)'))
                (|HasCategory| |#1| '|(IntegralDomain|)')
                (|HasCategory| |#1| '|(ExpressionSpace|)')
                (OR (|HasCategory| |#1| '|(Field|)')
                  (|HasCategory| |#1| '|(Group|)'))
                (OR (|HasCategory| |#1| '|(Group|)')
                  (|HasCategory| |#1| '|(Ring|)'))
                (LETT #:G1502

```



```

(|HasCategory| |#1|
  '(|CommutativeRing|))
|Equation|)
(OR #:G1502 (|HasCategory| |#1| '(|Field|))
  (|HasCategory| |#1| '(|Ring|)))
(OR #:G1502
  (|HasCategory| |#1| '(|Field|)))
(LETT #:G1501
  (|HasCategory| |#1| '(|Monoid|))
  |Equation|)
(OR (|HasCategory| |#1| '(|Group|))
  #:G1501)
(LETT #:G1500
  (|HasCategory| |#1| '(|SemiGroup|))
  |Equation|)
(OR (|HasCategory| |#1| '(|Group|)) #:G1501
  #:G1500)
(LETT #:G1499
  (|HasCategory| |#1|
    '(|AbelianGroup|))
  |Equation|)
(OR (|HasCategory| |#1|
  '(|PartialDifferentialRing|
    (|Symbol|)))
  #:G1499 #:G1502
  (|HasCategory| |#1| '(|Field|))
  (|HasCategory| |#1| '(|Ring|)))
(OR #:G1499 #:G1501)
(LETT #:G1498
  (|HasCategory| |#1|
    '(|AbelianSemiGroup|))
  |Equation|)
(OR (|HasCategory| |#1|
  '(|PartialDifferentialRing|
    (|Symbol|)))
  #:G1499 #:G1498 #:G1502
  (|HasCategory| |#1| '(|Field|))
  (|HasCategory| |#1| '(|Ring|)))
(OR (|HasCategory| |#1|
  '(|PartialDifferentialRing|
    (|Symbol|)))
  #:G1499 #:G1498 #:G1502
  (|HasCategory| |#1| '(|Field|))
  (|HasCategory| |#1| '(|Group|)) #:G1501
  (|HasCategory| |#1| '(|Ring|)) #:G1500
  (|HasCategory| |#1| '(|SetCategory|))))
|Equation|))
(|haddProp| |$ConstructorCache| '|Equation| (LIST DV$1)
  (CONS 1 $))
(|stuffDomainSlots| $)

```

```

(QSETREFV $ 6 |#1|)
(QSETREFV $ 7 (|Record| (|:| |lhs| |#1|) (|:| |rhs| |#1|)))
(COND
  ((|testBitVector| |pv$| 9)
    (QSETREFV $ 19
      (CONS (|dispatchFunction| |EQ;factorAndSplit;$L;1|) $))))
(COND
  ((|testBitVector| |pv$| 7)
    (PROGN
      (QSETREFV $ 27
        (CONS (|dispatchFunction| |EQ;eval;$SS$;8|) $))
      (QSETREFV $ 31
        (CONS (|dispatchFunction| |EQ;eval;$LL$;9|) $))))))
(COND
  ((|HasCategory| |#1| (LIST ' |Evalable| (|devaluate| |#1|)))
    (PROGN
      (QSETREFV $ 34
        (CONS (|dispatchFunction| |EQ;eval;3$;10|) $))
      (QSETREFV $ 37
        (CONS (|dispatchFunction| |EQ;eval;$L$;11|) $))))))
(COND
  ((|testBitVector| |pv$| 2)
    (PROGN
      (QSETREFV $ 40
        (CONS (|dispatchFunction| |EQ;=;2$B;12|) $))
      (QSETREFV $ 44
        (CONS (|dispatchFunction| |EQ;coerce;$0f;13|) $))
      (QSETREFV $ 45
        (CONS (|dispatchFunction| |EQ;coerce;$B;14|) $))))))
(COND
  ((|testBitVector| |pv$| 23)
    (PROGN
      (QSETREFV $ 47 (CONS (|dispatchFunction| |EQ;+;3$;15|) $))
      (QSETREFV $ 48
        (CONS (|dispatchFunction| |EQ;+;S2$;16|) $))
      (QSETREFV $ 49
        (CONS (|dispatchFunction| |EQ;+;$S$;17|) $))))))
(COND
  ((|testBitVector| |pv$| 20)
    (PROGN
      (QSETREFV $ 51 (CONS (|dispatchFunction| |EQ;-;2$;18|) $))
      (QSETREFV $ 53
        (CONS (|dispatchFunction| |EQ;-;S2$;19|) $))
      (QSETREFV $ 54
        (CONS (|dispatchFunction| |EQ;-;$S$;20|) $))
      (QSETREFV $ 57
        (CONS (|dispatchFunction| |EQ;leftZero;2$;21|) $))
      (QSETREFV $ 8
        (CONS (|dispatchFunction| |EQ;rightZero;2$;22|) $))
      (QSETREFV $ 55

```

```

(CONS IDENTITY
  (FUNCALL (|dispatchFunction| |EQ;Zero;$;23|) $)))
(QSETREFV $ 52 (CONS (|dispatchFunction| |EQ;-;3$;24|) $))))))
(COND
  ((|testBitVector| |pv$| 18)
    (PROGN
      (QSETREFV $ 59 (CONS (|dispatchFunction| |EQ;*;3$;25|) $))
      (QSETREFV $ 60
        (CONS (|dispatchFunction| |EQ;*;S2$;26|) $))
      (QSETREFV $ 60
        (CONS (|dispatchFunction| |EQ;*;S2$;27|) $))
      (QSETREFV $ 61
        (CONS (|dispatchFunction| |EQ;*;S$;28|) $))))))
(COND
  ((|testBitVector| |pv$| 16)
    (PROGN
      (QSETREFV $ 63
        (CONS IDENTITY
          (FUNCALL (|dispatchFunction| |EQ;One;$;29|) $)))
      (QSETREFV $ 66
        (CONS (|dispatchFunction| |EQ;recip;$U;30|) $))
      (QSETREFV $ 67
        (CONS (|dispatchFunction| |EQ;leftOne;$U;31|) $))
      (QSETREFV $ 68
        (CONS (|dispatchFunction| |EQ;rightOne;$U;32|) $))))))
(COND
  ((|testBitVector| |pv$| 6)
    (PROGN
      (QSETREFV $ 70
        (CONS (|dispatchFunction| |EQ;inv;2$;33|) $))
      (QSETREFV $ 67
        (CONS (|dispatchFunction| |EQ;leftOne;$U;34|) $))
      (QSETREFV $ 68
        (CONS (|dispatchFunction| |EQ;rightOne;$U;35|) $))))))
(COND
  ((|testBitVector| |pv$| 3)
    (PROGN
      (QSETREFV $ 73
        (CONS (|dispatchFunction| |EQ;characteristic;Nni;36|)
          $))
      (QSETREFV $ 76
        (CONS (|dispatchFunction| |EQ;*;I2$;37|) $))))))
(COND
  ((|testBitVector| |pv$| 9)
    (QSETREFV $ 19
      (CONS (|dispatchFunction| |EQ;factorAndSplit;$L;38|) $))))
(COND
  ((|testBitVector| |pv$| 4)
    (QSETREFV $ 85
      (CONS (|dispatchFunction| |EQ;differentiate;S$;39|) $))))

```

```

(COND
  ((|testBitVector| |pv$| 1)
   (PROGN
    (QSETREFV $ 88
     (CONS (|dispatchFunction| |EQ;dimension;Cn;40|) $))
    (QSETREFV $ 90 (CONS (|dispatchFunction| |EQ;/;3$;41|) $))
    (QSETREFV $ 70
     (CONS (|dispatchFunction| |EQ;inv;2$;42|) $))))))
(COND
  ((|testBitVector| |pv$| 10)
   (QSETREFV $ 93
    (CONS (|dispatchFunction| |EQ;subst;3$;43|) $))))
$))))

(MAKEPROP '|Equation| '|infovec|
  (LIST '#(NIL NIL NIL NIL NIL NIL (|local| |#1|) '|Rep|
    (0 . |rightZero|) |EQ;lhs;$S;4| (|Factored| $)
    (5 . |factor|)
    (|Record| (|:| |factor| 6) (|:| |exponent| 74))
    (|List| 12) (|Factored| 6) (10 . |factors|) (15 . |Zero|)
    |EQ;equation;2S$;3| (|List| $) (19 . |factorAndSplit|)
    |EQ;=;2S$;2| |EQ;rhs;$S;5| |EQ;swap;2$;6| (|Mapping| 6 6)
    |EQ;map;M2$;7| (|Symbol|) (24 . |eval|) (31 . |eval|)
    (|List| 25) (|List| 6) (38 . |eval|) (45 . |eval|)
    (|Equation| 6) (52 . |eval|) (58 . |eval|) (|List| 32)
    (64 . |eval|) (70 . |eval|) (|Boolean|) (76 . =) (82 . =)
    (|OutputForm|) (88 . |coerce|) (93 . =) (99 . |coerce|)
    (104 . |coerce|) (109 . +) (115 . +) (121 . +) (127 . +)
    (133 . -) (138 . -) (143 . -) (149 . -) (155 . -)
    (161 . |Zero|) (165 . -) (171 . |leftZero|) (176 . *)
    (182 . *) (188 . *) (194 . *) (200 . |One|) (204 . |One|)
    (|Union| $ '"failed") (208 . |recip|) (213 . |recip|)
    (218 . |leftOne|) (223 . |rightOne|) (228 . |inv|)
    (233 . |inv|) (|NonNegativeInteger|)
    (238 . |characteristic|) (242 . |characteristic|)
    (|Integer|) (246 . |coerce|) (251 . *) (|Factored| 78)
    (|Polynomial| 74)
    (|MultivariateFactorize| 25 (|IndexedExponents| 25) 74 78)
    (257 . |factor|)
    (|Record| (|:| |factor| 78) (|:| |exponent| 74))
    (|List| 81) (262 . |factors|) (267 . |differentiate|)
    (273 . |differentiate|) (|CardinalNumber|)
    (279 . |coerce|) (284 . |dimension|) (288 . /) (294 . /)
    (|Equation| $) (300 . |subst|) (306 . |subst|)
    (|PositiveInteger|) (|List| 71) (|SingleInteger|)
    (|String|))
  '#(~= 312 |zero?| 318 |swap| 323 |subtractIfCan| 328 |subst|
    334 |sample| 340 |rightZero| 344 |rightOne| 349 |rhs| 354
    |recip| 359 |one?| 364 |map| 369 |lhs| 375 |leftZero| 380
    |leftOne| 385 |latex| 390 |inv| 395 |hash| 400

```

```

|factorAndSplit| 405 |eval| 410 |equation| 436 |dimension|
442 |differentiate| 446 |conjugate| 472 |commutator| 478
|coerce| 484 |characteristic| 499 ^ 503 |Zero| 521 |One|
525 D 529 = 555 / 567 - 579 + 602 ** 620 * 638)
'((|unitsKnown| . 12) (|rightUnitary| . 3)
(|leftUnitary| . 3))
(CONS (|makeByteWordVec2| 25
      '(1 15 4 14 5 14 3 5 3 21 21 6 21 17 24 19 25 0 2
        25 2 7))
(CONS '#(|VectorSpace&| |Module&|
|PartialDifferentialRing&| NIL |Ring&| NIL NIL
NIL NIL |AbelianGroup&| NIL |Group&|
|AbelianMonoid&| |Monoid&| |AbelianSemiGroup&|
|SemiGroup&| |SetCategory&| NIL NIL
|BasicType&| NIL |InnerEvalable&|)
(CONS '#(|VectorSpace| 6) (|Module| 6)
(|PartialDifferentialRing| 25)
(|BiModule| 6 6) (|Ring|)
(|LeftModule| 6) (|RightModule| 6)
(|Rng|) (|LeftModule| $$)
(|AbelianGroup|)
(|CancellationAbelianMonoid|) (|Group|)
(|AbelianMonoid|) (|Monoid|)
(|AbelianSemiGroup|) (|SemiGroup|)
(|SetCategory|) (|Type|)
(|CoercibleTo| 41) (|BasicType|)
(|CoercibleTo| 38)
(|InnerEvalable| 25 6))
(|makeByteWordVec2| 97
  '(1 0 0 0 8 1 6 10 0 11 1 14 13 0 15 0
    6 0 16 1 0 18 0 19 3 6 0 0 25 6 26 3
    0 0 0 25 6 27 3 6 0 0 28 29 30 3 0 0
    0 28 29 31 2 6 0 0 32 33 2 0 0 0 0 34
    2 6 0 0 35 36 2 0 0 0 18 37 2 6 38 0
    0 39 2 0 38 0 0 40 1 6 41 0 42 2 41 0
    0 0 43 1 0 41 0 44 1 0 38 0 45 2 6 0
    0 0 46 2 0 0 0 0 47 2 0 0 6 0 48 2 0
    0 0 6 49 1 6 0 0 50 1 0 0 0 51 2 0 0
    0 0 52 2 0 0 6 0 53 2 0 0 0 6 54 0 0
    0 55 2 6 0 0 0 56 1 0 0 0 57 2 6 0 0
    0 58 2 0 0 0 0 59 2 0 0 6 0 60 2 0 0
    0 6 61 0 6 0 62 0 0 0 63 1 6 64 0 65
    1 0 64 0 66 1 0 64 0 67 1 0 64 0 68 1
    6 0 0 69 1 0 0 0 70 0 6 71 72 0 0 71
    73 1 6 0 74 75 2 0 0 74 0 76 1 79 77
    78 80 1 77 82 0 83 2 6 0 0 25 84 2 0
    0 0 25 85 1 86 0 71 87 0 0 86 88 2 6
    0 0 0 89 2 0 0 0 0 90 2 6 0 0 91 92 2
    0 0 0 0 93 2 2 38 0 0 1 1 20 38 0 1 1
    0 0 0 22 2 20 64 0 0 1 2 10 0 0 0 93

```

```

0 22 0 1 1 20 0 0 8 1 16 64 0 68 1 0
6 0 21 1 16 64 0 66 1 16 38 0 1 2 0 0
23 0 24 1 0 6 0 9 1 20 0 0 57 1 16 64
0 67 1 2 97 0 1 1 11 0 0 70 1 2 96 0
1 1 9 18 0 19 2 8 0 0 0 34 2 8 0 0 18
37 3 7 0 0 25 6 27 3 7 0 0 28 29 31 2
0 0 6 6 17 0 1 86 88 2 4 0 0 28 1 2 4
0 0 25 85 3 4 0 0 28 95 1 3 4 0 0 25
71 1 2 6 0 0 0 1 2 6 0 0 0 1 1 3 0 74
1 1 2 41 0 44 1 2 38 0 45 0 3 71 73 2
6 0 0 74 1 2 16 0 0 71 1 2 18 0 0 94
1 0 20 0 55 0 16 0 63 2 4 0 0 28 1 2
4 0 0 25 1 3 4 0 0 28 95 1 3 4 0 0 25
71 1 2 2 38 0 0 40 2 0 0 6 6 20 2 11
0 0 0 90 2 1 0 0 6 1 1 20 0 0 51 2 20
0 0 0 52 2 20 0 6 0 53 2 20 0 0 6 54
2 23 0 0 0 47 2 23 0 6 0 48 2 23 0 0
6 49 2 6 0 0 74 1 2 16 0 0 71 1 2 18
0 0 94 1 2 20 0 71 0 1 2 20 0 74 0 76
2 23 0 94 0 1 2 18 0 0 0 59 2 18 0 0
6 61 2 18 0 6 0 60))))))
'|lookupComplete|))

```

1.4 The code.o file

The Spad compiler translates the Spad language into Common Lisp. It eventually invokes the Common Lisp “compile-file” command to output files in binary. Depending on the lisp system this filename can vary (e.g “code.fasl”). The details of how these are used depends on the Common Lisp in use.

By default, Axiom uses Gnu Common Lisp (GCL), which generates “.o” files.

1.5 The info file

```

(((* (($ $ $) (|arguments| (|eq2| . $) (|eq1| . $)) (S (* S S S))
($ (= $ S S)))
(($ $ $ S) (|arguments| (|l| . S) (|eqn| . $)) (S (* S S S))
($ (= $ S S)))
(($ #0=(|Integer|) $) (|arguments| (|i| . #0#) (|eq| . $))
(S (|coerce| S (|Integer|))) ($ (* $ $ S $)))
(($ S $) (|arguments| (|l| . S) (|eqn| . $)) (S (* S S S))
($ (= $ S S)))
(+ (($ $ $ $) (|arguments| (|eq2| . $) (|eq1| . $)) (S (+ S S S))
($ (= $ S S)))
(($ $ $ S) (|arguments| (|s| . S) (|eq1| . $)) ($ (+ $ $ $ $)))
(($ S $) (|arguments| (|s| . S) (|eq2| . $)) ($ (+ $ $ $ $))))

```

```

(- (($ $ $) (|arguments| (|eq2| . $) (|eq1| . $)) (S (- S S S))
  ($ (= $ S S)))
  (($ $ S) (|arguments| (|s| . S) (|eq1| . $)) ($ (- $ $ $)))
  (($ $) (|arguments| (|eq| . $)) (S (- S S))
    ($ (|rhs| S $) (|lhs| S $) (= $ S S)))
  (($ S $) (|arguments| (|s| . S) (|eq2| . $)) ($ (- $ $ $)))
(/ (($ $ $) (|arguments| (|eq2| . $) (|eq1| . $)) (S (/ S S S))
  ($ (= $ S S)))
(= (($ S S) (|arguments| (|r| . S) (|l| . S)))
  (((|Boolean|) $) ((|Boolean|) (|false| (|Boolean|))))
  (|locals| (#:G1393 |Boolean|))
  (|arguments| (|eq2| . $) (|eq1| . $)) (S (= (|Boolean|) S S)))
(|One| (($) (S (|One| S)) ($ (|equation| $ S S))))
(|Zero| (($) (S (|Zero| S)) ($ (|equation| $ S S))))
(|characteristic|
  (((|NonNegativeInteger|))
  (S (|characteristic| (|NonNegativeInteger|)))))
(|coerce|
  (((|Boolean|) $) (|arguments| (|eqn| . $))
  (S (= (|Boolean|) S S)))
  (((|OutputForm|) $)
  ((|OutputForm|) (= (|OutputForm|) (|OutputForm|) (|OutputForm|)))
  (|arguments| (|eqn| . $)) (S (|coerce| (|OutputForm|) S))))
(|constructor|
  (NIL (|locals|
    (|Rep| |Join| (|SetCategory|)
      (CATEGORY |domain|
        (SIGNATURE |construct|
          ((|Record| (|:| |lhs| S) (|:| |rhs| S)) S)
          S))
        (SIGNATURE |coerce|
          ((|OutputForm|)
            (|Record| (|:| |lhs| S) (|:| |rhs| S))))
        (SIGNATURE |elt|
          (S (|Record| (|:| |lhs| S) (|:| |rhs| S))
            "lhs"))
        (SIGNATURE |elt|
          (S (|Record| (|:| |lhs| S) (|:| |rhs| S))
            "rhs"))
        (SIGNATURE |setelt|
          (S (|Record| (|:| |lhs| S) (|:| |rhs| S))
            "lhs" S))
        (SIGNATURE |setelt|
          (S (|Record| (|:| |lhs| S) (|:| |rhs| S))
            "rhs" S))
        (SIGNATURE |copy|
          ((|Record| (|:| |lhs| S) (|:| |rhs| S))
            (|Record| (|:| |lhs| S) (|:| |rhs| S)))))))))
(|differentiate|
  (($ $ #1=(|Symbol|)) (|arguments| (|sym| . #1#) (|eq| . $))

```

```

(S (|differentiate| S S (|Symbol|))) ($ (|rhs| S $) (|lhs| S $)))
(|dimension|
  ((#2=(|CardinalNumber|))
    (#2# (|coerce| (|CardinalNumber|) (|NonNegativeInteger|))))
(|equation| (($ S S) (|arguments| (|r| . S) (|l| . S))))
(|eval| (($ $ $) (|arguments| (|eqn2| . $) (|eqn1| . $))
  (S (|eval| S S (|Equation| S))) ($ (= $ S S)))
  (($ $ #3=(|List| $))
    (|arguments| (|eqn2| . #3#) (|eqn1| . $))
    (S (|eval| S S (|List| (|Equation| S)))) ($ (= $ S S)))
  (($ $ #4=(|List| #5=(|Symbol|)) #6=(|List| S))
    (|arguments| (|lx| . #6#) (|ls| . #4#) (|eqn| . $))
    (S (|eval| S S (|List| (|Symbol|)) (|List| S)))
    ($ (= $ S S)))
  (($ $ #5# S) (|arguments| (|lx| . S) (|ls| . #5#) (|eqn| . $))
    (S (|eval| S S (|Symbol| S)) ($ (= $ S S))))
(|factorAndSplit|
  ((|List| $) $)
  ((|MultivariateFactorize| (|Symbol|)
    (|IndexedExponents| (|Symbol|)) (|Integer|)
    (|Polynomial| (|Integer|)))
    (|factor| (|Factored| (|Polynomial| (|Integer|)))
      (|Polynomial| (|Integer|)))
    (|Factored| S)
    (|factors|
      (|List| (|Record| (|:| |factor| S)
        (|:| |exponent| (|Integer|))))
      (|Factored| S)))
  ((|Factored| (|Polynomial| (|Integer|)))
    (|factors|
      (|List| (|Record| (|:| |factor| (|Polynomial| (|Integer|)))
        (|:| |exponent| (|Integer|))))
      (|Factored| (|Polynomial| (|Integer|))))
    (|locals| (|p| |Polynomial| (|Integer|)) (|eq0| . $))
    (|arguments| (|eq| . $))
    (S (|factor| (|Factored| S) S) (|Zero| S))
    ($ (|rightZero| $ $) (|lhs| S $) (|equation| $ S S)))
(|inv| (($ $) (|arguments| (|eq| . $)) (S (|inv| S S))
  ($ (|rhs| S $) (|lhs| S $))))
(|leftOne|
  ((|Union| $ "failed") $) (|locals| (|re| |Union| S "failed"))
  (|arguments| (|eq| . $))
  (S (|recip| (|Union| S "failed") S) (|inv| S S) (|One| S)
    (* S S S))
  ($ (|rhs| S $) (|lhs| S $) (|One| $) (= $ S S)))
(|leftZero|
  (($ $) (|arguments| (|eq| . $)) (S (|Zero| S) (- S S S))
  ($ (|rhs| S $) (|lhs| S $) (|Zero| $) (= $ S S)))
(|lhs| (($ $) (|arguments| (|eqn| . $))))
(|map| (($ #7=(|Mapping| S S) $)

```



```

      (|arguments| (|fn| . #7#) (|eqn| . $)) ($ (|equation| $ S S)))
(|recip| (((|Union| $ "failed") $)
  (|locals| (|rh| |Union| S "failed")
    (|lh| |Union| S "failed")))
  (|arguments| (|eq| . $))
  (S (|recip| (|Union| S "failed") S))
  ($ (|rhs| S $) (|lhs| S $))))
(|rhs| ((S $) (|arguments| (|eqn| . $))))
(|rightOne|
  (((|Union| $ "failed") $) (|locals| (|re| |Union| S "failed")
  (|arguments| (|eq| . $))
  (S (|recip| (|Union| S "failed") S) (|inv| S S) (|One| S)
    (* S S S))
  ($ (|rhs| S $) (|lhs| S $) (= $ S S))))
(|rightZero|
  (($ $) (|arguments| (|eq| . $)) (S (|Zero| S) (- S S S))
  ($ (|rhs| S $) (|lhs| S $) (= $ S S))))
(|subst| (($ $ $) (|locals| (|eq3| |Equation| S))
  (|arguments| (|eq2| . $) (|eq1| . $))
  (S (|subst| S S (|Equation| S)))
  ($ (|rhs| S $) (|lhs| S $))))
(|swap| (($ $) (|arguments| (|eqn| . $)) ($ (|rhs| S $) (|lhs| S $))))

```

1.6 The EQ.fn file

```

(in-package 'compiler)(init-fn)
(ADD-FN-DATA '(
#S(FN NAME BOOT::|EQ;*;S2$;26| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;rightOne;$U;32| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (BOOT::|spadConstant| VMLISP:QCDR CONS VMLISP:QCAR EQL
    BOOT::|EQQCAR COND VMLISP:EXIT CDR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL BOOT::|LETT VMLISP:SEQ RETURN)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QCDR VMLISP:QCAR BOOT::|EQQCAR COND
    VMLISP:EXIT VMLISP:QREFELT BOOT:SPADCALL BOOT::|LETT
    VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;lhs;$S;4| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CAR VMLISP:QCAR) RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT
  NIL MACROS (VMLISP:QCAR))
#S(FN NAME BOOT::|EQ;+;3$;15| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT

```

```

        BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;dimension;Cn;40| DEF DEFUN VALUE-TYPE T FUN-VALUES
    NIL CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
    (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;rightZero;2$;22| DEF DEFUN VALUE-TYPE T FUN-VALUES
    NIL CALLEES
    (BOOT::|spadConstant| CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
    (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;coerce;$0f;13| DEF DEFUN VALUE-TYPE T FUN-VALUES
    NIL CALLEES
    (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
     BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;One;$;29| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES
    (CDR BOOT::|spadConstant| CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
    (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;inv;2$;42| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS)
    RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
    (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;-;$S$;20| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES (CDR CONS CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
    (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;=;2$B;12| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES
    (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
     BOOT:SPADCALL COND)
    RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL COND))
#S(FN NAME BOOT::|EQ;/;3$;41| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES
    (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
     BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;recip;$U;30| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES
    (VMLISP:QCDR LIST* CONS VMLISP:QCAR EQL BOOT::QEQCAR COND
     VMLISP:EXIT CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL
     BOOT::LETT VMLISP:SEQ RETURN)
    RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR BOOT::QEQCAR COND VMLISP:EXIT

```

```

        VMLISP:QREFELT BOOT:SPADCALL BOOT::LETT VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;-;3$;24| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;eval;$L$;11| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;leftZero;2$;21| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (CDR BOOT::|spadConstant| CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;*;S2$;27| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;*;I2$;37| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL) RETURN-TYPE
  NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;eval;3$;10| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;eval;$SS$;8| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;factorAndSplit;$L;38| DEF DEFUN VALUE-TYPE T
  FUN-VALUES NIL CALLEES
  (BOOT:|Integer| BOOT:|Polynomial| EQUAL BOOT:NREVERSEO
    BOOT::|spadConstant| VMLISP:QCAR CONS ATOM VMLISP:EXIT CDR
    CAR BOOT:SPADCALL BOOT::LETT BOOT::|devaluate| LIST SVREF
    VMLISP:QREFELT BOOT::|HasSignature| COND VMLISP:SEQ RETURN)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QCAR VMLISP:EXIT BOOT:SPADCALL
    BOOT::LETT VMLISP:QREFELT COND VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;differentiate;$S$;39| DEF DEFUN VALUE-TYPE T

```

```

FUN-VALUES NIL CALLEES
(CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS) RETURN-TYPE NIL
ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;eval;$LL$;9| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
   BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;leftOne;$U;34| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (CDR BOOT::|spadConstant| CAR SVREF VMLISP:QREFELT BOOT:SPADCALL
   CONS)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;map;M2$;7| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
   BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;-;S2$;19| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CDR CONS CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;equation;2S$;3| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES (CONS) RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL
  MACROS NIL)
#S(FN NAME BOOT::|EQ;+;$S$;17| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CDR CONS CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;factorAndSplit;$L;1| DEF DEFUN VALUE-TYPE T
  FUN-VALUES NIL CALLEES
  (BOOT:NREVERSEO BOOT::|spadConstant| VMLISP:QCAR CONS ATOM
   VMLISP:EXIT CDR CAR BOOT:SPADCALL BOOT::LETT
   BOOT::|devaluate| LIST SVREF VMLISP:QREFELT
   BOOT::|HasSignature| COND VMLISP:SEQ RETURN)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QCAR VMLISP:EXIT BOOT:SPADCALL
   BOOT::LETT VMLISP:QREFELT COND VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;*;3$;25| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
   BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;Zero;$;23| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES

```

```

(CDR BOOT::|spadConstant| CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
(BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;characteristic;Nni;36| DEF DEFUN VALUE-TYPE T
FUN-VALUES NIL CALLEES
(CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL) RETURN-TYPE NIL
ARG-TYPES (T) NO-EMIT NIL MACROS (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;leftOne;$U;31| DEF DEFUN VALUE-TYPE T FUN-VALUES
NIL CALLEES
(VMLISP:QCDR BOOT::|spadConstant| CONS VMLISP:QCAR EQL
BOOT::|QEQCAR COND VMLISP:EXIT CDR CAR SVREF VMLISP:QREFELT
BOOT:SPADCALL BOOT::LETT VMLISP:SEQ RETURN)
RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
(VMLISP:QCDR BOOT::|spadConstant| VMLISP:QCAR BOOT::|QEQCAR COND
VMLISP:EXIT VMLISP:QREFELT BOOT:SPADCALL BOOT::LETT
VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;swap;2$;6| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS)
RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;-;2$;18| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL) RETURN-TYPE
NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;subst;3$;43| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS VMLISP:EXIT
BOOT::LETT VMLISP:SEQ RETURN)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL VMLISP:EXIT BOOT::LETT VMLISP:SEQ
RETURN))
#S(FN NAME BOOT::|EQ;=;2S$;2| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CONS) RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL
MACROS NIL)
#S(FN NAME BOOT::|EQ;*;$S$;28| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;+;S2$;16| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CDR CONS CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|Equation;| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(BOOT::|EQ;One;$;29| BOOT::|EQ;Zero;$;23|
BOOT::|dispatchFunction| BOOT::|testBitVector| COND
BOOT::|Record0| BOOT::|Record| BOOT::|stuffDomainSlots| CONS
BOOT::|haddProp| BOOT::|HasCategory| BOOT::|buildPredVector|

```

```

SYSTEM:SVSET SETF VMLISP:QSETREFV VMLISP:GETREFV LIST
BOOT::|devaluate| BOOT::LETT RETURN)
RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
(BOOT::|dispatchFunction| COND BOOT::|Record| SETF
  VMLISP:QSETREFV BOOT::LETT RETURN))
#S(FN NAME BOOT::|EQ;coerce;$B;14| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (CDR VMLISP:QCDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;rhs;$S;5| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CDR VMLISP:QCDR) RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT
  NIL MACROS (VMLISP:QCDR))
#S(FN NAME OTHER-FORM DEF NIL VALUE-TYPE NIL FUN-VALUES NIL CALLEES NIL
  RETURN-TYPE NIL ARG-TYPES NIL NO-EMIT NIL MACROS NIL)
#S(FN NAME BOOT::|EQ;inv;2$;33| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;rightOne;$U;35| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (BOOT::|spadConstant| CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL
    CONS)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|Equation| DEF DEFUN VALUE-TYPE T FUN-VALUES
  (SINGLE-VALUE) CALLEES
  (REMHASH VMLISP:HREM BOOT::|Equation;| PROG1
    BOOT::|CDRwithIncrement| GETHASH VMLISP:HGET
    BOOT::|devaluate| LIST BOOT::|lassocShiftWithFunction|
    BOOT::LETT COND RETURN)
  RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
  (VMLISP:HREM PROG1 VMLISP:HGET BOOT::LETT COND RETURN)) )

```

1.7 The index.kaf file

Each constructor (e.g. EQ) had one library directory (e.g. EQ.nrlib). This directory contained a random access file called the index.kaf file. These files contain runtime information such as the operationAlist and the ConstructorModemap. At system build time we merge all of these .nrlib/index.kaf files into one database, INTERP.daase. Requests to get information from this database are cached so that multiple references do not cause additional disk i/o.

Before getting into the contents, we need to understand the format of an index.kaf file. The kaf file is a random access file, originally used as a database. In the current system we make a pass to combine these files at build time to construct the various daase files.

This is just a file of lisp objects, one after another, in (read) format.

A kaf file starts with an integer, in this case, 35695. This integer gives the byte offset to the index. Due to the way the file is constructed, the index is at the end of the file. To read a kaf file, first read the integer, then seek to that location in the file, and do a (read). This will return the index, in this case:

```
((("slot1Info" 0 32444)
  ("documentation" 0 29640)
  ("ancestors" 0 28691)
  ("parents" 0 28077)
  ("abbreviation" 0 28074)
  ("predicates" 0 25442)
  ("attributes" 0 25304)
  ("signaturesAndLocals" 0 23933)
  ("superDomain" 0 NIL)
  ("operationAlist" 0 20053)
  ("modemaps" 0 17216)
  ("sourceFile" 0 17179)
  ("constructorCategory" 0 15220)
  ("constructorModemap" 0 13215)
  ("constructorKind" 0 13206)
  ("constructorForm" 0 13191)
  ("compilerInfo" 0 4433)
  ("loadTimeStuff" 0 20))
```

This is a list of triples. The first item in each triple is a string that is used as a lookup key (e.g. “operationAlist”). The second element is no longer used. The third element is the byte offset from the beginning of the file.

So to read the “operationAlist” from this file you would:

1. open the index.kaf file
2. (read) the integer
3. (seek) to the integer offset from the beginning of the file
4. (read) the index of triples
5. find the keyword (e.g. “operationAlist”) triple
6. select the third element, an integer
7. (seek) to the integer offset from the beginning of the file
8. (read) the “operationAlist”

Note that the information below has been reformatted to fit this document. In order to save space the index.kaf file is does not use prettyprint since it is normally only read by machine.

1.7.1 The index offset byte

35695

1.7.2 The “loadTimeStuff”

```
(MAKEPROP '|Equation| '|infovec|
  (LIST '#(NIL NIL NIL NIL NIL NIL (|local| |#1|) '|Rep|
    (0 . |rightZero|) |EQ;lhs;$S;4| (|Factored| $)
    (5 . |factor|)
    (|Record| (|:| |factor| 6) (|:| |exponent| 74))
    (|List| 12) (|Factored| 6) (10 . |factors|) (15 . |Zero|)
    |EQ;equation;2S$;3| (|List| $) (19 . |factorAndSplit|)
    |EQ;=;2S$;2| |EQ;rhs;$S;5| |EQ;swap;2$;6| (|Mapping| 6 6)
    |EQ;map;M2$;7| (|Symbol|) (24 . |eval|) (31 . |eval|)
    (|List| 25) (|List| 6) (38 . |eval|) (45 . |eval|)
    (|Equation| 6) (52 . |eval|) (58 . |eval|) (|List| 32)
    (64 . |eval|) (70 . |eval|) (|Boolean|) (76 . =) (82 . =)
    (|OutputForm|) (88 . |coerce|) (93 . =) (99 . |coerce|)
    (104 . |coerce|) (109 . +) (115 . +) (121 . +) (127 . +)
    (133 . -) (138 . -) (143 . -) (149 . -) (155 . -)
    (161 . |Zero|) (165 . -) (171 . |leftZero|) (176 . *)
    (182 . *) (188 . *) (194 . *) (200 . |One|) (204 . |One|)
    (|Union| $ '"failed") (208 . |recip|) (213 . |recip|)
    (218 . |leftOne|) (223 . |rightOne|) (228 . |inv|)
    (233 . |inv|) (|NonNegativeInteger|)
    (238 . |characteristic|) (242 . |characteristic|)
    (|Integer|) (246 . |coerce|) (251 . *) (|Factored| 78)
    (|Polynomial| 74)
    (|MultivariateFactorize| 25 (|IndexedExponents| 25) 74 78)
    (257 . |factor|)
    (|Record| (|:| |factor| 78) (|:| |exponent| 74))
    (|List| 81) (262 . |factors|) (267 . |differentiate|)
    (273 . |differentiate|) (|CardinalNumber|)
    (279 . |coerce|) (284 . |dimension|) (288 . /) (294 . /)
    (|Equation| $) (300 . |subst|) (306 . |subst|)
    (|PositiveInteger|) (|List| 71) (|SingleInteger|)
    (|String|))
  '#(~= 312 |zero?| 318 |swap| 323 |subtractIfCan| 328 |subst|
    334 |sample| 340 |rightZero| 344 |rightOne| 349 |rhs| 354
    |recip| 359 |one?| 364 |map| 369 |lhs| 375 |leftZero| 380
    |leftOne| 385 |latex| 390 |inv| 395 |hash| 400
    |factorAndSplit| 405 |eval| 410 |equation| 436 |dimension|
    442 |differentiate| 446 |conjugate| 472 |commutator| 478
    |coerce| 484 |characteristic| 499 ^ 503 |Zero| 521 |One|
    525 D 529 = 555 / 567 - 579 + 602 ** 620 * 638)
  '((|unitsKnown| . 12) (|rightUnitary| . 3)
    (|leftUnitary| . 3))
(CONS (|makeByteWordVec2| 25
```



```

'(1 15 4 14 5 14 3 5 3 21 21 6 21 17 24 19 25 0 2
  25 2 7))
(CONS '#(|VectorSpace&| |Module&|
         |PartialDifferentialRing&| NIL |Ring&| NIL NIL
         NIL NIL |AbelianGroup&| NIL |Group&|
         |AbelianMonoid&| |Monoid&| |AbelianSemiGroup&|
         |SemiGroup&| |SetCategory&| NIL NIL
         |BasicType&| NIL |InnerEvalable&|)
(CONS '#(|VectorSpace| 6) (|Module| 6)
        (|PartialDifferentialRing| 25)
        (|BiModule| 6 6) (|Ring|)
        (|LeftModule| 6) (|RightModule| 6)
        (|Rng|) (|LeftModule| $$)
        (|AbelianGroup|)
        (|CancellationAbelianMonoid|) (|Group|)
        (|AbelianMonoid|) (|Monoid|)
        (|AbelianSemiGroup|) (|SemiGroup|)
        (|SetCategory|) (|Type|)
        (|CoercibleTo| 41) (|BasicType|)
        (|CoercibleTo| 38)
        (|InnerEvalable| 25 6))
(|makeByteWordVec2| 97
  '(1 0 0 0 8 1 6 10 0 11 1 14 13 0 15 0
    6 0 16 1 0 18 0 19 3 6 0 0 25 6 26 3
    0 0 0 25 6 27 3 6 0 0 28 29 30 3 0 0
    0 28 29 31 2 6 0 0 32 33 2 0 0 0 0 34
    2 6 0 0 35 36 2 0 0 0 18 37 2 6 38 0
    0 39 2 0 38 0 0 40 1 6 41 0 42 2 41 0
    0 0 43 1 0 41 0 44 1 0 38 0 45 2 6 0
    0 0 46 2 0 0 0 0 47 2 0 0 6 0 48 2 0
    0 0 6 49 1 6 0 0 50 1 0 0 0 51 2 0 0
    0 0 52 2 0 0 6 0 53 2 0 0 0 6 54 0 0
    0 55 2 6 0 0 0 56 1 0 0 0 57 2 6 0 0
    0 58 2 0 0 0 0 59 2 0 0 6 0 60 2 0 0
    0 6 61 0 6 0 62 0 0 0 63 1 6 64 0 65
    1 0 64 0 66 1 0 64 0 67 1 0 64 0 68 1
    6 0 0 69 1 0 0 0 70 0 6 71 72 0 0 71
    73 1 6 0 74 75 2 0 0 74 0 76 1 79 77
    78 80 1 77 82 0 83 2 6 0 0 25 84 2 0
    0 0 25 85 1 86 0 71 87 0 0 86 88 2 6
    0 0 0 89 2 0 0 0 0 90 2 6 0 0 91 92 2
    0 0 0 0 93 2 2 38 0 0 1 1 20 38 0 1 1
    0 0 0 22 2 20 64 0 0 1 2 10 0 0 0 93
    0 22 0 1 1 20 0 0 8 1 16 64 0 68 1 0
    6 0 21 1 16 64 0 66 1 16 38 0 1 2 0 0
    23 0 24 1 0 6 0 9 1 20 0 0 57 1 16 64
    0 67 1 2 97 0 1 1 11 0 0 70 1 2 96 0
    1 1 9 18 0 19 2 8 0 0 0 34 2 8 0 0 18
    37 3 7 0 0 25 6 27 3 7 0 0 28 29 31 2
    0 0 6 6 17 0 1 86 88 2 4 0 0 28 1 2 4

```

```

0 0 25 85 3 4 0 0 28 95 1 3 4 0 0 25
71 1 2 6 0 0 0 1 2 6 0 0 0 1 1 3 0 74
1 1 2 41 0 44 1 2 38 0 45 0 3 71 73 2
6 0 0 74 1 2 16 0 0 71 1 2 18 0 0 94
1 0 20 0 55 0 16 0 63 2 4 0 0 28 1 2
4 0 0 25 1 3 4 0 0 28 95 1 3 4 0 0 25
71 1 2 2 38 0 0 40 2 0 0 6 6 20 2 11
0 0 0 90 2 1 0 0 6 1 1 20 0 0 51 2 20
0 0 0 52 2 20 0 6 0 53 2 20 0 0 6 54
2 23 0 0 0 47 2 23 0 6 0 48 2 23 0 0
6 49 2 6 0 0 74 1 2 16 0 0 71 1 2 18
0 0 94 1 2 20 0 71 0 1 2 20 0 74 0 76
2 23 0 94 0 1 2 18 0 0 0 59 2 18 0 0
6 61 2 18 0 6 0 60))))))
'lookupComplete))

```

1.7.3 The “compilerInfo”

```

(SETQ |$CategoryFrame|
  (|put| '|Equation| '|isFunction|
    '(((|eval| ($ $ (|List| (|Symbol|)) (|List| |#1|)))
      (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))
      (ELT $ 31))
    ((|eval| ($ $ (|Symbol|) |#1|))
      (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))
      (ELT $ 27))
    ((~= ((|Boolean|) $ $)) (|has| |#1| (|SetCategory|))
      (ELT $ NIL))
    ((= ((|Boolean|) $ $)) (|has| |#1| (|SetCategory|))
      (ELT $ 40))
    ((|coerce| ((|OutputForm|) $))
      (|has| |#1| (|SetCategory|)) (ELT $ 44))
    ((|hash| ((|SingleInteger|) $))
      (|has| |#1| (|SetCategory|)) (ELT $ NIL))
    ((|latex| ((|String|) $)) (|has| |#1| (|SetCategory|))
      (ELT $ NIL))
    ((|coerce| ((|Boolean|) $)) (|has| |#1| (|SetCategory|))
      (ELT $ 45))
    ((+ ($ $ $)) (|has| |#1| (|AbelianSemiGroup|))
      (ELT $ 47))
    ((* ($ (|PositiveInteger|) $))
      (|has| |#1| (|AbelianSemiGroup|)) (ELT $ NIL))
    ((|Zero| ($)) (|has| |#1| (|AbelianGroup|))
      (CONST $ 55))
    ((|sample| ($))
      (OR (|has| |#1| (|AbelianGroup|))
        (|has| |#1| (|Monoid|)))
      (CONST $ NIL))
    ((|zero?| ((|Boolean|) $)) (|has| |#1| (|AbelianGroup|))

```

```

(ELT $ NIL))
(((* ($ (|NonNegativeInteger|) $))
  (|has| |#1| (|AbelianGroup|)) (ELT $ NIL))
(|subtractIfCan| ((|Union| $ "failed") $ $))
  (|has| |#1| (|AbelianGroup|)) (ELT $ NIL))
((- ($ $)) (|has| |#1| (|AbelianGroup|)) (ELT $ 51))
((- ($ $ $)) (|has| |#1| (|AbelianGroup|)) (ELT $ 52))
(((* ($ (|Integer|) $)) (|has| |#1| (|AbelianGroup|))
  (ELT $ 76))
(((* ($ $ $)) (|has| |#1| (|SemiGroup|)) (ELT $ 59))
((** ($ $ (|PositiveInteger|)))
  (|has| |#1| (|SemiGroup|)) (ELT $ NIL))
((^ ($ $ (|PositiveInteger|)))
  (|has| |#1| (|SemiGroup|)) (ELT $ NIL))
(|One| ($)) (|has| |#1| (|Monoid|)) (CONST $ 63))
(|one?| ((|Boolean|) $)) (|has| |#1| (|Monoid|))
  (ELT $ NIL))
((** ($ $ (|NonNegativeInteger|)))
  (|has| |#1| (|Monoid|)) (ELT $ NIL))
((^ ($ $ (|NonNegativeInteger|)))
  (|has| |#1| (|Monoid|)) (ELT $ NIL))
(|recip| ((|Union| $ "failed") $))
  (|has| |#1| (|Monoid|)) (ELT $ 66))
(|inv| ($ $))
  (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))
  (ELT $ 70))
(|/| ($ $ $))
  (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))
  (ELT $ 90))
((** ($ $ (|Integer|))) (|has| |#1| (|Group|))
  (ELT $ NIL))
((^ ($ $ (|Integer|))) (|has| |#1| (|Group|))
  (ELT $ NIL))
(|conjugate| ($ $ $)) (|has| |#1| (|Group|))
  (ELT $ NIL))
(|commutator| ($ $ $)) (|has| |#1| (|Group|))
  (ELT $ NIL))
(|characteristic| ((|NonNegativeInteger|)))
  (|has| |#1| (|Ring|)) (ELT $ 73))
(|coerce| ($ (|Integer|))) (|has| |#1| (|Ring|))
  (ELT $ NIL))
(((* ($ |#1| $)) (|has| |#1| (|SemiGroup|)) (ELT $ 60))
((($ $ |#1|)) (|has| |#1| (|SemiGroup|)) (ELT $ 61))
(|differentiate| ($ $ (|Symbol|)))
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ELT $ 85))
(|differentiate| ($ $ (|List| (|Symbol|))))
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ELT $ NIL))
(|differentiate|

```

```

($ $ (|Symbol|) (|NonNegativeInteger|))
(|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
(ELT $ NIL))
((|differentiate|
  ($ $ (|List| (|Symbol|))
    (|List| (|NonNegativeInteger|))))
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ELT $ NIL))
((D ($ $ (|Symbol|))
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ELT $ NIL))
  ((D ($ $ (|List| (|Symbol|)))
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
    (ELT $ NIL))
    ((D ($ $ (|Symbol|) (|NonNegativeInteger|))
      (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
      (ELT $ NIL))
      ((D ($ $ (|List| (|Symbol|))
        (|List| (|NonNegativeInteger|))))
        (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
        (ELT $ NIL))
        (/( $ $ |#1|) (|has| |#1| (|Field|)) (ELT $ NIL))
        ((|dimension| ((|CardinalNumber|))
          (|has| |#1| (|Field|)) (ELT $ 88))
          ((|subst| ($ $ $)) (|has| |#1| (|ExpressionSpace|))
            (ELT $ 93))
          ((|factorAndSplit| ((|List| $) $))
            (|has| |#1| (|IntegralDomain|)) (ELT $ 19))
          ((|rightOne| ((|Union| $ "failed") $))
            (|has| |#1| (|Monoid|)) (ELT $ 68))
          ((|leftOne| ((|Union| $ "failed") $))
            (|has| |#1| (|Monoid|)) (ELT $ 67))
          ((- ($ $ |#1|)) (|has| |#1| (|AbelianGroup|))
            (ELT $ 54))
          ((- ($ $ |#1| $)) (|has| |#1| (|AbelianGroup|))
            (ELT $ 53))
          ((|rightZero| ($ $)) (|has| |#1| (|AbelianGroup|))
            (ELT $ 8))
          ((|leftZero| ($ $)) (|has| |#1| (|AbelianGroup|))
            (ELT $ 57))
          ((+ ($ $ |#1|)) (|has| |#1| (|AbelianSemiGroup|))
            (ELT $ 49))
          ((+ ($ $ |#1| $)) (|has| |#1| (|AbelianSemiGroup|))
            (ELT $ 48))
          ((|eval| ($ $ (|List| $)))
            (AND (|has| |#1| (|Evalable| |#1|))
              (|has| |#1| (|SetCategory|)))
            (ELT $ 37))
          ((|eval| ($ $ $))
            (AND (|has| |#1| (|Evalable| |#1|))

```

```

(|has| |#1| (|SetCategory|)))
(ELT $ 34))
((|map| ($ (|Mapping| |#1| |#1|) $)) T (ELT $ 24))
((|rhs| (|#1| $)) T (ELT $ 21))
((|lhs| (|#1| $)) T (ELT $ 9))
((|swap| ($ $)) T (ELT $ 22))
((|equation| ($ |#1| |#1|)) T (ELT $ 17))
((= ($ |#1| |#1|)) T (ELT $ 20)))
(|addModemap| '|Equation| '(|Equation| |#1|)
'(|Join| (|Type|)
(CATEGORY |domain|
(SIGNATURE = ($ |#1| |#1|))
(SIGNATURE |equation| ($ |#1| |#1|))
(SIGNATURE |swap| ($ $))
(SIGNATURE |lhs| (|#1| $))
(SIGNATURE |rhs| (|#1| $))
(SIGNATURE |map|
($ (|Mapping| |#1| |#1|) $))
(IF (|has| |#1|
(|InnerEvalable| (|Symbol|) |#1|))
(ATTRIBUTE
(|InnerEvalable| (|Symbol|) |#1|))
|noBranch|)
(IF (|has| |#1| (|SetCategory|))
(PROGN
(ATTRIBUTE (|SetCategory|))
(ATTRIBUTE
(|CoercibleTo| (|Boolean|)))
(IF (|has| |#1| (|Evalable| |#1|))
(PROGN
(SIGNATURE |eval| ($ $ $))
(SIGNATURE |eval|
($ $ (|List| $))))
|noBranch|))
|noBranch|)
(IF (|has| |#1| (|AbelianSemiGroup|))
(PROGN
(ATTRIBUTE (|AbelianSemiGroup|))
(SIGNATURE + ($ |#1| $))
(SIGNATURE + ($ $ |#1|))
|noBranch|)
(IF (|has| |#1| (|AbelianGroup|))
(PROGN
(ATTRIBUTE (|AbelianGroup|))
(SIGNATURE |leftZero| ($ $))
(SIGNATURE |rightZero| ($ $))
(SIGNATURE - ($ |#1| $))
(SIGNATURE - ($ $ |#1|))
|noBranch|)
(IF (|has| |#1| (|SemiGroup|))

```

```

      (PROGN
        (ATTRIBUTE (|SemiGroup|))
        (SIGNATURE * ($ |#1| $))
        (SIGNATURE * ($ $ |#1|)))
      |noBranch|)
    (IF (|has| |#1| (|Monoid|))
      (PROGN
        (ATTRIBUTE (|Monoid|))
        (SIGNATURE |leftOne|
          ((|Union| $ "failed") $))
        (SIGNATURE |rightOne|
          ((|Union| $ "failed") $)))
        |noBranch|)
    (IF (|has| |#1| (|Group|))
      (PROGN
        (ATTRIBUTE (|Group|))
        (SIGNATURE |leftOne|
          ((|Union| $ "failed") $))
        (SIGNATURE |rightOne|
          ((|Union| $ "failed") $)))
        |noBranch|)
    (IF (|has| |#1| (|Ring|))
      (PROGN
        (ATTRIBUTE (|Ring|))
        (ATTRIBUTE (|BiModule| |#1| |#1|)))
        |noBranch|)
    (IF (|has| |#1| (|CommutativeRing|))
      (ATTRIBUTE (|Module| |#1|))
      |noBranch|)
    (IF (|has| |#1| (|IntegralDomain|))
      (SIGNATURE |factorAndSplit|
        ((|List| $) $))
      |noBranch|)
    (IF (|has| |#1|
      (|PartialDifferentialRing|
        (|Symbol|)))
      (ATTRIBUTE
        (|PartialDifferentialRing|
          (|Symbol|)))
      |noBranch|)
    (IF (|has| |#1| (|Field|))
      (PROGN
        (ATTRIBUTE (|VectorSpace| |#1|))
        (SIGNATURE / ($ $ $))
        (SIGNATURE |inv| ($ $)))
        |noBranch|)
    (IF (|has| |#1| (|ExpressionSpace|))
      (SIGNATURE |subst| ($ $ $))
      |noBranch|)))
(|Type|))

```

```

T '|Equation|
(|put| '|Equation| '|model|
  '|Mapping|
    (|Join| (|Type|)
      (CATEGORY |domain|
        (SIGNATURE = ($ |#1| |#1|))
        (SIGNATURE |equation|
          ($ |#1| |#1|))
        (SIGNATURE |swap| ($ $))
        (SIGNATURE |lhs| (|#1| $))
        (SIGNATURE |rhs| (|#1| $))
        (SIGNATURE |map|
          ($ (|Mapping| |#1| |#1|) $))
        (IF
          (|has| |#1|
            (|InnerEvalable| (|Symbol|)
              |#1|))
          (ATTRIBUTE
            (|InnerEvalable| (|Symbol|)
              |#1|))
          |noBranch|)
        (IF (|has| |#1| (|SetCategory|))
          (PROGN
            (ATTRIBUTE (|SetCategory|))
            (ATTRIBUTE
              (|CoercibleTo| (|Boolean|)))
            (IF
              (|has| |#1|
                (|Evalable| |#1|))
              (PROGN
                (SIGNATURE |eval| ($ $ $))
                (SIGNATURE |eval|
                  ($ $ (|List| $))))
              |noBranch|))
          |noBranch|)
        (IF
          (|has| |#1|
            (|AbelianSemiGroup|))
          (PROGN
            (ATTRIBUTE
              (|AbelianSemiGroup|))
            (SIGNATURE + ($ |#1| $))
            (SIGNATURE + ($ $ |#1|)))
          |noBranch|)
        (IF (|has| |#1| (|AbelianGroup|))
          (PROGN
            (ATTRIBUTE (|AbelianGroup|))
            (SIGNATURE |leftZero| ($ $))
            (SIGNATURE |rightZero| ($ $))
            (SIGNATURE - ($ |#1| $))

```

```

(SIGNATURE - ($ $ |#1|))
|noBranch|
(IF (|has| |#1| (|SemiGroup|))
  (PROGN
    (ATTRIBUTE (|SemiGroup|))
    (SIGNATURE * ($ |#1| $))
    (SIGNATURE * ($ $ |#1|))
    |noBranch|)
  (IF (|has| |#1| (|Monoid|))
    (PROGN
      (ATTRIBUTE (|Monoid|))
      (SIGNATURE |leftOne|
        ((|Union| $ "failed") $))
      (SIGNATURE |rightOne|
        ((|Union| $ "failed") $)))
      |noBranch|)
    (IF (|has| |#1| (|Group|))
      (PROGN
        (ATTRIBUTE (|Group|))
        (SIGNATURE |leftOne|
          ((|Union| $ "failed") $))
        (SIGNATURE |rightOne|
          ((|Union| $ "failed") $)))
        |noBranch|)
      (IF (|has| |#1| (|Ring|))
        (PROGN
          (ATTRIBUTE (|Ring|))
          (ATTRIBUTE
            (|BiModule| |#1| |#1|))
          |noBranch|)
        (IF
          (|has| |#1| (|CommutativeRing|))
          (ATTRIBUTE (|Module| |#1|))
          |noBranch|)
        (IF
          (|has| |#1| (|IntegralDomain|))
          (SIGNATURE |factorAndSplit|
            ((|List| $) $))
          |noBranch|)
        (IF
          (|has| |#1|
            (|PartialDifferentialRing|
              (|Symbol|)))
          (ATTRIBUTE
            (|PartialDifferentialRing|
              (|Symbol|)))
          |noBranch|)
        (IF (|has| |#1| (|Field|))
          (PROGN
            (ATTRIBUTE

```



```

(|VectorSpace| |#1|))
(SIGNATURE / ($ $ $))
(SIGNATURE |inv| ($ $ $))
|noBranch|)
(IF
(|has| |#1| (|ExpressionSpace|))
(SIGNATURE |subst| ($ $ $))
|noBranch|)))
(|Type|))
|$CategoryFrame|))))

```

1.7.4 The “constructorForm”

```
(|Equation| S)
```

1.7.5 The “constructorKind”

```
|domain|
```

1.7.6 The “constructorModemap”

```

(((|Equation| |#1|)
(|Join| (|Type|)
(CATEGORY |domain| (SIGNATURE = ($ |#1| |#1|))
(SIGNATURE |equation| ($ |#1| |#1|))
(SIGNATURE |swap| ($ $)) (SIGNATURE |lhs| (|#1| $))
(SIGNATURE |rhs| (|#1| $))
(SIGNATURE |map| ($ (|Mapping| |#1| |#1|) $))
(IF (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))
(ATTRIBUTE (|InnerEvalable| (|Symbol|) |#1|))
|noBranch|)
(IF (|has| |#1| (|SetCategory|))
(PROGN
(ATTRIBUTE (|SetCategory|))
(ATTRIBUTE (|CoercibleTo| (|Boolean|)))
(IF (|has| |#1| (|Evalable| |#1|))
(PROGN
(SIGNATURE |eval| ($ $ $))
(SIGNATURE |eval| ($ $ (|List| $))))
|noBranch|))
|noBranch|)
(IF (|has| |#1| (|AbelianSemiGroup|))
(PROGN
(ATTRIBUTE (|AbelianSemiGroup|))
(SIGNATURE + ($ |#1| $))
(SIGNATURE + ($ $ |#1|))
|noBranch|)

```

```

(IF (|has| |#1| (|AbelianGroup|))
  (PROGN
    (ATTRIBUTE (|AbelianGroup|))
    (SIGNATURE |leftZero| ($ $))
    (SIGNATURE |rightZero| ($ $))
    (SIGNATURE - ($ |#1| $))
    (SIGNATURE - ($ $ |#1|)))
  |noBranch|)
(IF (|has| |#1| (|SemiGroup|))
  (PROGN
    (ATTRIBUTE (|SemiGroup|))
    (SIGNATURE * ($ |#1| $))
    (SIGNATURE * ($ $ |#1|)))
  |noBranch|)
(IF (|has| |#1| (|Monoid|))
  (PROGN
    (ATTRIBUTE (|Monoid|))
    (SIGNATURE |leftOne| ((|Union| $ "failed") $))
    (SIGNATURE |rightOne| ((|Union| $ "failed") $)))
  |noBranch|)
(IF (|has| |#1| (|Group|))
  (PROGN
    (ATTRIBUTE (|Group|))
    (SIGNATURE |leftOne| ((|Union| $ "failed") $))
    (SIGNATURE |rightOne| ((|Union| $ "failed") $)))
  |noBranch|)
(IF (|has| |#1| (|Ring|))
  (PROGN
    (ATTRIBUTE (|Ring|))
    (ATTRIBUTE (|BiModule| |#1| |#1|)))
  |noBranch|)
(IF (|has| |#1| (|CommutativeRing|))
  (ATTRIBUTE (|Module| |#1|)) |noBranch|)
(IF (|has| |#1| (|IntegralDomain|))
  (SIGNATURE |factorAndSplit| ((|List| $) $))
  |noBranch|)
(IF (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ATTRIBUTE (|PartialDifferentialRing| (|Symbol|)))
  |noBranch|)
(IF (|has| |#1| (|Field|))
  (PROGN
    (ATTRIBUTE (|VectorSpace| |#1|))
    (SIGNATURE / ($ $ $))
    (SIGNATURE |inv| ($ $)))
  |noBranch|)
(IF (|has| |#1| (|ExpressionSpace|))
  (SIGNATURE |subst| ($ $ $)) |noBranch|))
(|Type|)
(T |Equation|)

```

1.7.7 The “constructorCategory”

```
(|Join| (|Type|)
  (CATEGORY |domain| (SIGNATURE = ($ |#1| |#1|))
    (SIGNATURE |equation| ($ |#1| |#1|))
    (SIGNATURE |swap| ($ $)) (SIGNATURE |lhs| (|#1| $))
    (SIGNATURE |rhs| (|#1| $))
    (SIGNATURE |map| ($ (|Mapping| |#1| |#1|) $))
    (IF (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))
      (ATTRIBUTE (|InnerEvalable| (|Symbol|) |#1|))
      |noBranch|)
    (IF (|has| |#1| (|SetCategory|))
      (PROGN
        (ATTRIBUTE (|SetCategory|))
        (ATTRIBUTE (|CoercibleTo| (|Boolean|)))
        (IF (|has| |#1| (|Evalable| |#1|))
          (PROGN
            (SIGNATURE |eval| ($ $ $))
            (SIGNATURE |eval| ($ $ (|List| $))))
          |noBranch|))
      |noBranch|)
    (IF (|has| |#1| (|AbelianSemiGroup|))
      (PROGN
        (ATTRIBUTE (|AbelianSemiGroup|))
        (SIGNATURE + ($ |#1| $))
        (SIGNATURE + ($ $ |#1|)))
      |noBranch|)
    (IF (|has| |#1| (|AbelianGroup|))
      (PROGN
        (ATTRIBUTE (|AbelianGroup|))
        (SIGNATURE |leftZero| ($ $))
        (SIGNATURE |rightZero| ($ $))
        (SIGNATURE - ($ |#1| $))
        (SIGNATURE - ($ $ |#1|)))
      |noBranch|)
    (IF (|has| |#1| (|SemiGroup|))
      (PROGN
        (ATTRIBUTE (|SemiGroup|))
        (SIGNATURE * ($ |#1| $))
        (SIGNATURE * ($ $ |#1|)))
      |noBranch|)
    (IF (|has| |#1| (|Monoid|))
      (PROGN
        (ATTRIBUTE (|Monoid|))
        (SIGNATURE |leftOne| ((|Union| $ "failed") $))
        (SIGNATURE |rightOne| ((|Union| $ "failed") $)))
      |noBranch|)
    (IF (|has| |#1| (|Group|))
      (PROGN
        (ATTRIBUTE (|Group|))
```

```

        (SIGNATURE |leftOne| ((|Union| $ "failed") $))
        (SIGNATURE |rightOne| ((|Union| $ "failed") $)))
|noBranch|)
(IF (|has| |#1| (|Ring|))
  (PROGN
    (ATTRIBUTE (|Ring|))
    (ATTRIBUTE (|BiModule| |#1| |#1|)))
|noBranch|)
(IF (|has| |#1| (|CommutativeRing|))
  (ATTRIBUTE (|Module| |#1|)) |noBranch|)
(IF (|has| |#1| (|IntegralDomain|))
  (SIGNATURE |factorAndSplit| ((|List| $) $)) |noBranch|)
(IF (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ATTRIBUTE (|PartialDifferentialRing| (|Symbol|)))
|noBranch|)
(IF (|has| |#1| (|Field|))
  (PROGN
    (ATTRIBUTE (|VectorSpace| |#1|))
    (SIGNATURE / ($ $ $))
    (SIGNATURE |inv| ($ $)))
|noBranch|)
(IF (|has| |#1| (|ExpressionSpace|))
  (SIGNATURE |subst| ($ $ $)) |noBranch|)))

```

1.7.8 The “sourceFile”

"/research/test/int/algebra/EQ.spad"

1.7.9 The “modemaps”

```

((= (*1 *1 *2 *2)
  (AND (|isDomain| *1 (|Equation| *2)) (|ofCategory| *2 (|Type|))))
  (|equation| (*1 *1 *2 *2)
    (AND (|isDomain| *1 (|Equation| *2)) (|ofCategory| *2 (|Type|))))
  (|swap| (*1 *1 *1)
    (AND (|isDomain| *1 (|Equation| *2))
      (|ofCategory| *2 (|Type|))))
  (|lhs| (*1 *2 *1)
    (AND (|isDomain| *1 (|Equation| *2))
      (|ofCategory| *2 (|Type|))))
  (|rhs| (*1 *2 *1)
    (AND (|isDomain| *1 (|Equation| *2))
      (|ofCategory| *2 (|Type|))))
  (|map| (*1 *1 *2 *1)
    (AND (|isDomain| *2 (|Mapping| *3 *3))
      (|ofCategory| *3 (|Type|))
      (|isDomain| *1 (|Equation| *3))))
  (|eval| (*1 *1 *1 *1)

```

```

      (AND (|ofCategory| *2 (|Evalable| *2))
            (|ofCategory| *2 (|SetCategory|))
            (|ofCategory| *2 (|Type|))
            (|isDomain| *1 (|Equation| *2))))
(|eval| (*1 *1 *1 *2)
      (AND (|isDomain| *2 (|List| (|Equation| *3)))
            (|ofCategory| *3 (|Evalable| *3))
            (|ofCategory| *3 (|SetCategory|))
            (|ofCategory| *3 (|Type|))
            (|isDomain| *1 (|Equation| *3))))
(+ (*1 *1 *2 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianSemiGroup|))
        (|ofCategory| *2 (|Type|))))
(+ (*1 *1 *1 *2)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianSemiGroup|))
        (|ofCategory| *2 (|Type|))))
(|leftZero| (*1 *1 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianGroup|))
        (|ofCategory| *2 (|Type|))))
(|rightZero| (*1 *1 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianGroup|))
        (|ofCategory| *2 (|Type|))))
(- (*1 *1 *2 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianGroup|)) (|ofCategory| *2 (|Type|))))
(- (*1 *1 *1 *2)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianGroup|)) (|ofCategory| *2 (|Type|))))
(|leftOne| (*1 *1 *1)
  (|partial| AND (|isDomain| *1 (|Equation| *2))
                 (|ofCategory| *2 (|Monoid|)) (|ofCategory| *2 (|Type|))))
(|rightOne| (*1 *1 *1)
  (|partial| AND (|isDomain| *1 (|Equation| *2))
                 (|ofCategory| *2 (|Monoid|)) (|ofCategory| *2 (|Type|))))
(|factorAndSplit| (*1 *2 *1)
  (AND (|isDomain| *2 (|List| (|Equation| *3)))
        (|isDomain| *1 (|Equation| *3))
        (|ofCategory| *3 (|IntegralDomain|))
        (|ofCategory| *3 (|Type|))))
(|subst| (*1 *1 *1 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|ExpressionSpace|))
        (|ofCategory| *2 (|Type|))))
(* (*1 *1 *1 *2)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|SemiGroup|)) (|ofCategory| *2 (|Type|))))

```

```

(* (*1 *1 *2 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|SemiGroup|)) (|ofCategory| *2 (|Type|))))
(/ (*1 *1 *1 *1)
  (OR (AND (|isDomain| *1 (|Equation| *2))
            (|ofCategory| *2 (|Field|)) (|ofCategory| *2 (|Type|)))
      (AND (|isDomain| *1 (|Equation| *2))
            (|ofCategory| *2 (|Group|)) (|ofCategory| *2 (|Type|))))
(|inv| (*1 *1 *1)
  (OR (AND (|isDomain| *1 (|Equation| *2))
            (|ofCategory| *2 (|Field|))
            (|ofCategory| *2 (|Type|)))
      (AND (|isDomain| *1 (|Equation| *2))
            (|ofCategory| *2 (|Group|))
            (|ofCategory| *2 (|Type|)))))

```

1.7.10 The “operationAlist”

```

((~= (((|Boolean|) $) $) NIL (|has| |#1| (|SetCategory|))))
(|zero?| (((|Boolean|) $) NIL (|has| |#1| (|AbelianGroup|))))
(|swap| (($ $) 22))
(|subtractIfCan|
  (((|Union| $ "failed") $) $) NIL (|has| |#1| (|AbelianGroup|)))
(|subst| (($ $ $) 93 (|has| |#1| (|ExpressionSpace|))))
(|sample|
  (($) NIL
    (OR (|has| |#1| (|AbelianGroup|)) (|has| |#1| (|Monoid|))) CONST))
(|rightZero| (($ $) 8 (|has| |#1| (|AbelianGroup|))))
(|rightOne| (((|Union| $ "failed") $) 68 (|has| |#1| (|Monoid|))))
(|rhs| ((|#1| $) 21))
(|recip| (((|Union| $ "failed") $) 66 (|has| |#1| (|Monoid|))))
(|one?| (((|Boolean|) $) NIL (|has| |#1| (|Monoid|))))
(|map| (($ (|Mapping| |#1| |#1|) $) 24) (|lhs| ((|#1| $) 9))
(|leftZero| (($ $) 57 (|has| |#1| (|AbelianGroup|))))
(|leftOne| (((|Union| $ "failed") $) 67 (|has| |#1| (|Monoid|))))
(|latex| (((|String|) $) NIL (|has| |#1| (|SetCategory|))))
(|inv| (($ $) 70 (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))))
(|hash| (((|SingleInteger|) $) NIL (|has| |#1| (|SetCategory|))))
(|factorAndSplit| (((|List| $) $) 19 (|has| |#1| (|IntegralDomain|))))
(|eval| (($ $ $) 34
  (AND (|has| |#1| (|Evalable| |#1|))
        (|has| |#1| (|SetCategory|))))
  (($ $ (|List| $)) 37
  (AND (|has| |#1| (|Evalable| |#1|))
        (|has| |#1| (|SetCategory|))))
  (($ $ (|Symbol|) |#1|) 27
  (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|)))
  (($ $ (|List| (|Symbol|)) (|List| |#1|)) 31
  (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))))

```

```
(equation| (($ |#1| |#1|) 17))
(|dimension| (((|CardinalNumber|) 88 (|has| |#1| (|Field|))))
(|differentiate|
  (($ $ (|List| (|Symbol|)) (|List| (|NonNegativeInteger|)) NIL
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|Symbol|) (|NonNegativeInteger|)) NIL
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|List| (|Symbol|)) NIL
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|Symbol|)) 85
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
(|conjugate| (($ $ $) NIL (|has| |#1| (|Group|)))
(|commutator| (($ $ $) NIL (|has| |#1| (|Group|)))
(|coerce| (($ (|Integer|)) NIL (|has| |#1| (|Ring|)))
  (((|Boolean|) $) 45 (|has| |#1| (|SetCategory|)))
  (((|OutputForm|) $) 44 (|has| |#1| (|SetCategory|)))
(|characteristic| (((|NonNegativeInteger|) 73 (|has| |#1| (|Ring|))))
(^ (($ $ (|Integer|)) NIL (|has| |#1| (|Group|)))
  (($ $ (|NonNegativeInteger|)) NIL (|has| |#1| (|Monoid|)))
  (($ $ (|PositiveInteger|)) NIL (|has| |#1| (|SemiGroup|)))
(|Zero| (($) 55 (|has| |#1| (|AbelianGroup|)) CONST))
(|One| (($) 63 (|has| |#1| (|Monoid|)) CONST))
(D (($ $ (|List| (|Symbol|)) (|List| (|NonNegativeInteger|)) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|Symbol|) (|NonNegativeInteger|)) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|List| (|Symbol|)) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|Symbol|)) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
(= (($ |#1| |#1|) 20)
  (((|Boolean|) $ $) 40 (|has| |#1| (|SetCategory|)))
(/ (($ $ |#1|) NIL (|has| |#1| (|Field|)))
  (($ $ $) 90 (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|))))
(- (($ |#1| $) 53 (|has| |#1| (|AbelianGroup|))
  (($ $ |#1|) 54 (|has| |#1| (|AbelianGroup|))
  (($ $ $) 52 (|has| |#1| (|AbelianGroup|))
  (($ $) 51 (|has| |#1| (|AbelianGroup|)))
(+ (($ |#1| $) 48 (|has| |#1| (|AbelianSemiGroup|))
  (($ $ |#1|) 49 (|has| |#1| (|AbelianSemiGroup|))
  (($ $ $) 47 (|has| |#1| (|AbelianSemiGroup|)))
(** (($ $ (|Integer|)) NIL (|has| |#1| (|Group|))
  (($ $ (|NonNegativeInteger|)) NIL (|has| |#1| (|Monoid|))
  (($ $ (|PositiveInteger|)) NIL (|has| |#1| (|SemiGroup|)))
(* (($ $ |#1|) 61 (|has| |#1| (|SemiGroup|))
  (($ |#1| $) 60 (|has| |#1| (|SemiGroup|))
  (($ $ $) 59 (|has| |#1| (|SemiGroup|))
  (($ (|Integer|) $) 76 (|has| |#1| (|AbelianGroup|))
  (($ (|NonNegativeInteger|) $) NIL (|has| |#1| (|AbelianGroup|))
  (($ (|PositiveInteger|) $) NIL (|has| |#1| (|AbelianSemiGroup|))))
```

1.7.11 The “superDomain”

1.7.12 The “signaturesAndLocals”

```

((|EQ;subst;3$;43| ($ $ $)) (|EQ;inv;2$;42| ($ $))
(|EQ;/;3$;41| ($ $ $)) (|EQ;dimension;Cn;40| ((|CardinalNumber|)))
(|EQ;differentiate;$S$;39| ($ $ (|Symbol|)))
(|EQ;factorAndSplit;$L;38| ((|List| $) $))
(|EQ;*;I2$;37| ($ (|Integer|) $))
(|EQ;characteristic;Nni;36| ((|NonNegativeInteger|)))
(|EQ;rightOne;$U;35| ((|Union| $ "failed") $))
(|EQ;leftOne;$U;34| ((|Union| $ "failed") $)) (|EQ;inv;2$;33| ($ $))
(|EQ;rightOne;$U;32| ((|Union| $ "failed") $))
(|EQ;leftOne;$U;31| ((|Union| $ "failed") $))
(|EQ;recip;$U;30| ((|Union| $ "failed") $)) (|EQ;One;$;29| ($))
(|EQ;*;$S$;28| ($ $ $ S)) (|EQ;*;S2$;27| ($ S $))
(|EQ;*;S2$;26| ($ S $)) (|EQ;*;3$;25| ($ $ $)) (|EQ;-;3$;24| ($ $ $))
(|EQ;Zero;$;23| ($)) (|EQ;rightZero;2$;22| ($ $))
(|EQ;leftZero;2$;21| ($ $)) (|EQ;-;$S$;20| ($ $ S))
(|EQ;-;S2$;19| ($ S $)) (|EQ;-;2$;18| ($ $)) (|EQ;+;$S$;17| ($ $ S))
(|EQ;+;S2$;16| ($ S $)) (|EQ;+;3$;15| ($ $ $))
(|EQ;coerce;$B;14| ((|Boolean|) $))
(|EQ;coerce;$Of;13| ((|OutputForm|) $))
(|EQ;=;2$B;12| ((|Boolean|) $ $)) (|EQ;eval;$L$;11| ($ $ (|List| $)))
(|EQ;eval;3$;10| ($ $ $))
(|EQ;eval;$LL$;9| ($ $ (|List| (|Symbol|)) (|List| S)))
(|EQ;eval;$SS$;8| ($ $ (|Symbol|) S))
(|EQ;map;M2$;7| ($ (|Mapping| S S) $)) (|EQ;swap;2$;6| ($ $))
(|EQ;rhs;$S;5| (S $)) (|EQ;lhs;$S;4| (S $))
(|EQ;equation;2S$;3| ($ S S)) (|EQ;=;2S$;2| ($ S S))
(|EQ;factorAndSplit;$L;1| ((|List| $) $)))

```

1.7.13 The “attributes”

```

((|unitsKnown| OR (|has| |#1| (|Ring|)) (|has| |#1| (|Group|)))
(|rightUnitary| |has| |#1| (|Ring|))
(|leftUnitary| |has| |#1| (|Ring|)))

```

1.7.14 The “predicates”

```

((|HasCategory| |#1| '(|Field|)) (|HasCategory| |#1| '(|SetCategory|))
(|HasCategory| |#1| '(|Ring|))
(|HasCategory| |#1| (LIST '(|PartialDifferentialRing| '(|Symbol|)))
(OR (|HasCategory| |#1| (LIST '(|PartialDifferentialRing| '(|Symbol|)))
(|HasCategory| |#1| '(|Ring|)))
(|HasCategory| |#1| '(|Group|))
(|HasCategory| |#1|
(LIST '(|InnerEvalable| '(|Symbol|) (|devaluate| |#1|))))

```



```

(AND (|HasCategory| |#1| (LIST ' |Evalable| (|devaluate| |#1|)))
      (|HasCategory| |#1| ' (|SetCategory|)))
(|HasCategory| |#1| ' (|IntegralDomain|))
(|HasCategory| |#1| ' (|ExpressionSpace|))
(OR (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Group|)))
(OR (|HasCategory| |#1| ' (|Group|)) (|HasCategory| |#1| ' (|Ring|)))
(|HasCategory| |#1| ' (|CommutativeRing|))
(OR (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Ring|)))
(OR (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)))
(|HasCategory| |#1| ' (|Monoid|))
(OR (|HasCategory| |#1| ' (|Group|)) (|HasCategory| |#1| ' (|Monoid|)))
(|HasCategory| |#1| ' (|SemiGroup|))
(OR (|HasCategory| |#1| ' (|Group|)) (|HasCategory| |#1| ' (|Monoid|))
      (|HasCategory| |#1| ' (|SemiGroup|)))
(|HasCategory| |#1| ' (|AbelianGroup|))
(OR (|HasCategory| |#1| (LIST ' |PartialDifferentialRing| ' (|Symbol|)))
      (|HasCategory| |#1| ' (|AbelianGroup|))
      (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Ring|)))
(OR (|HasCategory| |#1| ' (|AbelianGroup|))
      (|HasCategory| |#1| ' (|Monoid|)))
(|HasCategory| |#1| ' (|AbelianSemiGroup|))
(OR (|HasCategory| |#1| (LIST ' |PartialDifferentialRing| ' (|Symbol|)))
      (|HasCategory| |#1| ' (|AbelianGroup|))
      (|HasCategory| |#1| ' (|AbelianSemiGroup|))
      (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Ring|)))
(OR (|HasCategory| |#1| (LIST ' |PartialDifferentialRing| ' (|Symbol|)))
      (|HasCategory| |#1| ' (|AbelianGroup|))
      (|HasCategory| |#1| ' (|AbelianSemiGroup|))
      (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Group|))
      (|HasCategory| |#1| ' (|Monoid|)) (|HasCategory| |#1| ' (|Ring|))
      (|HasCategory| |#1| ' (|SemiGroup|))
      (|HasCategory| |#1| ' (|SetCategory|))))

```

1.7.15 The “abbreviation”

EQ

1.7.16 The “parents”

```

(((|Type|) . T)
  ((|InnerEvalable| (|Symbol|) S) |has| S
   (|InnerEvalable| (|Symbol|) S))
  ((|CoercibleTo| (|Boolean|)) |has| S (|SetCategory|))

```

```

((|SetCategory|) |has| S (|SetCategory|))
((|AbelianSemiGroup|) |has| S (|AbelianSemiGroup|))
((|AbelianGroup|) |has| S (|AbelianGroup|))
((|SemiGroup|) |has| S (|SemiGroup|)) ((|Monoid|) |has| S (|Monoid|))
((|Group|) |has| S (|Group|)) ((|BiModule| S S) |has| S (|Ring|))
((|Ring|) |has| S (|Ring|)) ((|Module| S) |has| S (|CommutativeRing|))
((|PartialDifferentialRing| (|Symbol|)) |has| S
  (|PartialDifferentialRing| (|Symbol|)))
((|VectorSpace| S) |has| S (|Field|))

```

1.7.17 The “ancestors”

```

(((|AbelianGroup|) |has| S (|AbelianGroup|))
  ((|AbelianMonoid|) |has| S (|AbelianGroup|))
  ((|AbelianSemiGroup|) |has| S (|AbelianSemiGroup|))
  ((|BasicType|) |has| S (|SetCategory|))
  ((|BiModule| S S) |has| S (|Ring|))
  ((|CancellationAbelianMonoid|) |has| S (|AbelianGroup|))
  ((|CoercibleTo| (|OutputForm|)) |has| S (|SetCategory|))
  ((|CoercibleTo| (|Boolean|)) |has| S (|SetCategory|))
  ((|Group|) |has| S (|Group|))
  ((|InnerEvalable| (|Symbol|) S) |has| S
    (|InnerEvalable| (|Symbol|) S))
  ((|LeftModule| $) |has| S (|Ring|))
  ((|LeftModule| S) |has| S (|Ring|))
  ((|Module| S) |has| S (|CommutativeRing|))
  ((|Monoid|) |has| S (|Monoid|))
  ((|PartialDifferentialRing| (|Symbol|)) |has| S
    (|PartialDifferentialRing| (|Symbol|)))
  ((|RightModule| S) |has| S (|Ring|)) ((|Ring|) |has| S (|Ring|))
  ((|Rng|) |has| S (|Ring|)) ((|SemiGroup|) |has| S (|SemiGroup|))
  ((|SetCategory|) |has| S (|SetCategory|)) ((|Type|) . T)
  ((|VectorSpace| S) |has| S (|Field|))

```

1.7.18 The “documentation”

```

((|constructor|
  (NIL "Equations as mathematical objects. All properties of the basis
    domain,{ } \\spadignore{e.g.} being an abelian group are carried
    over the equation domain,{ } by performing the structural operations
    on the left and on the right hand side."))
  (|subst| (($ $ $)
    "\\spad{subst(eq1,{ }eq2)} substitutes \\spad{eq2} into both sides
    of \\spad{eq1} the \\spad{lhs} of \\spad{eq2} should be a kernel"))
  (|inv| (($ $)
    "\\spad{inv(x)} returns the multiplicative inverse of \\spad{x}.")
  (/ (($ $ $)
    "\\spad{e1/e2} produces a new equation by dividing the left and right

```

```

    hand sides of equations \spad{e1} and \spad{e2}.)")
(|factorAndSplit|
  (((|List| $) $)
    "\spad{factorAndSplit(eq)} make the right hand side 0 and factors the
    new left hand side. Each factor is equated to 0 and put into the
    resulting list without repetitions."))
(|rightOne|
  (((|Union| $ "failed") $)
    "\spad{rightOne(eq)} divides by the right hand side.")
  (((|Union| $ "failed") $)
    "\spad{rightOne(eq)} divides by the right hand side,{ } if possible."))
(|leftOne|
  (((|Union| $ "failed") $)
    "\spad{leftOne(eq)} divides by the left hand side.")
  (((|Union| $ "failed") $)
    "\spad{leftOne(eq)} divides by the left hand side,{ } if possible."))
(* (($ $ |#1|)
  "\spad{eqn*x} produces a new equation by multiplying both sides of
  equation eqn by \spad{x}."
  (($ |#1| $)
    "\spad{x*eqn} produces a new equation by multiplying both sides of
    equation eqn by \spad{x}."))
(- (($ $ |#1|)
  "\spad{eqn-x} produces a new equation by subtracting \spad{x} from
  both sides of equation eqn."
  (($ |#1| $)
    "\spad{x-eqn} produces a new equation by subtracting both sides of
    equation eqn from \spad{x}."))
(|rightZero|
  (($ $) "\spad{rightZero(eq)} subtracts the right hand side.")
(|leftZero|
  (($ $) "\spad{leftZero(eq)} subtracts the left hand side.")
(+ (($ $ |#1|)
  "\spad{eqn+x} produces a new equation by adding \spad{x} to both
  sides of equation eqn."
  (($ |#1| $)
    "\spad{x+eqn} produces a new equation by adding \spad{x} to both
    sides of equation eqn."))
(|eval| (($ $ (|List| $))
  "\spad{eval(eqn,{ } [x1=v1,{ } ... xn=vn])} replaces \spad{xi}
  by \spad{vi} in equation \spad{eqn}."
  (($ $ $)
    "\spad{eval(eqn,{ } x=f)} replaces \spad{x} by \spad{f} in
    equation \spad{eqn}."))
(|map| (($ (|Mapping| |#1| |#1|) $)
  "\spad{map(f,{ }eqn)} constructs a new equation by applying
  \spad{f} to both sides of \spad{eqn}."))
(|rhs| ((|#1| $)
  "\spad{rhs(eqn)} returns the right hand side of equation
  \spad{eqn}."))

```

```
(|lhs| ((|#1| $)
  "\\spad{lhs(eqn)} returns the left hand side of equation
  \\spad{eqn}."))
(|swap| (($ $)
  "\\spad{swap(eq)} interchanges left and right hand side of
  equation \\spad{eq}."))
(|equation|
  (($ |#1| |#1|) "\\spad{equation(a,{b})} creates an equation."))
(= (($ |#1| |#1|) "\\spad{a=b} creates an equation."))
```

1.7.19 The “slotInfo”

```
(|Equation|
  (NIL (~= ((38 0 0) NIL (|has| |#1| (|SetCategory|))))
    (|zero?| ((38 0) NIL (|has| |#1| (|AbelianGroup|))))
    (|swap| ((0 0) 22))
    (|subtractIfCan| ((64 0 0) NIL (|has| |#1| (|AbelianGroup|))))
    (|subst| ((0 0 0) 93 (|has| |#1| (|ExpressionSpace|))))
    (|sample|
      ((0) NIL
        (OR (|has| |#1| (|AbelianGroup|))
          (|has| |#1| (|Monoid|))))
      CONST))
    (|rightZero| ((0 0) 8 (|has| |#1| (|AbelianGroup|))))
    (|rightOne| ((64 0) 68 (|has| |#1| (|Monoid|))))
    (|rhs| ((6 0) 21))
    (|recip| ((64 0) 66 (|has| |#1| (|Monoid|))))
    (|one?| ((38 0) NIL (|has| |#1| (|Monoid|))))
    (|map| ((0 23 0) 24) (|lhs| ((6 0) 9))
    (|leftZero| ((0 0) 57 (|has| |#1| (|AbelianGroup|))))
    (|leftOne| ((64 0) 67 (|has| |#1| (|Monoid|))))
    (|latex| ((97 0) NIL (|has| |#1| (|SetCategory|))))
    (|inv| ((0 0) 70
      (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|))))
    (|hash| ((96 0) NIL (|has| |#1| (|SetCategory|))))
    (|factorAndSplit| ((18 0) 19 (|has| |#1| (|IntegralDomain|))))
    (|eval| ((0 0 28 29) 31
      (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|)))
      ((0 0 25 6) 27
        (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|)))
      ((0 0 18) 37
        (AND (|has| |#1| (|Evalable| |#1|))
          (|has| |#1| (|SetCategory|))))
      ((0 0 0) 34
        (AND (|has| |#1| (|Evalable| |#1|))
          (|has| |#1| (|SetCategory|))))
    (|equation| ((0 6 6) 17))
    (|dimension| ((86) 88 (|has| |#1| (|Field|))))
    (|differentiate|
```

```

((0 0 25 71) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
((0 0 28 95) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
((0 0 25) 85
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
((0 0 28) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
(|conjugate| ((0 0 0) NIL (|has| |#1| (|Group|))))
(|commutator| ((0 0 0) NIL (|has| |#1| (|Group|))))
(|coerce| ((38 0) 45 (|has| |#1| (|SetCategory|))))
  ((41 0) 44 (|has| |#1| (|SetCategory|)))
  ((0 74) NIL (|has| |#1| (|Ring|)))
(|characteristic| ((71) 73 (|has| |#1| (|Ring|))))
(^ ((0 0 94) NIL (|has| |#1| (|SemiGroup|)))
  ((0 0 71) NIL (|has| |#1| (|Monoid|)))
  ((0 0 74) NIL (|has| |#1| (|Group|))))
(|Zero| ((0) 55 (|has| |#1| (|AbelianGroup|)) CONST))
(|One| ((0) 63 (|has| |#1| (|Monoid|)) CONST))
(D ((0 0 25 71) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  ((0 0 28 95) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  ((0 0 25) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  ((0 0 28) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  ((0 6 6) 20) ((38 0 0) 40 (|has| |#1| (|SetCategory|))))
(/ ((0 0 6) NIL (|has| |#1| (|Field|)))
  ((0 0 0) 90
  (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))))
(- ((0 0 6) 54 (|has| |#1| (|AbelianGroup|)))
  ((0 6 0) 53 (|has| |#1| (|AbelianGroup|)))
  ((0 0 0) 52 (|has| |#1| (|AbelianGroup|)))
  ((0 0) 51 (|has| |#1| (|AbelianGroup|))))
(+ ((0 0 6) 49 (|has| |#1| (|AbelianSemiGroup|)))
  ((0 6 0) 48 (|has| |#1| (|AbelianSemiGroup|)))
  ((0 0 0) 47 (|has| |#1| (|AbelianSemiGroup|))))
(** ((0 0 94) NIL (|has| |#1| (|SemiGroup|)))
  ((0 0 71) NIL (|has| |#1| (|Monoid|)))
  ((0 0 74) NIL (|has| |#1| (|Group|))))
(* ((0 6 0) 60 (|has| |#1| (|SemiGroup|)))
  ((0 0 6) 61 (|has| |#1| (|SemiGroup|)))
  ((0 0 0) 59 (|has| |#1| (|SemiGroup|)))
  ((0 94 0) NIL (|has| |#1| (|AbelianSemiGroup|)))
  ((0 74 0) 76 (|has| |#1| (|AbelianGroup|)))
  ((0 71 0) NIL (|has| |#1| (|AbelianGroup|)))))

```

1.7.20 The “index”

```
((("slot1Info" 0 32444) ("documentation" 0 29640) ("ancestors" 0 28691)
  ("parents" 0 28077) ("abbreviation" 0 28074) ("predicates" 0 25442)
  ("attributes" 0 25304) ("signaturesAndLocals" 0 23933)
  ("superDomain" 0 NIL) ("operationAlist" 0 20053) ("modemaps" 0 17216)
  ("sourceFile" 0 17179) ("constructorCategory" 0 15220)
  ("constructorModemap" 0 13215) ("constructorKind" 0 13206)
  ("constructorForm" 0 13191) ("compilerInfo" 0 4433)
  ("loadTimeStuff" 0 20))
```


Chapter 2

Compiler top level

2.1 Global Data Structures

2.2 Pratt Parsing

Parsing involves understanding the association of symbols and operators. Vaughn Pratt [8] poses the question “Given a substring AEB where A takes a right argument, B a left, and E is an expression, does E associate with A or B?”.

Floyd [9] associates a precedence with operators, storing them in a table, called “binding powers”. The expression E would associate with the argument position having the highest binding power. This leads to a large set of numbers, one for every situation.

Pratt assigns data types to “classes” and then creates a total order on the classes. He lists, in ascending order, Outcomes, Booleans, Graphs (trees, lists, etc), Strings, Algebraics (e.g. Integer, complex numbers, polynomials, real arrays) and references (e.g. the left hand side of assignments). Thus, Strings \leq References. The key restriction is “that the class of the type at any argument that might participate in an association problem not be less than the class of the data type of the result of the function taking that argument”.

For a less-than comparison (“ $<$ ”) the argument types are Algebraics but the result type is Boolean. Since Algebraics are greater than Boolean we can associate the Algebraics together and apply them as arguments to the Boolean.

In more detail, there an “association” is a function of 4 types:

- a_A – The data type of the right argument
- r_A – The return type of the right argument
- a_B – The data type of the left argument
- r_B – The return type of the left argument

Note that the return types might depend on the type of the expression E . If all 4 are of the same class then the association is to the left.

Using these ideas and given the restriction above, Pratt proves that every association problem has at most one solution consistent with the data types of the associated operators.

Pratt proves that there exists an assignment of integers to the argument positions of each token in the language such that the correct association, if any, is always in the direction of the argument position with the larger number, with ties being broken to the left.

To construct the proper numbers, first assign even integers to the data type classes. Then to each argument position assign an integer lying strictly (where possible) between the integers corresponding to the classes of the argument and result types.

For tokens like “and”, “or”, +, *, and the Booleans and Algebras can be subdivided into pseudo-classes so that

terms < factors < primaries

Then + is defined over terms, * over factors, and over primaries with coercions allowed from primaries to factors to terms. To be consistent with Algol, the primaries should be a right associative class (e.g. xyz)

2.3)compile

This is the implementation of the)compile command.

You use this command to invoke the new Axiom library compiler or the old Axiom system compiler. The)compile system command is actually a combination of Axiom processing and a call to the Aldor compiler. It is performing double-duty, acting as a front-end to both the Aldor compiler and the old Axiom system compiler. (The old Axiom system compiler was written in Lisp and was an integral part of the Axiom environment. The Aldor compiler is written in C and executed by the operating system when called from within Axiom.)

User Level Required: compiler

Command Syntax:

```

)compile
)compile fileName
)compile fileName.spad
)compile directory/fileName.spad
)compile fileName )old
)compile fileName )translate
)compile fileName )quiet
)compile fileName )noquiet

```

```

)compile fileName )moreargs
)compile fileName )onlyargs
)compile fileName )break
)compile fileName )nobreak
)compile fileName )library
)compile fileName )nolibrary
)compile fileName )vartrace
)compile fileName )constructor nameOrAbbrev

```

These command forms invoke the Aldor compiler.

```

)compile fileName.as
)compile directory/fileName.as
)compile fileName.ao
)compile directory/fileName.ao
)compile fileName.al
)compile directory/fileName.al
)compile fileName.lsp
)compile directory/fileName.lsp
)compile fileName )new

```

Command Description:

The first thing `)compile` does is look for a source code filename among its arguments. Thus

```

)compile mycode.spad
)compile /u/jones/mycode.spad
)compile mycode

```

all invoke `)compiler` on the file `/u/jones/mycode.spad` if the current Axiom working directory is `/u/jones`. (Recall that you can set the working directory via the `)cd` command. If you don't set it explicitly, it is the directory from which you started Axiom.)

If you omit the file extension, the command looks to see if you have specified the `)new` or `)old` option. If you have given one of these options, the corresponding compiler is used.

The command first looks in the standard system directories for files with extension `.as`, `.ao` and `.al` and then files with extension `.spad`. The first file found has the appropriate compiler invoked on it. If the command cannot find a matching file, an error message is displayed and the command terminates.

The first thing `)compile` does is look for a source code filename among its arguments. Thus

```

)compile mycode
)co mycode
)co mycode.spad

```

all invoke `)compiler` on the file `/u/jones/mycode.spad` if the current Axiom working directory is `/u/jones`. Recall that you can set the working directory via the `)cd` command. If you don't set it explicitly, it is the directory from which you started Axiom.

This is frequently all you need to compile your file.

This simple command:

1. Invokes the Spad compiler and produces Lisp output.
2. Calls the Lisp compiler if the compilation was successful.
3. Uses the `)library` command to tell Axiom about the contents of your compiled file and arrange to have those contents loaded on demand.

Should you not want the `)library` command automatically invoked, call `)compile` with the `)nolibrary` option. For example,

```

)compile mycode )nolibrary

```

By default, the `)library` system command *exposes* all domains and categories it processes. This means that the Axiom interpreter will consider those domains and categories when it is trying to resolve a reference to a function. Sometimes domains and categories should not be exposed. For example, a domain may just be used privately by another domain and may not be meant for top-level use. The `)library` command should still be used, though, so that the code will be loaded on demand. In this case, you should use the `)nolibrary` option on `)compile` and the `)noexpose` option in the `)library` command. For example,

```

)compile mycode )nolibrary
)library mycode )noexpose

```

Once you have established your own collection of compiled code, you may find it handy to use the `)dir` option on the `)library` command. This causes `)library` to process all compiled code in the specified directory. For example,

```

)library )dir /u/jones/quantum

```

You must give an explicit directory after `)dir`, even if you want all compiled code in the current working directory processed, e.g.

```

)library )dir .

```

2.3.1 Spad compiler

This command compiles files with file extension `.spad` with the Spad system compiler.

The `)translate` option is used to invoke a special version of the old system compiler that will translate a `.spad` file to a `.as` file. That is, the `.spad` file will be parsed and analyzed and a file using the new syntax will be created.

By default, the `.as` file is created in the same directory as the `.spad` file. If that directory is not writable, the current directory is used. If the current directory is not writable, an error message is given and the command terminates. Note that `)translate` implies the `)old` option so the file extension can safely be omitted. If `)translate` is given, all other options are ignored. Please be aware that the translation is not necessarily one hundred percent complete or correct. You should attempt to compile the output with the Aldor compiler and make any necessary corrections.

You can compile category, domain, and package constructors contained in files with file extension `.spad`. You can compile individual constructors or every constructor in a file.

The full filename is remembered between invocations of this command and `)edit` commands. The sequence of commands

```
)compile matrix.spad
)edit
)compile
```

will call the compiler, edit, and then call the compiler again on the file **matrix.spad**. If you do not specify a *directory*, the working current directory is searched for the file. If the file is not found, the standard system directories are searched.

If you do not give any options, all constructors within a file are compiled. Each constructor should have an `)abbreviation` command in the file in which it is defined. We suggest that you place the `)abbreviation` commands at the top of the file in the order in which the constructors are defined.

The `)library` option causes directories containing the compiled code for each constructor to be created in the working current directory. The name of such a directory consists of the constructor abbreviation and the `.nrlib` file extension. For example, the directory containing the compiled code for the **MATRIX** constructor is called **MATRIX.nrlib**. The `)nolibrary` option says that such files should not be created. The default is `)library`. Note that the semantics of `)library` and `)nolibrary` for the new Aldor compiler and for the old system compiler are completely different.

The `)vartrace` option causes the compiler to generate extra code for the constructor to support conditional tracing of variable assignments. Without this option, this code is suppressed and one cannot use the `)vars` option for the trace command.

The `)constructor` option is used to specify a particular constructor to compile. All other constructors in the file are ignored. The constructor name or abbreviation follows `)constructor`. Thus either

```
)compile matrix.spad )constructor RectangularMatrix
```

or

```
)compile matrix.spad )constructor RMATRIX
```

compiles the `RectangularMatrix` constructor defined in `matrix.spad`.

The `)break` and `)nobreak` options determine what the spad compiler does when it encounters an error. `)break` is the default and it indicates that processing should stop at the first error. The value of the `)set break` variable then controls what happens.

2.4 Operator Precedence Table Initialization

```
; PURPOSE: This file sets up properties which are used by the Boot lexical
;           analyzer for bottom-up recognition of operators. Also certain
;           other character-class definitions are included, as well as
;           table accessing functions.
;
; ORGANIZATION: Each section is organized in terms of Creation and Access code.
;
;               1. Led and Nud Tables
;               2. GLIPH Table
;               3. RENAMETOK Table
;               4. GENERIC Table
;               5. Character syntax class predicates
```

2.4.1 LED and NUD Tables

```
; **** 1. LED and NUD Tables

; ** TABLE PURPOSE

; Led and Nud have to do with operators. An operator with a Led property takes
; an operand on its left (infix/suffix operator).

; An operator with a Nud takes no operand on its left (prefix/nilfix).
; Some have both (e.g. - ). This terminology is from the Pratt parser.
; The translator for Scratchpad II is a modification of the Pratt parser which
; branches to special handlers when it is most convenient and practical to
; do so (Pratt's scheme cannot handle local contexts very easily).

; Both LEDs and NUDs have right and left binding powers. This is meaningful
; for prefix and infix operators. These powers are stored as the values of
; the LED and NUD properties of an atom, if the atom has such a property.
; The format is:

;           <Operator Left-Binding-Power Right-Binding-Power <Special-Handler>>
```

```

; where the Special-Handler is the name of a function to be evaluated when that
; keyword is encountered.

; The default values of Left and Right Binding-Power are NIL.  NIL is a
; legitimate value signifying no precedence.  If the Special-Handler is NIL,
; this is just an ordinary operator (as opposed to a surfix operator like
; if-then-else).
;
; The Nud value gives the precedence when the operator is a prefix op.
; The Led value gives the precedence when the operator is an infix op.
; Each op has 2 priorities, left and right.
; If the right priority of the first is greater than or equal to the
; left priority of the second then collect the second operator into
; the right argument of the first operator.

```

— LEDNUDTables —

```

; ** TABLE CREATION

(defun makenewop (x y) (makeop x y '|PARSE-NewKEY|))

(defun makeop (x y keyname)
  (if (or (not (cdr x)) (numberp (second x)))
      (setq x (cons (first x) x)))
      (if (and (alpha-char-p (elt (princ-to-string (first x)) 0))
                (not (member (first x) (eval keyname))))
          (set keyname (cons (first x) (eval keyname))))
          (put (first x) y x)
          (second x)))

(setq |PARSE-NewKEY| nil) ;;list of keywords

(mapcar #'(LAMBDA(J) (MAKENEWOP J '|Led|))
  '((* 800 801) (|rem| 800 801) (|mod| 800 801)
    (|quo| 800 801) (|div| 800 801)
    (/ 800 801) (** 900 901) (^ 900 901)
    (|exquo| 800 801) (+ 700 701)
    (\- 700 701) (\-> 1001 1002) (\<- 1001 1002)
    (\: 996 997) (\:\: 996 997)
    (\@ 996 997) (|pretend| 995 996)
    (\.) (\! \! 1002 1001)
    (\, 110 111)
    (\; 81 82 (|PARSE-SemiColon|))
    (\< 400 400) (\> 400 400)
    (\<\< 400 400) (\>\> 400 400)
    (\<= 400 400) (\>= 400 400)
    (= 400 400) (^= 400 400)
    (\~= 400 400)

```

```

(|in| 400 400)      (|case| 400 400)
(|add| 400 120)     (|with| 2000 400 (|PARSE-InfixWith|))
(|has| 400 400)
(|where| 121 104)    ; must be 121 for SPAD, 126 for boot--> nboot
(|when| 112 190)
(|otherwise| 119 190 (|PARSE-Suffix|))
(|is| 400 400)      (|isnt| 400 400)
(|and| 250 251)     (|or| 200 201)
(|/\ 250 251)       (|\ / 200 201)
(|\.\. SEGMENT 401 699 (|PARSE-Seg|))
(=> 123 103)
(+-> 995 112)
(== DEF 122 121)
(==> MDEF 122 121)
(| 108 111)          ;was 190 190
(|:- LETD 125 124) (|:= LET 125 124)))

(mapcar #'(LAMBDA (J) (MAKENEWOP J '(|Nud|))
'(|for| 130 350 (|PARSE-Loop|))
(|while| 130 190 (|PARSE-Loop|))
(|until| 130 190 (|PARSE-Loop|))
(|repeat| 130 190 (|PARSE-Loop|))
(|import| 120 0 (|PARSE-Import|) )
(|unless|)
(|add| 900 120)
(|with| 1000 300 (|PARSE-With|))
(|has| 400 400)
(|- 701 700) ; right-prec. wants to be -1 + left-prec
;; (|+ 701 700)
(|# 999 998)
(|! 1002 1001)
(|' 999 999 (|PARSE-Data|))
(|<< 122 120 (|PARSE-LabelExpr|))
(|>>)
(|^ 260 259 NIL)
(|-> 1001 1002)
(|: 194 195)
(|not| 260 259 NIL)
(|~ 260 259 nil)
(|= 400 700)
(|return| 202 201 (|PARSE-Return|))
(|leave| 202 201 (|PARSE-Leave|))
(|exit| 202 201 (|PARSE-Exit|))
(|from|)
(|iterate|)
(|yield|)
(|if| 130 0 (|PARSE-Conditional|)) ; was 130
(| 0 190)
(|suchthat|)
(|then| 0 114)

```

```
(|else| 0 114)))
```

2.5 Glyph Table

Gliph is symbol clumps. The gliph property of a symbol gives the tree describing the tokens which begin with that symbol. The token reader uses the gliph property to determine the longest token. Thus `:=` is read as one token not as `:` followed by `=`.

— GLIPHTable —

```
(mapcar #'(lambda (x) (put (car x) 'gliph (cdr x)))
  '(
    ( \| (\))      )
    ( *  (*)      )
    ( \ ( (<) (\|) )
    ( +  (- (>))   )
    ( -  (>)       )
    ( <  (=) (<)   )
  ;;   ( /  (\\)      ) breaks */xxx
    ( \\ (/)       )
    ( >  (=) (>) (\))
    ( =  (= (>)) (>) )
    ( \. (\.)      )
    ( ^  (=)       )
    ( \~ (=)       )
    ( \: (=) (-) (\:)))
```

2.5.1 Rename Token Table

RENAMETOK defines alternate token strings which can be used for different keyboards which define equivalent tokens.

— RENAMETOKTable —

```
(mapcar
  #'(lambda (x) (put (car x) 'renametok (cadr x)) (makenewop x nil))
  '((\(\| \[]) ; (| |) means []
    (\|\) \])
    (\(< \{) ; (< >) means {}
    (>\) \})))
```

2.5.2 Generic function table

GENERIC operators be suffixed by \$ qualifications in SPAD code. \$ is then followed by a domain label, such as I for Integer, which signifies which domain the operator refers to. For example `+$Integer` is `+` for Integers.

— **GENERICTable** —

```
(mapcar #'(lambda (x) (put x 'generic 'true))
  '(- = * |rem| |mod| |quo| |div| / ** |exquo| + - < > <= >= ^= ))
```

2.6 Giant steps, Baby steps

We will walk through the compiler with the EQ.spad example using a Giant-steps, Baby-steps approach. That is, we will show the large scale (Giant) transformations at each stage of compilation and discuss the details (Baby) in subsequent chapters.

Chapter 3

The Parser

3.1 EQ.spad

We will explain the compilation function using the file `EQ.spad`. We trace the execution of the various functions to understand the actual call parameters and results returned. The `EQ.spad` file is:

```
)abbrev domain EQ Equation
--FOR THE BENEFIT OF LIBAXO GENERATION
++ Author: Stephen M. Watt, enhancements by Johannes Grabmeier
++ Date Created: April 1985
++ Date Last Updated: June 3, 1991; September 2, 1992
++ Basic Operations: =
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: equation
++ Examples:
++ References:
++ Description:
++ Equations as mathematical objects. All properties of the basis domain,
++ e.g. being an abelian group are carried over the equation domain, by
++ performing the structural operations on the left and on the
++ right hand side.
-- The interpreter translates "=" to "equation". Otherwise, it will
-- find a modemap for "=" in the domain of the arguments.

Equation(S: Type): public == private where
  Ex ==> OutputForm
  public ==> Type with
    "=": (S, S) -> $
    ++ a=b creates an equation.
```

```

equation: (S, S) -> $
  ++ equation(a,b) creates an equation.
swap: $ -> $
  ++ swap(eq) interchanges left and right hand side of equation eq.
lhs: $ -> S
  ++ lhs(eqn) returns the left hand side of equation eqn.
rhs: $ -> S
  ++ rhs(eqn) returns the right hand side of equation eqn.
map: (S -> S, $) -> $
  ++ map(f,eqn) constructs a new equation by applying f to both
  ++ sides of eqn.
if S has InnerEvalable(Symbol,S) then
  InnerEvalable(Symbol,S)
if S has SetCategory then
  SetCategory
  CoercibleTo Boolean
  if S has Evalable(S) then
    eval: ($, $) -> $
      ++ eval(eqn, x=f) replaces x by f in equation eqn.
    eval: ($, List $) -> $
      ++ eval(eqn, [x1=v1, ... xn=vn]) replaces xi by vi in equation eqn.
if S has AbelianSemiGroup then
  AbelianSemiGroup
  "+": (S, $) -> $
    ++ x+eqn produces a new equation by adding x to both sides of
    ++ equation eqn.
  "+": ($, S) -> $
    ++ eqn+x produces a new equation by adding x to both sides of
    ++ equation eqn.
if S has AbelianGroup then
  AbelianGroup
  leftZero : $ -> $
    ++ leftZero(eq) subtracts the left hand side.
  rightZero : $ -> $
    ++ rightZero(eq) subtracts the right hand side.
  "-": (S, $) -> $
    ++ x-eqn produces a new equation by subtracting both sides of
    ++ equation eqn from x.
  "-": ($, S) -> $
    ++ eqn-x produces a new equation by subtracting x from both sides of
    ++ equation eqn.
if S has SemiGroup then
  SemiGroup
  "*": (S, $) -> $
    ++ x*eqn produces a new equation by multiplying both sides of
    ++ equation eqn by x.
  "*": ($, S) -> $
    ++ eqn*x produces a new equation by multiplying both sides of
    ++ equation eqn by x.
if S has Monoid then

```

```

Monoid
leftOne : $ -> Union($,"failed")
  ++ leftOne(eq) divides by the left hand side, if possible.
rightOne : $ -> Union($,"failed")
  ++ rightOne(eq) divides by the right hand side, if possible.
if S has Group then
  Group
  leftOne : $ -> Union($,"failed")
    ++ leftOne(eq) divides by the left hand side.
  rightOne : $ -> Union($,"failed")
    ++ rightOne(eq) divides by the right hand side.
if S has Ring then
  Ring
  BiModule(S,S)
if S has CommutativeRing then
  Module(S)
  --Algebra(S)
if S has IntegralDomain then
  factorAndSplit : $ -> List $
    ++ factorAndSplit(eq) make the right hand side 0 and
    ++ factors the new left hand side. Each factor is equated
    ++ to 0 and put into the resulting list without repetitions.
if S has PartialDifferentialRing(Symbol) then
  PartialDifferentialRing(Symbol)
if S has Field then
  VectorSpace(S)
  "/": ($, $) -> $
    ++ e1/e2 produces a new equation by dividing the left and right
    ++ hand sides of equations e1 and e2.
  inv: $ -> $
    ++ inv(x) returns the multiplicative inverse of x.
if S has ExpressionSpace then
  subst: ($, $) -> $
    ++ subst(eq1,eq2) substitutes eq2 into both sides of eq1
    ++ the lhs of eq2 should be a kernel

private ==> add
Rep := Record(lhs: S, rhs: S)
eq1,eq2: $
s : S
if S has IntegralDomain then
  factorAndSplit eq ==
    (S has factor : S -> Factored S) =>
      eq0 := rightZero eq
      [equation(rcf.factor,0) for rcf in factors factor lhs eq0]
  [eq]
  l:S = r:S      == [1, r]
  equation(l, r) == [1, r]    -- hack! See comment above.
  lhs eqn        == eqn.lhs
  rhs eqn        == eqn.rhs

```

```

swap eqn      == [rhs eqn, lhs eqn]
map(fn, eqn)  == equation(fn(eqn.lhs), fn(eqn.rhs))

if S has InnerEvalable(Symbol,S) then
  s:Symbol
  ls:List Symbol
  x:S
  lx:List S
  eval(eqn,s,x) == eval(eqn.lhs,s,x) = eval(eqn.rhs,s,x)
  eval(eqn,ls,lx) == eval(eqn.lhs,ls,lx) = eval(eqn.rhs,ls,lx)
if S has Evalable(S) then
  eval(eqn1:$, eqn2:$):$ ==
    eval(eqn1.lhs, eqn2 pretend Equation S) =
      eval(eqn1.rhs, eqn2 pretend Equation S)
  eval(eqn1:$, leqn2:List $):$ ==
    eval(eqn1.lhs, leqn2 pretend List Equation S) =
      eval(eqn1.rhs, leqn2 pretend List Equation S)
if S has SetCategory then
  eq1 = eq2 == (eq1.lhs = eq2.lhs)@Boolean and
               (eq1.rhs = eq2.rhs)@Boolean
  coerce(eqn:$):Ex == eqn.lhs::Ex = eqn.rhs::Ex
  coerce(eqn:$):Boolean == eqn.lhs = eqn.rhs
if S has AbelianSemiGroup then
  eq1 + eq2 == eq1.lhs + eq2.lhs = eq1.rhs + eq2.rhs
  s + eq2 == [s,s] + eq2
  eq1 + s == eq1 + [s,s]
if S has AbelianGroup then
  - eq == (- lhs eq) = (-rhs eq)
  s - eq2 == [s,s] - eq2
  eq1 - s == eq1 - [s,s]
  leftZero eq == 0 = rhs eq - lhs eq
  rightZero eq == lhs eq - rhs eq = 0
  0 == equation(0$S,0$S)
  eq1 - eq2 == eq1.lhs - eq2.lhs = eq1.rhs - eq2.rhs
if S has SemiGroup then
  eq1:$ * eq2:$ == eq1.lhs * eq2.lhs = eq1.rhs * eq2.rhs
  1:S * eqn:$ == 1 * eqn.lhs = 1 * eqn.rhs
  1:S * eqn:$ == 1 * eqn.lhs = 1 * eqn.rhs
  eqn:$ * 1:S == eqn.lhs * 1 = eqn.rhs * 1
  -- We have to be a bit careful here: raising to a +ve integer is OK
  -- (since it's the equivalent of repeated multiplication)
  -- but other powers may cause contradictions
  -- Watch what else you add here! JHD 2/Aug 1990
if S has Monoid then
  1 == equation(1$S,1$S)
  recip eq ==
    (lh := recip lhs eq) case "failed" => "failed"
    (rh := recip rhs eq) case "failed" => "failed"
    [lh :: S, rh :: S]
  leftOne eq ==

```

```

      (re := recip lhs eq) case "failed" => "failed"
      1 = rhs eq * re
    rightOne eq ==
      (re := recip rhs eq) case "failed" => "failed"
      lhs eq * re = 1
  if S has Group then
    inv eq == [inv lhs eq, inv rhs eq]
    leftOne eq == 1 = rhs eq * inv rhs eq
    rightOne eq == lhs eq * inv rhs eq = 1
  if S has Ring then
    characteristic() == characteristic()$S
    i:Integer * eq:$ == (i::S) * eq
  if S has IntegralDomain then
    factorAndSplit eq ==
      (S has factor : S -> Factored S) =>
        eq0 := rightZero eq
        [equation(rcf.factor,0) for rcf in factors factor lhs eq0]
      (S has Polynomial Integer) =>
        eq0 := rightZero eq
        MF ==> MultivariateFactorize(Symbol, IndexedExponents Symbol, _
          Integer, Polynomial Integer)
        p : Polynomial Integer := (lhs eq0) pretend Polynomial Integer
        [equation((rcf.factor) pretend S,0) for rcf in factors factor(p)$MF]
      [eq]
  if S has PartialDifferentialRing(Symbol) then
    differentiate(eq:$, sym:Symbol):$ ==
      [differentiate(lhs eq, sym), differentiate(rhs eq, sym)]
  if S has Field then
    dimension() == 2 :: CardinalNumber
    eq1:$ / eq2:$ == eq1.lhs / eq2.lhs = eq1.rhs / eq2.rhs
    inv eq == [inv lhs eq, inv rhs eq]
  if S has ExpressionSpace then
    subst(eq1,eq2) ==
      eq3 := eq2 pretend Equation S
      [subst(lhs eq1,eq3),subst(rhs eq1,eq3)]

```

3.2 preparse

The first large transformation of this input occurs in the function `preparse`. The `preparse` function reads the source file and breaks the input into a list of pairs. The first part of the pair is the line number of the input file and the second part of the pair is the actual source text as a string.

One feature that is the added semicolons at the end of the strings where the “pile” structure of the code has been converted to a semicolon delimited form.

3.2.1 defvar \$index

— initvars —

```
(defvar $index 0 "File line number of most recently read line")
```

—————

3.2.2 defvar \$linelist

— initvars —

```
(defvar $linelist nil "Stack of preparsed lines")
```

—————

3.2.3 defvar \$echolinestack

— initvars —

```
(defvar $echolinestack nil "Stack of lines to list")
```

—————

3.2.4 defvar \$preparse-last-line

— initvars —

```
(defvar $preparse-last-line nil "Most recently read line")
```

—————

3.3 Parsing routines

The **initialize-preparse** expects to be called before the **preparse** function. It initializes the state, in particular, it reads a single line from the input stream and stores it in

`$preparse-last-line`. The caller gives a stream and the `$preparse-last-line` variable is initialized as:

```
2> (INITIALIZE-PREPARSE #<input stream "/tmp/EQ.spad">)
<2 (INITIALIZE-PREPARSE ")abbrev domain EQ Equation")
```

3.3.1 defun initialize-preparse

```
[get-a-line p540]
[$index p72]
[$linelist p72]
[$echolinestack p72]
[$preparse-last-line p72]
```

— defun initialize-preparse —

```
(defun initialize-preparse (strm)
  (setq $index 0)
  (setq $linelist nil)
  (setq $echolinestack nil)
  (setq $preparse-last-line (get-a-line strm)))
```

The **preparse** function returns a list of pairs of the form: ((linenumber . linestring) (linenumber . linestring)) For instance, for the file `EQ.spad`, we get:

```
2> (PREPARSE #<input stream "/tmp/EQ.spad">)
3> (PREPARSE1 (")abbrev domain EQ Equation"))
4> (|doSystemCommand| "abbrev domain EQ Equation")
<4 (|doSystemCommand| NIL)
<3 (PREPARSE1 ( ...[snip]... )
<2 (PREPARSE (
(19 . "Equation(S: Type): public == private where")
(20 . " (Ex ==> OutputForm;")
(21 . " public ==> Type with")
(22 . " (\ "=": (S, S) -> $;")
(24 . " equation: (S, S) -> $;")
(26 . " swap: $ -> $;")
(28 . " lhs: $ -> S;")
(30 . " rhs: $ -> S;")
(32 . " map: (S -> S, $) -> $;")
(35 . " if S has InnerEvalable(Symbol,S) then")
(36 . " InnerEvalable(Symbol,S);")
(37 . " if S has SetCategory then")
(38 . " (SetCategory;")
(39 . " CoercibleTo Boolean;")
```



```

(40 . "      if S has Evaluable(S) then")
(41 . "      (eval: ($, $) -> $;")
(43 . "      eval: ($, List $) -> $));")
(45 . "  if S has AbelianSemiGroup then")
(46 . "    (AbelianSemiGroup;")
(47 . "    \"+\": (S, $) -> $;")
(50 . "    \"+\": ($, S) -> $;")
(53 . "  if S has AbelianGroup then")
(54 . "    (AbelianGroup;")
(55 . "    leftZero : $ -> $;")
(57 . "    rightZero : $ -> $;")
(59 . "    \"-\": (S, $) -> $;")
(62 . "    \"-\": ($, S) -> $;")
(65 . "  if S has SemiGroup then")
(66 . "    (SemiGroup;")
(67 . "    \": (S, $) -> $;")
(70 . "    \": ($, S) -> $;")
(73 . "  if S has Monoid then")
(74 . "    (Monoid;")
(75 . "    leftOne : $ -> Union($,\"failed\");")
(77 . "    rightOne : $ -> Union($,\"failed\");")
(79 . "  if S has Group then")
(80 . "    (Group;")
(81 . "    leftOne : $ -> Union($,\"failed\");")
(83 . "    rightOne : $ -> Union($,\"failed\");")
(85 . "  if S has Ring then")
(86 . "    (Ring;")
(87 . "    BiModule(S,S));")
(88 . "  if S has CommutativeRing then")
(89 . "    Module(S;")
(91 . "  if S has IntegralDomain then")
(92 . "    factorAndSplit : $ -> List $;")
(96 . "  if S has PartialDifferentialRing(Symbol) then")
(97 . "    PartialDifferentialRing(Symbol);")
(98 . "  if S has Field then")
(99 . "    (VectorSpace(S;")
(100 . "    \"/\": ($, $) -> $;")
(103 . "    inv: $ -> $;")
(105 . "  if S has ExpressionSpace then")
(106 . "    subst: ($, $) -> $;")
(109 . "  private ==> add")
(110 . "    (Rep := Record(lhs: S, rhs: S);")
(111 . "    eq1,eq2: $;")
(112 . "    s : S;")
(113 . "    if S has IntegralDomain then")
(114 . "      factorAndSplit eq ==")
(115 . "        ((S has factor : S -> Factored S) =>")
(116 . "          (eq0 := rightZero eq;")
(117 . "            [equation(rcf.factor,0)
              for rcf in factors factor lhs eq0]));")

```

```

(118 . "      [eq]);")
(119 . "    l:S = r:S      == [l, r];")
(120 . "    equation(l, r) == [l, r];")
(121 . "    lhs eqn          == eqn.lhs;")
(122 . "    rhs eqn          == eqn.rhs;")
(123 . "    swap eqn         == [rhs eqn, lhs eqn];")
(124 . "    map(fn, eqn)      == equation(fn(eqn.lhs), fn(eqn.rhs));")
(125 . "    if S has InnerEvalable(Symbol,S) then")
(126 . "      (s:Symbol;")
(127 . "      ls:List Symbol;")
(128 . "      x:S;")
(129 . "      lx:List S;")
(130 . "      eval(eqn,s,x) == eval(eqn.lhs,s,x) = eval(eqn.rhs,s,x);")
(131 . "      eval(eqn,ls,lx) == eval(eqn.lhs,ls,lx) =
                                eval(eqn.rhs,ls,lx));")
(132 . "    if S has Evalable(S) then")
(133 . "      (eval(eqn1:$, eqn2:$):$ ==")
(134 . "        eval(eqn1.lhs, eqn2 pretend Equation S) ==")
(135 . "          eval(eqn1.rhs, eqn2 pretend Equation S);")
(136 . "      eval(eqn1:$, leqn2:List $):$ ==")
(137 . "        eval(eqn1.lhs, leqn2 pretend List Equation S) ==")
(138 . "          eval(eqn1.rhs, leqn2 pretend List Equation S));")
(139 . "    if S has SetCategory then")
(140 . "      (eq1 = eq2 == (eq1.lhs = eq2.lhs)@Boolean and")
(141 . "        (eq1.rhs = eq2.rhs)@Boolean;")
(142 . "      coerce(eqn:$):Ex == eqn.lhs::Ex = eqn.rhs::Ex;")
(143 . "      coerce(eqn:$):Boolean == eqn.lhs = eqn.rhs;")
(144 . "    if S has AbelianSemiGroup then")
(145 . "      (eq1 + eq2 == eq1.lhs + eq2.lhs = eq1.rhs + eq2.rhs;")
(146 . "      s + eq2 == [s,s] + eq2;")
(147 . "      eq1 + s == eq1 + [s,s]);")
(148 . "    if S has AbelianGroup then")
(149 . "      (- eq == (- lhs eq) = (-rhs eq);")
(150 . "      s - eq2 == [s,s] - eq2;")
(151 . "      eq1 - s == eq1 - [s,s];")
(152 . "      leftZero eq == 0 = rhs eq - lhs eq;")
(153 . "      rightZero eq == lhs eq - rhs eq = 0;")
(154 . "      0 == equation(0$S,0$S);")
(155 . "      eq1 - eq2 == eq1.lhs - eq2.lhs = eq1.rhs - eq2.rhs;")
(156 . "    if S has SemiGroup then")
(157 . "      (eq1:$ * eq2:$ == eq1.lhs * eq2.lhs = eq1.rhs * eq2.rhs;")
(158 . "      l:S * eqn:$ == l * eqn.lhs = l * eqn.rhs;")
(159 . "      l:S * eqn:$ == l * eqn.lhs = l * eqn.rhs;")
(160 . "      eqn:$ * l:S == eqn.lhs * l = eqn.rhs * l;")
(165 . "    if S has Monoid then")
(166 . "      (1 == equation(1$S,1$S);")
(167 . "      recip eq ==")
(168 . "        ((lh := recip lhs eq) case \"failed\" => \"failed\");")
(169 . "        (rh := recip rhs eq) case \"failed\" => \"failed\");")
(170 . "      [lh :: S, rh :: S]);")

```

```

(171 . "      leftOne eq ==")
(172 . "      ((re := recip lhs eq) case \"failed\" => \"failed\");")
(173 . "      1 = rhs eq * re);")
(174 . "      rightOne eq ==")
(175 . "      ((re := recip rhs eq) case \"failed\" => \"failed\");")
(176 . "      lhs eq * re = 1));")
(177 . "  if S has Group then")
(178 . "    (inv eq == [inv lhs eq, inv rhs eq];")
(179 . "    leftOne eq == 1 = rhs eq * inv rhs eq;")
(180 . "    rightOne eq == lhs eq * inv rhs eq = 1);")
(181 . "  if S has Ring then")
(182 . "    (characteristic() == characteristic()$S;")
(183 . "    i:Integer * eq:$ == (i::S) * eq);")
(184 . "  if S has IntegralDomain then")
(185 . "    factorAndSplit eq ==")
(186 . "    ((S has factor : S -> Factored S) =>")
(187 . "    (eq0 := rightZero eq;")
(188 . "    [equation(rcf.factor,0)
      for rcf in factors factor lhs eq0]);")
(189 . "    (S has Polynomial Integer) =>")
(190 . "    (eq0 := rightZero eq;")
(191 . "    MF ==> MultivariateFactorize(Symbol,
      IndexedExponents Symbol,
      Integer, Polynomial Integer);")
(193 . "    p : Polynomial Integer :=
      (lhs eq0) pretend Polynomial Integer;")
(194 . "    [equation((rcf.factor) pretend S,0)
      for rcf in factors factor(p)$MF]);")
(195 . "    [eq]);")
(196 . "  if S has PartialDifferentialRing(Symbol) then")
(197 . "    differentiate(eq:$, sym:Symbol):$ ==")
(198 . "    [differentiate(lhs eq, sym), differentiate(rhs eq, sym)];")
(199 . "  if S has Field then")
(200 . "    (dimension() == 2 :: CardinalNumber;")
(201 . "    eq1:$ / eq2:$ == eq1.lhs / eq2.lhs = eq1.rhs / eq2.rhs;")
(202 . "    inv eq == [inv lhs eq, inv rhs eq]);")
(203 . "  if S has ExpressionSpace then")
(204 . "    subst(eq1,eq2) ==")
(205 . "    (eq3 := eq2 pretend Equation S;")
(206 . "    [subst(lhs eq1,eq3),subst(rhs eq1,eq3)])))")

```

3.3.2 defun preparse

```

[preparse p76]
[preparse1 p81]
[parseprint p468]
[ifcar p??]
[$comblocklist p465]

```

```

[ $skipme p?? ]
[ $preparse-last-line p72 ]
[ $index p72 ]
[ $docList p?? ]
[ $preparseReportIfTrue p?? ]
[ $headerDocumentation p?? ]
[ $maxSignatureLineNumber p?? ]
[ $constructorLineNumber p?? ]

```

— **defun preparse** —

```

(defun preparse (strm &aux (stack ()))
  (declare (special $comblocklist $skipme $preparse-last-line $index |$docList|
                    $preparseReportIfTrue |$headerDocumentation|
                    |$maxSignatureLineNumber| |$constructorLineNumber|))
  (setq $comblocklist nil)
  (setq $skipme nil)
  (when $preparse-last-line
    (if (consp $preparse-last-line)
        (setq stack $preparse-last-line)
        (push $preparse-last-line stack))
    (setq $index (- $index (length stack))))
  (let ((u (preparse1 stack)))
    (if $skipme
        (preparse strm)
        (progn
          (when $preparseReportIfTrue (parseprint u))
          (setq |$headerDocumentation| nil)
          (setq |$docList| nil)
          (setq |$maxSignatureLineNumber| 0)
          (setq |$constructorLineNumber| (ifcar (ifcar u)))
          u))))

```

The **preparse** function returns a list of pairs of the form: ((linenumber . linestring) (linenumber . linestring)) For instance, for the file `EQ.spad`, we get:

```

2> (PREPARSE #<input stream "/tmp/EQ.spad">)
3> (PREPARSE1 (")abbrev domain EQ Equation"))
4> (|doSystemCommand| "abbrev domain EQ Equation")
<4 (|doSystemCommand| NIL)
<3 (PREPARSE1 (
(19 . "Equation(S: Type): public == private where")
(20 . " (Ex ==> OutputForm;")
(21 . "   public ==> Type with")
(22 . "   (\ "=": (S, S) -> $;")
(24 . "   equation: (S, S) -> $;")

```

```

(26 . "      swap: $ -> $;")
(28 . "      lhs: $ -> S;")
(30 . "      rhs: $ -> S;")
(32 . "      map: (S -> S, $) -> $;")
(35 . "      if S has InnerEvalable(Symbol,S) then")
(36 . "          InnerEvalable(Symbol,S);")
(37 . "      if S has SetCategory then")
(38 . "          (SetCategory;")
(39 . "              CoercibleTo Boolean;")
(40 . "              if S has Evalable(S) then")
(41 . "                  (eval: ($, $) -> $;")
(43 . "                      eval: ($, List $) -> $);")
(45 . "      if S has AbelianSemiGroup then")
(46 . "          (AbelianSemiGroup;")
(47 . "              \"+\": (S, $) -> $;")
(50 . "              \"+\": ($, S) -> $);")
(53 . "      if S has AbelianGroup then")
(54 . "          (AbelianGroup;")
(55 . "              leftZero : $ -> $;")
(57 . "              rightZero : $ -> $;")
(59 . "              \"-\": (S, $) -> $;")
(62 . "              \"-\": ($, S) -> $);")
(65 . "      if S has SemiGroup then")
(66 . "          (SemiGroup;")
(67 . "              \"*\": (S, $) -> $;")
(70 . "              \"*\": ($, S) -> $);")
(73 . "      if S has Monoid then")
(74 . "          (Monoid;")
(75 . "              leftOne : $ -> Union($,\"failed\");")
(77 . "              rightOne : $ -> Union($,\"failed\");")
(79 . "      if S has Group then")
(80 . "          (Group;")
(81 . "              leftOne : $ -> Union($,\"failed\");")
(83 . "              rightOne : $ -> Union($,\"failed\");")
(85 . "      if S has Ring then")
(86 . "          (Ring;")
(87 . "              BiModule(S,S);")
(88 . "      if S has CommutativeRing then")
(89 . "          Module(S);")
(91 . "      if S has IntegralDomain then")
(92 . "          factorAndSplit : $ -> List $;")
(96 . "      if S has PartialDifferentialRing(Symbol) then")
(97 . "          PartialDifferentialRing(Symbol);")
(98 . "      if S has Field then")
(99 . "          (VectorSpace(S);")
(100 . "              \"/\": ($, $) -> $;")
(103 . "              inv: $ -> $);")
(105 . "      if S has ExpressionSpace then")
(106 . "          subst: ($, $) -> $;")
(109 . "      private ==> add")

```

```

(110 . " (Rep := Record(lhs: S, rhs: S);")
(111 . "   eq1,eq2: $;")
(112 . "   s : S;")
(113 . "   if S has IntegralDomain then")
(114 . "       factorAndSplit eq ==")
(115 . "           ((S has factor : S -> Factored S) =>)")
(116 . "           (eq0 := rightZero eq;")
(117 . "               [equation(rcf.factor,0)
                   for rcf in factors factor lhs eq0]));")
(118 . "       [eq]);")
(119 . "   l:S = r:S      == [l, r];")
(120 . "   equation(l, r) == [l, r];")
(121 . "   lhs eqn        == eqn.lhs;")
(122 . "   rhs eqn        == eqn.rhs;")
(123 . "   swap eqn       == [rhs eqn, lhs eqn];")
(124 . "   map(fn, eqn)    == equation(fn(eqn.lhs), fn(eqn.rhs));")
(125 . "   if S has InnerEvalable(Symbol,S) then")
(126 . "       (s:Symbol;")
(127 . "       ls:List Symbol;")
(128 . "       x:S;")
(129 . "       lx:List S;")
(130 . "       eval(eqn,s,x) == eval(eqn.lhs,s,x) = eval(eqn.rhs,s,x);")
(131 . "       eval(eqn,ls,lx) == eval(eqn.lhs,ls,lx) =
                               eval(eqn.rhs,ls,lx));")
(132 . "   if S has Evalable(S) then")
(133 . "       (eval(eqn1:$, eqn2:$):$ ==")
(134 . "           eval(eqn1.lhs, eqn2 pretend Equation S) ==")
(135 . "           eval(eqn1.rhs, eqn2 pretend Equation S));")
(136 . "       eval(eqn1:$, leqn2:List $):$ ==")
(137 . "           eval(eqn1.lhs, leqn2 pretend List Equation S) ==")
(138 . "           eval(eqn1.rhs, leqn2 pretend List Equation S));")
(139 . "   if S has SetCategory then")
(140 . "       (eq1 = eq2 == (eq1.lhs = eq2.lhs)@Boolean and")
(141 . "           (eq1.rhs = eq2.rhs)@Boolean;")
(142 . "       coerce(eqn:$):Ex == eqn.lhs::Ex = eqn.rhs::Ex;")
(143 . "       coerce(eqn:$):Boolean == eqn.lhs = eqn.rhs;")
(144 . "   if S has AbelianSemiGroup then")
(145 . "       (eq1 + eq2 == eq1.lhs + eq2.lhs = eq1.rhs + eq2.rhs;")
(146 . "       s + eq2 == [s,s] + eq2;")
(147 . "       eq1 + s == eq1 + [s,s]);")
(148 . "   if S has AbelianGroup then")
(149 . "       (- eq == (- lhs eq) = (-rhs eq);")
(150 . "       s - eq2 == [s,s] - eq2;")
(151 . "       eq1 - s == eq1 - [s,s];")
(152 . "       leftZero eq == 0 = rhs eq - lhs eq;")
(153 . "       rightZero eq == lhs eq - rhs eq = 0;")
(154 . "       0 == equation(0$S,0$S);")
(155 . "       eq1 - eq2 == eq1.lhs - eq2.lhs = eq1.rhs - eq2.rhs;")
(156 . "   if S has SemiGroup then")
(157 . "       (eq1:$ * eq2:$ == eq1.lhs * eq2.lhs = eq1.rhs * eq2.rhs;")

```

```

(158 . "      l:S * eqn:$ == l      * eqn.lhs = l      * eqn.rhs;")
(159 . "      l:S * eqn:$ == l * eqn.lhs    =    l * eqn.rhs;")
(160 . "      eqn:$ * l:S == eqn.lhs * l    =    eqn.rhs * l;")
(165 . "  if S has Monoid then")
(166 . "    (1 == equation(1$S,1$S);")
(167 . "    recip eq ==")
(168 . "      ((lh := recip lhs eq) case \"failed\" => \"failed\");")
(169 . "      (rh := recip rhs eq) case \"failed\" => \"failed\");")
(170 . "      [lh :: S, rh :: S]);")
(171 . "    leftOne eq ==")
(172 . "      ((re := recip lhs eq) case \"failed\" => \"failed\");")
(173 . "      1 = rhs eq * re;")
(174 . "    rightOne eq ==")
(175 . "      ((re := recip rhs eq) case \"failed\" => \"failed\");")
(176 . "      lhs eq * re = 1));")
(177 . "  if S has Group then")
(178 . "    (inv eq == [inv lhs eq, inv rhs eq];")
(179 . "    leftOne eq == 1 = rhs eq * inv rhs eq;")
(180 . "    rightOne eq == lhs eq * inv rhs eq = 1);")
(181 . "  if S has Ring then")
(182 . "    (characteristic() == characteristic()$S;")
(183 . "    i:Integer * eq:$ == (i::S) * eq;")
(184 . "  if S has IntegralDomain then")
(185 . "    factorAndSplit eq ==")
(186 . "      ((S has factor : S -> Factored S) =>")
(187 . "        (eq0 := rightZero eq;")
(188 . "          [equation(rcf.factor,0)
            for rcf in factors factor lhs eq0]));")
(189 . "      (S has Polynomial Integer) =>")
(190 . "        (eq0 := rightZero eq;")
(191 . "          MF ==> MultivariateFactorize(Symbol,
            IndexedExponents Symbol,
            Integer, Polynomial Integer);")
(193 . "          p : Polynomial Integer :=
            (lhs eq0) pretend Polynomial Integer;")
(194 . "          [equation((rcf.factor) pretend S,0)
            for rcf in factors factor(p)$MF]);")
(195 . "          [eq]);")
(196 . "  if S has PartialDifferentialRing(Symbol) then")
(197 . "    differentiate(eq:$, sym:Symbol):$ ==")
(198 . "      [differentiate(lhs eq, sym), differentiate(rhs eq, sym)];")
(199 . "  if S has Field then")
(200 . "    (dimension() == 2 :: CardinalNumber;")
(201 . "    eq1:$ / eq2:$ == eq1.lhs / eq2.lhs = eq1.rhs / eq2.rhs;")
(202 . "    inv eq == [inv lhs eq, inv rhs eq]);")
(203 . "  if S has ExpressionSpace then")
(204 . "    subst(eq1,eq2) ==")
(205 . "      (eq3 := eq2 pretend Equation S;")
(206 . "        [subst(lhs eq1,eq3),subst(rhs eq1,eq3)])))))

```

3.3.3 defun Build the lines from the input for piles

The READLOOP calls `preparseReadLine` which returns a pair of the form

```
(number . string)
```

```
[preparseReadLine p85]
[preparse-echo p88]
[fincomblock p466]
[parsepiles p84]
[preparse1 doSystemCommand (vol5)]
[escaped p465]
[indent-pos p466]
[make-full-cvec p??]
[maxindex p??]
[preparse1 strposl (vol5)]
[is-console p467]
[spad-reader p??]
[$echolinestack p72]
[$byConstructors p531]
[$skipme p??]
[$constructorsSeen p531]
[$preparse-last-line p72]
[$preparse-last-line p72]
[$index p72]
[$index p72]
[$linelist p72]
[$in-stream p??]
```

— defun `preparse1` —

```
(defun preparse1 (linelist)
  (labels (
    (isSystemCommand (line)
      (and (> (length line) 0) (eq (char line 0) #\ ) )))
    (executeSystemCommand (line)
      (catch 'spad_reader (|doSystemCommand| (subseq line 1))))
  )
  (prog (($linelist linelist) $echolinestack num line i l psloc
    instring pcount comsym strsym oparsym cparsym n ncomsym tmp1
    (sloc -1) continue (parenlev 0) ncomblock lines locs nums functor)
    (declare (special $linelist $echolinestack |$byConstructors| $skipme
      |$constructorsSeen| $preparse-last-line $index in-stream))
  READLOOP
    (setq tmp1 (preparseReadLine linelist))
    (setq num (car tmp1))
    (setq line (cdr tmp1))
    (unless (stringp line)
```



```

(preparse-echo linelist)
(cond
  ((null lines) (return nil))
  (ncomblock (fincomblock nil nums locs ncomblock nil)))
(return
  (pair (nreverse nums) (parsepiles (nreverse locs) (nreverse lines))))))
(when (and (null lines) (isSystemCommand line))
  (preparse-echo linelist)
  (setq $preparse-last-line nil) ;don't reread this line
  (executeSystemCommand line)
  (go READLOOP))
(setq l (length line))
; if we get a null line, read the next line
(when (eq l 0) (go READLOOP))
; otherwise we have to parse this line
(setq psloc sloc)
(setq i 0)
(setq instring nil)
(setq pcount 0)
STRLOOP ;; handle things that need ignoring, quoting, or grouping
; are we in a comment, quoting, or grouping situation?
(setq strsym (or (position #" line :start i ) 1))
(setq comsym (or (search "--" line :start2 i ) 1))
(setq ncomsym (or (search "++" line :start2 i ) 1))
(setq oparsym (or (position #\ ( line :start i ) 1))
(setq cparsym (or (position #\ ) line :start i ) 1))
(setq n (min strsym comsym ncomsym oparsym cparsym))
(cond
  ; nope, we found no comment, quoting, or grouping
  ((= n 1) (go NOCOMS))
  ((escaped line n))
  ; scan until we hit the end of the string
  ((= n strsym) (setq instring (not instring)))
  ; we are in a string, just continue looping
  (instring)
  ;; handle -- comments by ignoring them
  ((= n comsym)
   (setq line (subseq line 0 n))
   (go NOCOMS)) ; discard trailing comment
  ;; handle ++ comments by chunking them together
  ((= n ncomsym)
   (setq sloc (indent-pos line))
   (cond
     ((= sloc n)
      (when (and ncomblock (not (= n (car ncomblock))))
        (fincomblock num nums locs ncomblock linelist)
        (setq ncomblock nil))
      (setq ncomblock (cons n (cons line (ifcdr ncomblock))))
      (setq line ""))
     (t

```

```

    (push (strconc (make-full-cvec n " ") (substring line n ())) $linelist)
    (setq $index (1- $index))
    (setq line (subseq line 0 n))))
  (go NOCOMS))
; know how deep we are into parens
((= n oparsym) (setq pcount (1+ pcount)))
((= n cparsym) (setq pcount (1- pcount)))
(setq i (1+ n))
(go STRLOOP)
NOCOMS
; remember the indentation level
(setq sloc (indent-pos line))
(setq line (string-right-trim " " line))
(when (null sloc)
  (setq sloc psloc)
  (go READLOOP))
; handle line that ends in a continuation character
(cond
  ((eq (elt line (maxindex line)) #\_)
   (setq continue t)
   (setq line (subseq line (maxindex line))))
  ((setq continue nil)))
; test for skipping constructors
(when (and (null lines) (= sloc 0))
  (if (and |$byConstructors|
        (null (search "==" line))
        (not
         (member
          (setq functor
                (intern (substring line 0 (strpos1 ": (" line 0 nil))))
          |$byConstructors|)))
      (setq $skipme 't)
      (progn
        (push functor |$constructorsSeen|)
        (setq $skipme nil))))))
; is this thing followed by ++ comments?
(when (and lines (eql sloc 0))
  (when (and ncomblock (not (zerop (car ncomblock))))
    (fincomblock num nums locs ncomblock linelist))
  (when (not (is-console in-stream))
    (setq $preparse-last-line (nreverse $echolinestack)))
  (return
   (pair (nreverse nums) (parsepiles (nreverse locs) (nreverse lines)))))
(when (> parenlev 0)
  (push nil locs)
  (setq sloc psloc)
  (go REREAD))
(when ncomblock
  (fincomblock num nums locs ncomblock linelist)
  (setq ncomblock ()))

```

```

(push sloc locs)
REREAD
  (preparse-echo linelist)
  (push line lines)
  (push num nums)
  (setq parenlev (+ parenlev pcount))
  (when (and (is-console in-stream) (not continue))
    (setq $preparse-last-line nil)
    (return
      (pair (nreverse nums) (parsepiles (nreverse locs) (nreverse lines))))))
  (go READLOOP)))

```

3.3.4 defun parsepiles

Add parens and semis to lines to aid parsing. [add-parens-and-semis-to-line p84]

— defun parsepiles —

```

(defun parsepiles (locs lines)
  (mapl #'add-parens-and-semis-to-line
    (nconc lines '(" ")) (nconc locs '(nil)))
  lines)

```

3.3.5 defun add-parens-and-semis-to-line

The line to be worked on is (CAR SLINES). It's indentation is (CAR SLOCS). There is a notion of current indentation. Then:

- Add open paren to beginning of following line if following line's indentation is greater than current, and add close paren to end of last succeeding line with following line's indentation.
- Add semicolon to end of line if following line's indentation is the same.
- If the entire line consists of the single keyword then or else, leave it alone."

```

[infxtok p467]
[drop p465]
[addclose p464]
[nonblankloc p468]

```

— defun add-parens-and-semis-to-line —

```

(defun add-parens-and-semis-to-line (slines slocs)
  (let ((start-column (car slocs)))
    (when (and start-column (> start-column 0))
      (let ((count 0) (i 0))
        (seq
         (mapl #'(lambda (next-lines nlocs)
                   (let ((next-line (car next-lines)) (next-column (car nlocs)))
                     (incf i)
                     (when next-column
                      (setq next-column (abs next-column))
                      (when (< next-column start-column) (exit nil))
                      (cond
                       ((and (eq next-column start-column)
                            (rplaca nlocs (- (car nlocs)))
                            (not (infixtok next-line)))
                        (setq next-lines (drop (1- i) slines))
                        (rplaca next-lines (addclose (car next-lines) #\;))
                        (setq count (1+ count)))))))
                    (cdr slines) (cdr slocs)))
         (when (> count 0)
          (setf (char (car slines) (1- (nonblankloc (car slines)))) #\ ( )
                (setf slines (drop (1- i) slines))
                (rplaca slines (addclose (car slines) #\ ) ))))))))

```

3.3.6 defun preparseReadLine

```

[dcq p??]
[preparseReadLine1 p87]
[initial-substring p539]
[string2BootTree p??]
[storeblanks p539]
[skip-to-endif p468]
[preparseReadLine p85]
[$*eof* p??]

```

— defun preparseReadLine —

```

(defun preparseReadLine (x)
  (let (line ind tmp1)
    (declare (special *eof*))
    (setq tmp1 (preparseReadLine1))
    (setq ind (car tmp1))
    (setq line (cdr tmp1))
    (cond
     ((not (stringp line)) (cons ind line))

```

```

((zerop (size line)) (cons ind line))
((char= (elt line 0) #\ )
 (cond
  ((initial-substring ")if" line)
   (if (eval (|string2BootTree| (storeblanks line 3)))
       (preparseReadLine x)
       (skip-ifblock x)))
  ((initial-substring ")elseif" line) (skip-to-endif x))
  ((initial-substring ")else" line) (skip-to-endif x))
  ((initial-substring ")endif" line) (preparseReadLine x))
  ((initial-substring ")fin" line)
   (setq *eof* t)
   (cons ind nil))))
(cons ind line))

```

3.3.7 defun skip-ifblock

```

[preparseReadLine1 p87]
[skip-ifblock p86]
[initial-substring p539]
[string2BootTree p??]
[storeblanks p539]

```

— defun skip-ifblock —

```

(defun skip-ifblock (x)
  (let (line ind tmp1)
    (setq tmp1 (preparseReadLine1))
    (setq ind (car tmp1))
    (setq line (cdr tmp1))
    (cond
     ((not (stringp line))
      (cons ind line))
     ((zerop (size line))
      (skip-ifblock x))
     ((char= (elt line 0) #\ )
      (cond
       ((initial-substring ")if" line)
        (cond
         ((eval (|string2BootTree| (storeblanks line 3)))
          (preparseReadLine X))
         (t (skip-ifblock x))))
       ((initial-substring ")elseif" line)
        (cond
         ((eval (|string2BootTree| (storeblanks line 7)))

```

```

      (preparseReadLine X))
    (t (skip-ifblock x))))
  ((initial-substring ")else" line)
    (preparseReadLine x))
  ((initial-substring ")endif" line)
    (preparseReadLine x))
  ((initial-substring ")fin" line)
    (cons ind nil))))
  (t (skip-ifblock x))))

```

3.3.8 defun preparseReadLine1

```

[get-a-line p540]
[expand-tabs p??]
[maxindex p??]
[strconc p??]
[preparseReadLine1 p87]
[$linelist p72]
[$preparse-last-line p72]
[$index p72]
[$EchoLineStack p??]

```

— defun preparseReadLine1 —

```

(defun preparseReadLine1 ()
  (labels (
    (accumulateLinesWithTrailingEscape (line)
      (let (ind)
        (declare (special $preparse-last-line))
        (if (and (> (setq ind (maxindex line)) -1) (char= (elt line ind) #\_))
            (setq $preparse-last-line
              (strconc (substring line 0 ind) (cdr (preparseReadLine1))))
          line))))
    (let (line)
      (declare (special $linelist $preparse-last-line $index $EchoLineStack))
      (setq line
        (if $linelist
            (pop $linelist)
            (expand-tabs (get-a-line in-stream))))
      (setq $preparse-last-line line)
      (if (stringp line)
          (progn
            (incf $index) ;; $index is the current line number
            (setq line (string-right-trim " " line))
            (push (copy-seq line) $EchoLineStack)

```

```
(cons $index (accumulateLinesWithTrailingEscape line)))
(cons $index line))))
```

3.4 I/O Handling

3.4.1 defun preparse-echo

```
[Echo-Meta p??]
[$EchoLineStack p??]
```

— defun preparse-echo —

```
(defun preparse-echo (linelist)
  (declare (special $EchoLineStack Echo-Meta) (ignore linelist))
  (if Echo-Meta
    (dolist (x (reverse $EchoLineStack))
      (format out-stream "~&;~A~%" x)))
  (setq $EchoLineStack ()))
```

3.4.2 Parsing stack

3.4.3 defstruct \$stack

— initvars —

```
(defstruct stack "A stack"
  (store nil) ; contents of the stack
  (size 0) ; number of elements in Store
  (top nil) ; first element of Store
  (updated nil) ; whether something has been pushed on the stack
  ; since this flag was last set to NIL
)
```

3.4.4 defun stack-load

[\$stack p88]

— defun stack-load —

```
(defun stack-load (list stack)
  (setf (stack-store stack) list)
  (setf (stack-size stack) (length list))
  (setf (stack-top stack) (car list)))
```

—————

3.4.5 defun stack-clear

[\$stack p88]

— defun stack-clear —

```
(defun stack-clear (stack)
  (setf (stack-store stack) nil)
  (setf (stack-size stack) 0)
  (setf (stack-top stack) nil)
  (setf (stack-updated stack) nil))
```

—————

3.4.6 defmacro stack-/empty

[\$stack p88]

— defmacro stack-/empty —

```
(defmacro stack-/empty (stack) '(> (stack-size ,stack) 0))
```

—————

3.4.7 defun stack-push

[\$stack p88]

— defun stack-push —


```
(defun stack-push (x stack)
  (push x (stack-store stack))
  (setf (stack-top stack) x)
  (setf (stack-updated stack) t)
  (incf (stack-size stack))
  x)
```

3.4.8 defun stack-pop

[*\$stack* p88]

— **defun stack-pop** —

```
(defun stack-pop (stack)
  (let ((y (pop (stack-store stack))))
    (decf (stack-size stack))
    (setf (stack-top stack)
          (if (stack-/empty stack) (car (stack-store stack))
              y)))
```

3.4.9 Parsing token

3.4.10 defstruct \$token

A token is a Symbol with a Type. The type is either NUMBER, IDENTIFIER or SPECIAL-CHAR. NonBlank is true if the token is not preceded by a blank.

— **initvars** —

```
(defstruct token
  (symbol nil)
  (type nil)
  (nonblank t))
```

3.4.11 defvar \$prior-token

[*\$token* p90]

— **initvars** —

```
(defvar prior-token (make-token) "What did I see last")
```

3.4.12 defvar \$nonblank

— initvars —

```
(defvar nonblank t "Is there no blank in front of the current token.")
```

3.4.13 defvar \$current-token

Token at head of input stream. [\$token p90]

— initvars —

```
(defvar current-token (make-token))
```

3.4.14 defvar \$next-token

[\$token p90]

— initvars —

```
(defvar next-token (make-token) "Next token in input stream.")
```

3.4.15 defvar \$valid-tokens

[\$token p90]

— initvars —

```
(defvar valid-tokens 0 "Number of tokens in buffer (0, 1 or 2)")
```

3.4.16 defun token-install

[\$token p90]

— defun token-install —

```
(defun token-install (symbol type token &optional (nonblank t))
  (setf (token-symbol token) symbol)
  (setf (token-type token) type)
  (setf (token-nonblank token) nonblank)
  token)
```

3.4.17 defun token-print

[\$token p90]

— defun token-print —

```
(defun token-print (token)
  (format out-stream "(token (symbol ~S) (type ~S))~%"
    (token-symbol token) (token-type token)))
```

3.4.18 Parsing reduction**3.4.19 defstruct \$reduction**

A reduction of a rule is any S-Expression the rule chooses to stack.

— initvars —

```
(defstruct (reduction (:type list))
  (rule nil) ; Name of rule
  (value nil))
```

Chapter 4

Parse Transformers

4.1 Direct called parse routines

4.1.1 defun parseTransform

```
[msubst p??]  
[parseTran p93]  
[$defOp p??]
```

— defun parseTransform —

```
(defun |parseTransform| (x)  
  (let (|$defOp|)  
    (declare (special |$defOp|))  
    (setq |$defOp| nil)  
    (setq x (msubst '$ '% x)) ; for new compiler compatibility  
    (|parseTran| x)))
```

—————

4.1.2 defun parseTran

```
[parseAtom p94]  
[parseConstruct p95]  
[parseTran p93]  
[parseTranList p95]  
[getl p??]  
[$op p??]
```

— defun parseTran —

```

(defun |parseTran| (x)
  (labels (
    (g (op)
      (let (tmp1 tmp2 x)
        (seq
          (if (and (consp op) (eq (qfirst op) '|elt|))
              (progn
                (setq tmp1 (qrest op))
                (and (consp tmp1)
                    (progn
                     (setq op (qfirst tmp1))
                     (setq tmp2 (qrest tmp1))
                     (and (consp tmp2)
                         (eq (qrest tmp2) nil)
                         (progn (setq x (qfirst tmp2)) t))))))
              (exit (g x)))
          (exit op))))))
  (let (|$op| arg1 u r fn)
    (declare (special |$op|))
    (setq |$op| nil)
    (if (atom x)
        (|parseAtom| x)
        (progn
          (setq |$op| (car x))
          (setq arg1 (cdr x))
          (setq u (g |$op|))
          (cond
            ((eq u '|construct|)
             (setq r (|parseConstruct| arg1))
             (if (and (consp |$op|) (eq (qfirst |$op|) '|elt|))
                 (cons (|parseTran| |$op|) (cdr r))
                 r))
            ((and (atom u) (setq fn (get1 u '|parseTran|)))
             (funcall fn arg1))
            (t (cons (|parseTran| |$op|) (|parseTranList| arg1)))))))

```

4.1.3 defun parseAtom

```

[parseLeave p121]
[$NoValue p??]

```

— defun parseAtom —

```

(defun |parseAtom| (x)
  (declare (special |$NoValue|))

```

```
(if (eq x '|break|)
    (|parseLeave| (list '$NoValue|))
    x))
```

4.1.4 defun parseTranList

```
[parseTran p93]
[parseTranList p95]
```

— defun parseTranList —

```
(defun |parseTranList| (x)
  (if (atom x)
      (|parseTran| x)
      (cons (|parseTran| (car x)) (|parseTranList| (cdr x)))))
```

4.1.5 defplist parseConstruct

— postvars —

```
(eval-when (eval load)
  (setf (get '|construct| '|parseTran|) '|parseConstruct|))
```

4.1.6 defun parseConstruct

```
[parseTranList p95]
[$insideConstructIfTrue p??]
```

— defun parseConstruct —

```
(defun |parseConstruct| (u)
  (let (|$insideConstructIfTrue| x)
    (declare (special |$insideConstructIfTrue|))
    (setq |$insideConstructIfTrue| t)
    (setq x (|parseTranList| u))
    (cons '|construct| x)))
```

4.2 Indirect called parse routines

In the `parseTran` function there is the code:

```
((and (atom u) (setq fn (get1 u '|parseTran|)))
  (funcall fn arg1))
```

The functions in this section are called through the symbol-plist of the symbol being parsed. The original list read:

<code>and</code>	<code>parseAnd</code>
<code>@</code>	<code>parseAtSign</code>
<code>CATEGORY</code>	<code>parseCategory</code>
<code>::</code>	<code>parseCoerce</code>
<code>\:</code>	<code>parseColon</code>
<code>construct</code>	<code>parseConstruct</code>
<code>DEF</code>	<code>parseDEF</code>
<code>\$<=</code>	<code>parseDollarLessEqual</code>
<code>\$></code>	<code>parseDollarGreaterThan</code>
<code>\$>=</code>	<code>parseDollarGreaterEqual</code>
<code>\$^=</code>	<code>parseDollarNotEqual</code>
<code>eqv</code>	<code>parseEquivalence</code>
<code>exit</code>	<code>parseExit</code>
<code>></code>	<code>parseGreaterThan</code>
<code>>=</code>	<code>parseGreaterEqual</code>
<code>has</code>	<code>parseHas</code>
<code>IF</code>	<code>parseIf</code>
<code>implies</code>	<code>parseImplies</code>
<code>IN</code>	<code>parseIn</code>
<code>INBY</code>	<code>parseInBy</code>
<code>is</code>	<code>parseIs</code>
<code>isnt</code>	<code>parseIsnt</code>
<code>Join</code>	<code>parseJoin</code>
<code>leave</code>	<code>parseLeave</code>
<code>;;control-H</code>	<code>parseLeftArrow</code>
<code><=</code>	<code>parseLessEqual</code>
<code>LET</code>	<code>parseLET</code>
<code>LETD</code>	<code>parseLETD</code>
<code>MDEF</code>	<code>parseMDEF</code>
<code>^</code>	<code>parseNot</code>
<code>not</code>	<code>parseNot</code>
<code>^=</code>	<code>parseNotEqual</code>
<code>or</code>	<code>parseOr</code>
<code>pretend</code>	<code>parsePretend</code>
<code>return</code>	<code>parseReturn</code>
<code>SEGMENT</code>	<code>parseSegment</code>

```

SEQ          parseSeq
;;control-V  parseUpArrow
VCONS       parseVCONS
where        parseWhere

```

4.2.1 defplist parseAnd

— postvars —

```

(eval-when (eval load)
  (setf (get '|and| '|parseTran|) '|parseAnd|))

```

4.2.2 defun parseAnd

```

[parseTran p93]
[parseAnd p97]
[parseTranList p95]
[parseIf p114]
[$InteractiveMode p??]

```

— defun parseAnd —

```

(defun |parseAnd| (arg)
  (cond
    (|$InteractiveMode| (cons '|and| (|parseTranList| arg)))
    ((null arg) '|true|)
    ((null (cdr arg)) (car arg))
    (t
     (|parseIf|
      (list (|parseTran| (car arg)) (|parseAnd| (cdr arg)) '|false| )))))

```

4.2.3 defplist parseAtSign

— postvars —

```

(eval-when (eval load)
  (setf (get '@ '|parseTran|) '|parseAtSign|))

```

4.2.4 defun parseAtSign

```
[parseTran p93]
[parseType p98]
[$InteractiveMode p??]
```

— defun parseAtSign —

```
(defun |parseAtSign| (arg)
  (declare (special |$InteractiveMode|))
  (if |$InteractiveMode|
    (list '@ (|parseTran| (first arg)) (|parseTran| (|parseType| (second arg))))
    (list '@ (|parseTran| (first arg)) (|parseTran| (second arg)))))
```

4.2.5 defun parseType

```
[msubst p??]
[parseTran p93]
```

— defun parseType —

```
(defun |parseType| (x)
  (declare (special |$EmptyMode| |$quadSymbol|))
  (setq x (msubst |$EmptyMode| |$quadSymbol| x))
  (if (and (consp x) (eq (qfirst x) '|typeOf|)
        (consp (qrest x)) (eq (qcddr x) nil))
    (list '|typeOf| (|parseTran| (qsecond x)))
    x))
```

4.2.6 defplist parseCategory

— postvars —

```
(eval-when (eval load)
  (setf (get 'category '|parseTran|) '|parseCategory|))
```

4.2.7 defun parseCategory

```
[parseTranList p95]
[parseDropAssertions p99]
[contained p??]
```

— defun parseCategory —

```
(defun |parseCategory| (arg)
  (let (z key)
    (setq z (|parseTranList| (|parseDropAssertions| arg)))
    (setq key (if (contained '$ z) '|domain| '|package|))
    (cons 'category (cons key z))))
```

—————

4.2.8 defun parseDropAssertions

```
[parseDropAssertions p99]
```

— defun parseDropAssertions —

```
(defun |parseDropAssertions| (x)
  (cond
    ((not (consp x)) x)
    ((and (consp (qfirst x)) (eq (qcaar x) 'if)
          (consp (qcdar x))
          (eq (qcadar x) '|asserted|))
     (|parseDropAssertions| (qrest x)))
    (t (cons (qfirst x) (|parseDropAssertions| (qrest x))))))
```

—————

4.2.9 defpllist parseCoerce

— postvars —

```
(eval-when (eval load)
  (setf (get '|::| '|parseTran|) '|parseCoerce|))
```

—————

4.2.10 defun parseCoerce

```
[parseType p98]
[parseTran p93]
[$InteractiveMode p??]
```

— defun parseCoerce —

```
(defun |parseCoerce| (arg)
  (if |$InteractiveMode|
    (list '|::|
      (|parseTran| (first arg)) (|parseTran| (|parseType| (second arg))))
    (list '|::| (|parseTran| (first arg)) (|parseTran| (second arg)))))
```

—————

4.2.11 defplist parseColon

— postvars —

```
(eval-when (eval load)
  (setf (get '|:| '|parseTran|) '|parseColon|))
```

—————

4.2.12 defun parseColon

```
[parseTran p93]
[parseType p98]
[$InteractiveMode p??]
[$insideConstructIfTrue p??]
```

— defun parseColon —

```
(defun |parseColon| (arg)
  (declare (special |$insideConstructIfTrue|))
  (cond
    ((and (consp arg) (eq (qrest arg) nil))
     (list '|:| (|parseTran| (first arg))))
    ((and (consp arg) (consp (qrest arg)) (eq (qcddr arg) nil))
     (if |$InteractiveMode|
       (if |$insideConstructIfTrue|
         (list 'tag (|parseTran| (first arg))
```

```

(|parseTran| (second arg)))
(list '|:| (|parseTran| (first arg))
      (|parseTran| (|parseType| (second arg)))))
(list '|:| (|parseTran| (first arg))
      (|parseTran| (second arg)))))

```

4.2.13 defplist parseDEF

— postvars —

```

(eval-when (eval load)
  (setf (get 'def '|parseTran|) '|parseDEF|))

```

4.2.14 defun parseDEF

```

[setDefOp p386]
[parseLhs p102]
[parseTranList p95]
[parseTranCheckForRecord p457]
[opFf p??]
[$lhs p??]

```

— defun parseDEF —

```

(defun |parseDEF| (arg)
  (let (|$lhs| tList specialList body)
    (declare (special |$lhs|))
    (setq |$lhs| (first arg))
    (setq tList (second arg))
    (setq specialList (third arg))
    (setq body (fourth arg))
    (|setDefOp| |$lhs|)
    (list 'def (|parseLhs| |$lhs|)
          (|parseTranList| tList)
          (|parseTranList| specialList)
          (|parseTranCheckForRecord| body (|opOf| |$lhs|)))))

```

4.2.15 defun parseLhs

[parseTran p93]
[transIs p102]

— **defun parseLhs** —

```
(defun |parseLhs| (x)
  (let (result)
    (cond
      ((atom x) (|parseTran| x))
      ((atom (car x))
       (cons (|parseTran| (car x))
              (dolist (y (cdr x) (nreverse result))
                (push (|transIs| (|parseTran| y)) result))))
      (t (|parseTran| x)))))
```

4.2.16 defun transIs

[isListConstructor p103]
[transIs1 p102]

— **defun transIs** —

```
(defun |transIs| (u)
  (if (|isListConstructor| u)
      (cons '|construct| (|transIs1| u))
      u))
```

4.2.17 defun transIs1

[qcar p??]
[qcdr p??]
[nreverse0 p??]
[transIs p102]
[transIs1 p102]

— **defun transIs1** —

```
(defun |transIs1| (u)
```

```

(let (x h v tmp3)
  (cond
    ((and (consp u) (eq (qfirst u) '|construct|))
      (dolist (x (qrest u) (nreverse0 tmp3))
        (push (|transIs| x) tmp3)))
    ((and (consp u) (eq (qfirst u) '|append|) (consp (qrest u))
      (consp (qcddr u)) (eq (qcdddr u) nil))
      (setq x (qsecond u))
      (setq h (list '|:| (|transIs| x)))
      (setq v (|transIs1| (qthird u)))
      (cond
        ((and (consp v) (eq (qfirst v) '|:|)
          (consp (qrest v)) (eq (qcddr v) nil))
          (list h (qsecond v)))
        ((eq v '|nil|) (car (cdr h)))
        ((atom v) (list h (list '|:| v)))
        (t (cons h v))))
    ((and (consp u) (eq (qfirst u) '|cons|) (consp (qrest u))
      (consp (qcddr u)) (eq (qcdddr u) nil))
      (setq h (|transIs| (qsecond u)))
      (setq v (|transIs1| (qthird u)))
      (cond
        ((and (consp v) (eq (qfirst v) '|:|) (consp (qrest v))
          (eq (qcddr v) nil))
          (cons h (list (qsecond v))))
        ((eq v '|nil|) (cons h nil))
        ((atom v) (list h (list '|:| v)))
        (t (cons h v))))
    (t u))))

```

4.2.18 defun isListConstructor

[member p??]

— defun isListConstructor —

```

(defun |isListConstructor| (u)
  (and (consp u) (|member| (qfirst u) '(|construct| |append| |cons|))))

```

4.2.19 defplist parseDollarGreaterthan

— postvars —

```
(eval-when (eval load)
  (setf (get '|$>| ' |parseTran|) '|parseDollarGreaterthan|))
```

4.2.20 defun parseDollarGreaterThan

```
[msubst p??]
[parseTran p93]
[$op p??]
```

— defun parseDollarGreaterThan —

```
(defun |parseDollarGreaterthan| (arg)
  (declare (special |$op|))
  (list (msubst '$< '$> |$op|)
        (|parseTran| (second arg))
        (|parseTran| (first arg))))
```

4.2.21 defplist parseDollarGreaterEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '|$>=| ' |parseTran|) '|parseDollarGreaterEqual|))
```

4.2.22 defun parseDollarGreaterEqual

```
[msubst p??]
[parseTran p93]
[$op p??]
```

— defun parseDollarGreaterEqual —

```
(defun |parseDollarGreaterEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '$< '$>= |$op|) arg)))))
```

— postvars —

```
(eval-when (eval load)
  (setf (get '$<=| '|parseTran|) '|parseDollarLessEqual|))
```

4.2.23 defun parseDollarLessEqual

```
[msubst p??]
[parseTran p93]
[$op p??]
```

— defun parseDollarLessEqual —

```
(defun |parseDollarLessEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '$> '$<= |$op|) arg)))))
```

4.2.24 defplist parseDollarNotEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '$^=| '|parseTran|) '|parseDollarNotEqual|))
```

4.2.25 defun parseDollarNotEqual

```
[parseTran p93]
[msubst p??]
```


`[$op p??]`

— `defun parseDollarNotEqual` —

```
(defun |parseDollarNotEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '$= '$~= |$op|) arg))))
```

—————

4.2.26 `defplist parseEquivalence`

— `postvars` —

```
(eval-when (eval load)
  (setf (get '|eqv| '|parseTran|) '|parseEquivalence|))
```

—————

4.2.27 `defun parseEquivalence`

`[parseIf p114]`

— `defun parseEquivalence` —

```
(defun |parseEquivalence| (arg)
  (|parseIf|
   (list (first arg) (second arg)
         (|parseIf| (cons (second arg) '(|false| |true|))))))
```

—————

4.2.28 `defplist parseExit`

— `postvars` —

```
(eval-when (eval load)
  (setf (get '|exit| '|parseTran|) '|parseExit|))
```

—————

4.2.29 defun parseExit

```
[parseTran p93]
[moan p??]
```

— defun parseExit —

```
(defun |parseExit| (arg)
  (let (a b)
    (setq a (|parseTran| (car arg)))
    (setq b (|parseTran| (cdr arg)))
    (if b
      (cond
        ((null (integerp a))
         (moan "first arg " a " for exit must be integer")
         (list '|exit| 1 a ))
        (t
         (cons '|exit| (cons a b))))
      (list '|exit| 1 a ))))
```

4.2.30 defplist parseGreaterEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '|>=| '|parseTran|) '|parseGreaterEqual|))
```

4.2.31 defun parseGreaterEqual

```
[parseTran p93]
[$op p??]
```

— defun parseGreaterEqual —

```
(defun |parseGreaterEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '<' '>= '|$op|) arg))))
```

4.2.32 defplist parseGreaterThan

— postvars —

```
(eval-when (eval load)
  (setf (get '|>' '|parseTran|) '|parseGreaterThan|))
```

—————

4.2.33 defun parseGreaterThan

```
[parseTran p93]
[$op p??]
```

— defun parseGreaterThan —

```
(defun |parseGreaterThan| (arg)
  (declare (special |$op|))
  (list (msubst '<' '>' |$op|)
        (|parseTran| (second arg)) (|parseTran| (first arg))))
```

—————

4.2.34 defplist parseHas

— postvars —

```
(eval-when (eval load)
  (setf (get '|has| '|parseTran|) '|parseHas|))
```

—————

4.2.35 defun parseHas

```
[unabbrevAndLoad p??]
[qcar p??]
[qcdr p??]
[getdatabase p??]
[opOf p??]
[makeNonAtomic p??]
```

```
[parseHasRhs p110]
[member p??]
[parseType p98]
[nreverse0 p??]
[$InteractiveMode p??]
[$CategoryFrame p??]
```

— defun parseHas —

```
(defun |parseHas| (arg)
  (labels (
    (fn (arg)
      (let (tmp4 tmp6 map op kk)
        (declare (special |$InteractiveMode|))
        (when |$InteractiveMode| (setq arg (|unabbrevAndLoad| arg)))
        (cond
          ((and (consp arg) (eq (qfirst arg) '|:|) (consp (qrest arg))
            (consp (qcddr arg)) (eq (qcdddr arg) nil)
            (consp (qthird arg))
            (eq (qcaaddr arg) '|Mapping|))
           (setq map (rest (third arg)))
           (setq op (second arg))
           (setq op (if (stringp op) (intern op) op))
           (list (list 'signature op map)))
          ((and (consp arg) (eq (qfirst arg) '|Join|))
           (dolist (z (rest arg) tmp4)
             (setq tmp4 (append tmp4 (fn z))))))
          ((and (consp arg) (eq (qfirst arg) 'category))
           (dolist (z (rest arg) tmp6)
             (setq tmp6 (append tmp6 (fn z))))))
          (t
           (setq kk (getdatabase (|opOf| arg) 'constructorkind))
           (cond
            ((or (eq kk '|domain|) (eq kk '|category|))
             (list (|makeNonAtomic| arg)))
            ((and (consp arg) (eq (qfirst arg) 'attribute))
             (list arg))
            ((and (consp arg) (eq (qfirst arg) 'signature))
             (list arg))
            (|$InteractiveMode|
             (|parseHasRhs| arg))
            (t
             (list (list 'attribute arg)))))))
    (let (tmp1 tmp2 tmp3 x)
      (declare (special |$InteractiveMode| |$CategoryFrame|))
      (setq x (first arg))
      (setq tmp1 (|get| x '|value| |$CategoryFrame|))
      (when |$InteractiveMode|
        (setq x
```

```

      (if (and (consp tmp1) (consp (qrest tmp1)) (consp (qcddr tmp1))
              (eq (qcddr tmp1) nil)
              (|member| (second tmp1)
                        '((|Mode|) (|Domain|) (|SubDomain| (|Domain|)))))
          (first tmp1)
          (|parseType| x))))
    (setq tmp2
      (dolist (u (fn (second arg)) (nreverse0 tmp3))
        (push (list '|has| x u ) tmp3)))
    (if (and (consp tmp2) (eq (qrest tmp2) nil))
        (qfirst tmp2)
        (cons '|and| tmp2))))

```

4.2.36 defun parseHasRhs

```

[get p??]
[qcar p??]
[qcdr p??]
[member p??]
[abbreviation? p??]
[loadIfNecessary p11]
[unabbrevAndLoad p??]
[$CategoryFrame p??]

```

— defun parseHasRhs —

```

(defun |parseHasRhs| (u)
  (let (tmp1 y)
    (declare (special |$CategoryFrame|))
    (setq tmp1 (|get| u '|value| |$CategoryFrame|))
    (cond
      ((and (consp tmp1) (consp (qrest tmp1))
              (consp (qcddr tmp1)) (eq (qcddr tmp1) nil)
              (|member| (second tmp1)
                        '((|Mode|) (|Domain|) (|SubDomain| (|Domain|)))))
          (second tmp1))
      ((setq y (|abbreviation?| u))
          (if (|loadIfNecessary| y)
              (list (|unabbrevAndLoad| y))
              (list (list 'attribute u))))
      (t (list (list 'attribute u)))))

```

4.2.37 defun loadIfNecessary

```
[loadLibIfNecessary p111]
```

— **defun loadIfNecessary** —

```
(defun |loadIfNecessary| (u)
  (|loadLibIfNecessary| u t))
```

4.2.38 defun loadLibIfNecessary

```
[loadLibIfNecessary p111]
[functionp p??]
[macrop p??]
[getl p??]
[loadLib p??]
[lassoc p??]
[getProplist p??]
[getdatabase p??]
[updateCategoryFrameForCategory p113]
[updateCategoryFrameForConstructor p112]
[throwKeyedMsg p??]
[$CategoryFrame p??]
[$InteractiveMode p??]
```

— **defun loadLibIfNecessary** —

```
(defun |loadLibIfNecessary| (u mustExist)
  (let (value y)
    (declare (special |$CategoryFrame| |$InteractiveMode|))
    (cond
      ((eq u '|$EmptyMode|) u)
      ((null (atom u)) (|loadLibIfNecessary| (car u) mustExist))
      (t
       (setq value
        (cond
          ((or (|functionp| u) (|macrop| u)) u)
          ((getl u 'loaded) u)
          ((|loadLib| u) u)))
        (cond
          ((and (null |$InteractiveMode|)
            (or (null (setq y (|getProplist| u |$CategoryFrame|)))
              (and (null (lassoc '|isFunctor| y))
                (null (lassoc '|isCategory| y))))))
```

```

      (if (setq y (getdatabase u 'constructorkind))
        (if (eq y '|category|)
          (|updateCategoryFrameForCategory| u)
          (|updateCategoryFrameForConstructor| u))
        (|throwKeyedMsg| 's2il0005 (list u))))
      (t value))))))

```

4.2.39 defun updateCategoryFrameForConstructor

```

[getdatabase p??]
[put p??]
[convertOpAlist2compilerInfo p112]
[addModemap p245]
[$CategoryFrame p??]
[$CategoryFrame p??]

```

— defun updateCategoryFrameForConstructor —

```

(defun |updateCategoryFrameForConstructor| (constructor)
  (let (opAlist tmp1 dc sig pred impl)
    (declare (special |$CategoryFrame|))
    (setq opalist (getdatabase constructor 'operationalist))
    (setq tmp1 (getdatabase constructor 'constructormodemap))
    (setq dc (caar tmp1))
    (setq sig (cdar tmp1))
    (setq pred (caadr tmp1))
    (setq impl (cadadr tmp1))
    (setq |$CategoryFrame|
      (|put| constructor '|isFunction|
        (|convertOpAlist2compilerInfo| opalist)
        (|addModemap| constructor dc sig pred impl
          (|put| constructor '|mode| (cons '|Mapping| sig) |$CategoryFrame|))))))

```

4.2.40 defun convertOpAlist2compilerInfo

— defun convertOpAlist2compilerInfo —

```

(defun |convertOpAlist2compilerInfo| (opalist)
  (labels (
    (formatSig (op arg2)

```

```

(let (typelist slot stuff pred impl)
  (setq typelist (car arg2))
  (setq slot (cadr arg2))
  (setq stuff (caddr arg2))
  (setq pred (if stuff (car stuff) t))
  (setq impl (if (cdr stuff) (cadr stuff) 'elt))
  (list (list op typelist) pred (list impl '$ slot))))
(let (data result)
  (setq data
    (loop for item in opalist
      collect
        (loop for sig in (rest item)
          collect (formatSig (car item) sig))))
  (dolist (term data result)
    (setq result (append result term)))))

```

4.2.41 defun updateCategoryFrameForCategory

```

[getdatabase p??]
[put p??]
[addModemap p245]
[$CategoryFrame p??]
[$CategoryFrame p??]

```

— defun updateCategoryFrameForCategory —

```

(defun |updateCategoryFrameForCategory| (category)
  (let (tmp1 dc sig pred impl)
    (declare (special |$CategoryFrame|))
    (setq tmp1 (getdatabase category 'constructormodemap))
    (setq dc (caar tmp1))
    (setq sig (cdar tmp1))
    (setq pred (caadr tmp1))
    (setq impl (cadadr tmp1))
    (setq |$CategoryFrame|
      (|put| category '|isCategory| t
        (|addModemap| category dc sig pred impl |$CategoryFrame|))))

```

4.2.42 defplist parseIf

— postvars —


```
(eval-when (eval load)
  (setf (get 'if '|parseTran|) '|parseIf|))
```

4.2.43 defun parseIf

```
[parseIf,ifTran p114]
[parseTran p93]
```

— defun parseIf —

```
(defun |parseIf| (arg)
  (if (null (and (consp arg) (consp (qrest arg))
                (consp (qcddr arg)) (eq (qcddr arg) nil)))
      arg
      (|parseIf,ifTran|
        (|parseTran| (first arg))
        (|parseTran| (second arg))
        (|parseTran| (third arg)))))
```

4.2.44 defun parseIf,ifTran

```
[parseIf,ifTran p114]
[incExitLevel p??]
[makeSimplePredicateOrNil p458]
[incExitLevel p??]
[parseTran p93]
[$InteractiveMode p??]
```

— defun parseIf,ifTran —

```
(defun |parseIf,ifTran| (pred a b)
  (let (pp z ap bp tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 val s)
    (declare (special |$InteractiveMode|))
    (cond
      ((and (null |$InteractiveMode|) (eq pred '|true|))
       a)
      ((and (null |$InteractiveMode|) (eq pred '|false|))
       b)
      ((and (consp pred) (eq (qfirst pred) '|not|)
            (consp (qrest pred)) (eq (qcddr pred) nil))
       (|parseIf,ifTran| (second pred) b a))
```

```

((and (consp pred) (eq (qfirst pred) 'if)
  (progn
    (setq tmp1 (qrest pred))
    (and (consp tmp1)
      (progn
        (setq pp (qfirst tmp1))
        (setq tmp2 (qrest tmp1))
        (and (consp tmp2)
          (progn
            (setq ap (qfirst tmp2))
            (setq tmp3 (qrest tmp2))
            (and (consp tmp3)
              (eq (qrest tmp3) nil)
              (progn (setq bp (qfirst tmp3)) t)))))))
  (|parseIf,ifTran| pp
    (|parseIf,ifTran| ap (copy a) (copy b))
    (|parseIf,ifTran| bp a b)))
((and (consp pred) (eq (qfirst pred) 'seq)
  (consp (qrest pred)) (progn (setq tmp2 (reverse (qrest pred))) t)
  (and (consp tmp2)
    (consp (qfirst tmp2))
    (eq (qcaar tmp2) '|exit|)
    (progn
      (setq tmp4 (qcddr tmp2))
      (and (consp tmp4)
        (equal (qfirst tmp4) 1)
        (progn
          (setq tmp5 (qrest tmp4))
          (and (consp tmp5)
            (eq (qrest tmp5) nil)
            (progn (setq pp (qfirst tmp5)) t))))
      (progn (setq z (qrest tmp2)) t))
    (progn (setq z (nreverse z)) t))
  (cons 'seq
    (append z
      (list
        (list '|exit| 1 (|parseIf,ifTran| pp
          (|incExitLevel| a)
          (|incExitLevel| b)))))))
((and (consp a) (eq (qfirst a) 'if) (consp (qrest a))
  (equal (qsecond a) pred) (consp (qcddr a))
  (consp (qcdddr a))
  (eq (qcdddr a) nil))
  (list 'if pred (third a) b))
((and (consp b) (eq (qfirst b) 'if)
  (consp (qrest b)) (equal (qsecond b) pred)
  (consp (qcddr b))
  (consp (qcdddr b))
  (eq (qcdddr b) nil))
  (list 'if pred a (fourth b)))

```

```

((progn
  (setq tmp1 (|makeSimplePredicateOrNil| pred))
  (and (consp tmp1) (eq (qfirst tmp1) 'seq)
    (progn
      (setq tmp2 (qrest tmp1))
      (and (and (consp tmp2)
        (progn (setq tmp3 (reverse tmp2)) t))
        (and (consp tmp3)
          (progn
            (setq tmp4 (qfirst tmp3))
            (and (consp tmp4) (eq (qfirst tmp4) '|exit|)
              (progn
                (setq tmp5 (qrest tmp4))
                (and (consp tmp5) (equal (qfirst tmp5) 1)
                  (progn
                    (setq tmp6 (qrest tmp5))
                    (and (consp tmp6) (eq (qrest tmp6) nil)
                      (progn (setq val (qfirst tmp6)) t)))))))
                (progn (setq s (qrest tmp3)) t))))))
      (setq s (nreverse s))
      (|parseTran|
        (cons 'seq
          (append s
            (list (list '|exit| 1 (|incExitLevel| (list 'if val a b))))))))
      (t
        (list 'if pred a b )))))

```

4.2.45 defplist parseImplies

— postvars —

```

(eval-when (eval load)
  (setf (get '|implies| '|parseTran|) '|parseImplies|))

```

4.2.46 defun parseImplies

[parseIf p114]

— defun parseImplies —

```
(defun |parseImplies| (arg)
  (|parseIf| (list (first arg) (second arg) '|true|)))
```

4.2.47 defplist parseIn

— postvars —

```
(eval-when (eval load)
  (setf (get 'in '|parseTran|) '|parseIn|))
```

4.2.48 defun parseIn

```
[parseTran p93]
[postError p356]
```

— defun parseIn —

```
(defun |parseIn| (arg)
  (let (i n)
    (setq i (|parseTran| (first arg)))
    (setq n (|parseTran| (second arg)))
    (cond
      ((and (consp n) (eq (qfirst n) 'segment)
            (consp (qrest n)) (eq (qcddr n) nil))
       (list 'step i (second n) 1))
      ((and (consp n) (eq (qfirst n) '|reverse|)
            (consp (qrest n)) (eq (qcddr n) nil)
            (consp (qsecond n)) (eq (qcaadr n) 'segment)
            (consp (qcdadr n))
            (eq (qcddadr n) nil))
       (|postError| (list " You cannot reverse an infinite sequence." )))
      ((and (consp n) (eq (qfirst n) 'segment)
            (consp (qrest n)) (consp (qcddr n))
            (eq (qcdddr n) nil))
       (if (third n)
           (list 'step i (second n) 1 (third n))
           (list 'step i (second n) 1)))
      ((and (consp n) (eq (qfirst n) '|reverse|)
            (consp (qrest n)) (eq (qcddr n) nil)
            (consp (qsecond n)) (eq (qcaadr n) 'segment)
```

```

      (consp (qcdadr n))
      (consp (qcddadr n))
      (eq (qrest (qcddadr n)) nil))
    (if (third (second n))
        (list 'step i (third (second n)) -1 (second (second n)))
        (|postError| (list " You cannot reverse an infinite sequence."))))
    ((and (consp n) (eq (qfirst n) '|tails|)
          (consp (qrest n)) (eq (qcddr n) nil))
      (list 'on i (second n)))
    (t
      (list 'in i n))))

```

4.2.49 defplist parseInBy

— postvars —

```

(eval-when (eval load)
  (setf (get 'inby '|parseTran|) '|parseInBy|))

```

4.2.50 defun parseInBy

```

[postError p356]
[parseTran p93]
[bright p??]
[parseIn p117]

```

— defun parseInBy —

```

(defun |parseInBy| (arg)
  (let (i n inc u)
    (setq i (first arg))
    (setq n (second arg))
    (setq inc (third arg))
    (setq u (|parseIn| (list i n)))
    (cond
      ((null (and (consp u) (eq (qfirst u) 'step)
                  (consp (qrest u))
                  (consp (qcddr u))
                  (consp (qcdddr u)))))
      (|postError|

```

```

      (cons '| You cannot use|
        (append (|bright| "by")
          (list "except for an explicitly indexed sequence."))))))
(t
  (setq inc (|parseTran| inc))
  (cons 'step
    (cons (second u)
      (cons (third u)
        (cons (|parseTran| inc) (cddddr u))))))))))

```

4.2.51 defplist parseIs

— postvars —

```

(eval-when (eval load)
  (setf (get '|is| '|parseTran|) '|parseIs|))

```

4.2.52 defun parseIs

```

[parseTran p93]
[transIs p102]

```

— defun parseIs —

```

(defun |parseIs| (arg)
  (list '|is| (|parseTran| (first arg)) (|transIs| (|parseTran| (second arg)))))

```

4.2.53 defplist parseIsnt

— postvars —

```

(eval-when (eval load)
  (setf (get '|isnt| '|parseTran|) '|parseIsnt|))

```

4.2.54 defun parseIsnt

[parseTran p93]
 [transIs p102]

— defun parseIsnt —

```
(defun |parseIsnt| (arg)
  (list '|isnt|
        (|parseTran| (first arg))
        (|transIs| (|parseTran| (second arg)))))
```

—————

4.2.55 defplist parseJoin

— postvars —

```
(eval-when (eval load)
  (setf (get '|Join| '|parseTran|) '|parseJoin|))
```

—————

4.2.56 defun parseJoin

[parseTranList p95]

— defun parseJoin —

```
(defun |parseJoin| (thejoin)
  (labels (
    (fn (arg)
      (cond
        ((null arg)
         nil)
        ((and (consp arg) (consp (qfirst arg)) (eq (qcaar arg) '|Join|))
         (append (cdar arg) (fn (rest arg))))
        (t
         (cons (first arg) (fn (rest arg)))))))
    )
  (cons '|Join| (fn (|parseTranList| thejoin)))))
```

—————

4.2.57 defplist parseLeave

— postvars —

```
(eval-when (eval load)
  (setf (get '|leave| '|parseTran|) '|parseLeave|))
```

4.2.58 defun parseLeave

[parseTran p93]

— defun parseLeave —

```
(defun |parseLeave| (arg)
  (let (a b)
    (setq a (|parseTran| (car arg)))
    (setq b (|parseTran| (cdr arg)))
    (cond
      (b
       (cond
         ((null (integerp a))
          (moan "first arg " a " for 'leave' must be integer")
          (list '|leave| 1 a))
         (t (cons '|leave| (cons a b))))))
    (t (list '|leave| 1 a)))))
```

4.2.59 defplist parseLessEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '|<=| '|parseTran|) '|parseLessEqual|))
```

4.2.60 defun parseLessEqual

```
[parseTran p93]
[$op p??]
```

— defun parseLessEqual —

```
(defun |parseLessEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '>' '<=' |$op|) arg)))))
```

—————

4.2.61 defplist parseLET

— postvars —

```
(eval-when (eval load)
  (setf (get 'let '|parseTran|) '|parseLET|))
```

—————

4.2.62 defun parseLET

```
[parseTran p93]
[parseTranCheckForRecord p457]
[opOf p??]
[transIs p102]
```

— defun parseLET —

```
(defun |parseLET| (arg)
  (let (p)
    (setq p
      (list 'let (|parseTran| (first arg))
        (|parseTranCheckForRecord| (second arg) (|opOf| (first arg)))))
    (if (eq (|opOf| (first arg)) '|cons|)
      (list 'let (|transIs| (second p)) (third p))
      p)))
```

—————

4.2.63 defplist parseLETD

— postvars —

```
(eval-when (eval load)
  (setf (get 'letd '|parseTran|) '|parseLETD|))
```

4.2.64 defun parseLETD

```
[parseTran p93]
[parseType p98]
```

— defun parseLETD —

```
(defun |parseLETD| (arg)
  (list 'letd
    (|parseTran| (first arg))
    (|parseTran| (|parseType| (second arg)))))
```

4.2.65 defplist parseMDEF

— postvars —

```
(eval-when (eval load)
  (setf (get 'mdef '|parseTran|) '|parseMDEF|))
```

4.2.66 defun parseMDEF

```
[parseTran p93]
[parseTranList p95]
[parseTranCheckForRecord p457]
[opOf p??]
[$lhs p??]
```

— defun parseMDEF —

```
(defun |parseMDEF| (arg)
  (let (|$lhs|)
    (declare (special |$lhs|))
    (setq |$lhs| (first arg))
    (list 'mdef
      (|parseTran| |$lhs|)
      (|parseTranList| (second arg))
      (|parseTranList| (third arg))
      (|parseTranCheckForRecord| (fourth arg) (|opOf| |$lhs|))))
```

4.2.67 defplist parseNot

— postvars —

```
(eval-when (eval load)
  (setf (get '|not| '|parseTran|) '|parseNot|))
```

4.2.68 defplist parseNot

— postvars —

```
(eval-when (eval load)
  (setf (get '|^| '|parseTran|) '|parseNot|))
```

4.2.69 defun parseNot

```
[parseTran p93]
[$InteractiveMode p??]
```

— defun parseNot —

```
(defun |parseNot| (arg)
  (declare (special |$InteractiveMode|))
  (if |$InteractiveMode|
    (list '|not| (|parseTran| (car arg)))
```

```
(|parseTran| (cons 'if (cons (car arg) '(|false| |true|))))))
```

4.2.70 defplist parseNotEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '|^=' '|parseTran|) '|parseNotEqual|))
```

4.2.71 defun parseNotEqual

```
[parseTran p93]
[msubst p??]
[$op p??]
```

— defun parseNotEqual —

```
(defun |parseNotEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '= '^= |$op|) arg))))
```

4.2.72 defplist parseOr

— postvars —

```
(eval-when (eval load)
  (setf (get '|or| '|parseTran|) '|parseOr|))
```

4.2.73 defun parseOr

```
[parseTran p93]
[parseTranList p95]
```

[parseIf p114]
 [parseOr p125]

— defun parseOr —

```
(defun |parseOr| (arg)
  (let (x)
    (setq x (|parseTran| (car arg)))
    (cond
      ($InteractiveMode| (cons '|or| (|parseTranList| arg)))
      ((null arg) '|false|)
      ((null (cdr arg)) (car arg))
      ((and (consp x) (eq (qfirst x) '|not|)
            (consp (qrest x)) (eq (qcddr x) nil))
       (|parseIf| (list (second x) (|parseOr| (cdr arg)) '|true|)))
      (t
       (|parseIf| (list x '|true| (|parseOr| (cdr arg))))))))
```

—

4.2.74 defplist parsePretend

— postvars —

```
(eval-when (eval load)
  (setf (get '|pretend| '|parseTran|) '|parsePretend|))
```

—

4.2.75 defun parsePretend

[parseTran p93]
 [parseType p98]

— defun parsePretend —

```
(defun |parsePretend| (arg)
  (if |$InteractiveMode|
    (list '|pretend|
          (|parseTran| (first arg))
          (|parseTran| (|parseType| (second arg)))))
    (list '|pretend|
          (|parseTran| (first arg))
          (|parseTran| (second arg)))))
```

4.2.76 defplist parseReturn

— postvars —

```
(eval-when (eval load)
  (setf (get '|return| '|parseTran|) '|parseReturn|))
```

4.2.77 defun parseReturn

```
[parseTran p93]
[moan p??]
```

— defun parseReturn —

```
(defun |parseReturn| (arg)
  (let (a b)
    (setq a (|parseTran| (car arg)))
    (setq b (|parseTran| (cdr arg)))
    (cond
     (b
      (when (nequal a 1) (moan "multiple-level 'return' not allowed"))
      (cons '|return| (cons 1 b)))
     (t (list '|return| 1 a)))))
```

4.2.78 defplist parseSegment

— postvars —

```
(eval-when (eval load)
  (setf (get '|segment| '|parseTran|) '|parseSegment|))
```

4.2.79 defun parseSegment

[parseTran p93]

— defun parseSegment —

```
(defun |parseSegment| (arg)
  (if (and (consp arg) (consp (qrest arg)) (eq (qcddr arg) nil))
      (if (second arg)
          (list 'segment (|parseTran| (first arg)) (|parseTran| (second arg)))
          (list 'segment (|parseTran| (first arg))))
      (cons 'segment arg)))
```

4.2.80 defplist parseSeq

— postvars —

```
(eval-when (eval load)
  (setf (get 'seq '|parseTran|) '|parseSeq|))
```

4.2.81 defun parseSeq

```
[postError p356]
[transSeq p??]
[mapInto p??]
[last p??]
```

— defun parseSeq —

```
(defun |parseSeq| (arg)
  (let (tmp1)
    (when (consp arg) (setq tmp1 (reverse arg)))
    (if (null (and (consp arg) (consp tmp1)
                  (consp (qfirst tmp1)) (eq (qcaar tmp1) '|exit|))))
        (|postError| (list " Invalid ending to block: " (|last| arg)))
        (|transSeq| (|mapInto| arg '|parseTran|)))))
```

4.2.82 defplist parseVCONS

— postvars —

```
(eval-when (eval load)
  (setf (get 'vcons '|parseTran|) '|parseVCONS|))
```

4.2.83 defun parseVCONS

[parseTranList p95]

— defun parseVCONS —

```
(defun |parseVCONS| (arg)
  (cons 'vector (|parseTranList| arg)))
```

4.2.84 defplist parseWhere

— postvars —

```
(eval-when (eval load)
  (setf (get '|where| '|parseTran|) '|parseWhere|))
```

4.2.85 defun parseWhere

[mapInto p??]

— defun parseWhere —

```
(defun |parseWhere| (arg)
  (cons '|where| (|mapInto| arg '|parseTran|)))
```

Chapter 5

Compile Transformers

5.0.86 `defvar $NoValueMode`

— **initvars** —

```
(defvar |$NoValueMode| ' |NoValueMode|)
```

—————

5.0.87 `defvar $EmptyMode`

`$EmptyMode` is a constant whose value is `$EmptyMode`. It is used by `isPartialMode` to decide if a modemap is partially constructed. If the `$EmptyMode` constant occurs anywhere in the modemap structure at any depth then the modemap is still incomplete. To find this constant the `isPartialMode` function calls `CONTAINED $EmptyMode Y` which will walk the structure `Y` looking for this constant.

— **initvars** —

```
(defvar |$EmptyMode| ' |EmptyMode|)
```

—————

5.1 Routines for handling forms

The functions in this section are called through the symbol-plist of the symbol being parsed.

- `add (p247) compAdd(form mode env) → (form mode env)`

- @ (p343) compAtSign(form mode env) →
- CAPSULE (p250) compCapsule(form mode env) →
- case (p259) compCase(form mode env) →
- Mapping (p261) compCat(form mode env) →
- Record (p261) compCat(form mode env) →
- Union (p261) compCat(form mode env) →
- CATEGORY (p262) compCategory(form mode env) →
- :: (p343) compCoerce(form mode env) →
- : (p267) compColon(form mode env) →
- CONS (p270) compCons(form mode env) →
- construct (p272) compConstruct(form mode env) →
- ListCategory (p274) compConstructorCategory(form mode env) →
- RecordCategory (p274) compConstructorCategory(form mode env) →
- UnionCategory (p274) compConstructorCategory(form mode env) →
- VectorCategory (p274) compConstructorCategory(form mode env) →
- DEF (p274) compDefine(form mode env) →
- elt (p292) compElt(form mode env) →
- exit (p293) compExit(form mode env) →
- has (p294) compHas(pred mode \$e) →
- IF (p296) compIf(form mode env) →
- import (p304) compImport(form mode env) →
- is (p304) compIs(form mode env) →
- Join (p305) compJoin(form mode env) →
- +-> (p307) compLambda(form mode env) →
- leave (p308) compLeave(form mode env) →
- MDEF (p309) compMacro(form mode env) →
- pretend (p310) compPretend →
- QUOTE (p311) compQuote(form mode env) →

- REDUCE (p312) compReduce(form mode env) →
- COLLECT (p314) compRepeatOrCollect(form mode env) →
- REPEAT (p314) compRepeatOrCollect(form mode env) →
- return (p317) compReturn(form mode env) →
- SEQ (p318) compSeq(form mode env) →
- LET (p321) compSetq(form mode env) →
- SETQ (p321) compSetq(form mode env) →
- String (p331) compString(form mode env) →
- SubDomain (p331) compSubDomain(form mode env) →
- SubsetCategory (p333) compSubsetCategory(form mode env) →
- | (p334) compSuchthat(form mode env) →
- VECTOR (p335) compVector(form mode env) →
- where (p336) compWhere(form mode eInit) →

5.2 Functions which handle == statements

5.2.1 defun compDefineAddSignature

```
[hasFullSignature p134]
[assoc p??]
[lassoc p??]
[getProplist p??]
[comp p497]
[$EmptyMode p131]
```

— **defun compDefineAddSignature** —

```
(defun |compDefineAddSignature| (form signature env)
  (let (sig declForm)
    (declare (special |$EmptyMode|))
    (if
      (and (setq sig (|hasFullSignature| (rest form) signature env))
           (null (|assoc| (cons '$ sig)
                          (|lassoc| ' |modemap| (|getProplist| (car form) env)))))
      (progn
        (setq declForm
          (list ' |:
```

```

      (cons (car form)
            (loop for x in (rest form)
                  for m in (rest sig)
                  collect (list '|:| x m)))
      (car signature)))
    (third (|comp| declForm |$EmptyMode| env)))
  env)))

```

5.2.2 defun hasFullSignature

TPDHERE: test with BASTYPE [get p??]

— defun hasFullSignature —

```

(defun |hasFullSignature| (argl signature env)
  (let (target ml u)
    (setq target (first signature))
    (setq ml (rest signature))
    (when target
      (setq u
        (loop for x in argl for m in ml
              collect (or m (|get| x '|mode| env) (return 'failed))))
      (unless (eq u 'failed) (cons target u)))))

```

5.2.3 defun addEmptyCapsuleIfNecessary

[kar p??]
 [\$SpecialDomainNames p??]

— defun addEmptyCapsuleIfNecessary —

```

(defun |addEmptyCapsuleIfNecessary| (target rhs)
  (declare (special |$SpecialDomainNames|) (ignore target))
  (if (member (kar rhs) |$SpecialDomainNames|)
      rhs
      (list '|add| rhs (list 'capsule))))

```

5.2.4 defun getTargetFromRhs

[stackSemanticError p??]
 [getTargetFromRhs p135]
 [compOrCroak p496]

— defun getTargetFromRhs —

```
(defun |getTargetFromRhs| (lhs rhs env)
  (declare (special |$EmptyMode|))
  (cond
    ((and (consp rhs) (eq (qfirst rhs) 'capsule))
     (|stackSemanticError|
      (list "target category of " lhs
            " cannot be determined from definition")
      nil))
    ((and (consp rhs) (eq (qfirst rhs) '|SubDomain|) (consp (qrest rhs)))
     (|getTargetFromRhs| lhs (second rhs) env))
    ((and (consp rhs) (eq (qfirst rhs) '|add|)
          (consp (qrest rhs)) (consp (qcddr rhs))
          (eq (qcddr rhs) nil)
          (consp (qthird rhs))
          (eq (qcaaddr rhs) 'capsule))
     (|getTargetFromRhs| lhs (second rhs) env))
    ((and (consp rhs) (eq (qfirst rhs) '|Record|))
     (cons '|RecordCategory| (rest rhs)))
    ((and (consp rhs) (eq (qfirst rhs) '|Union|))
     (cons '|UnionCategory| (rest rhs)))
    ((and (consp rhs) (eq (qfirst rhs) '|List|))
     (cons '|ListCategory| (rest rhs)))
    ((and (consp rhs) (eq (qfirst rhs) '|Vector|))
     (cons '|VectorCategory| (rest rhs)))
    (t
     (second (|compOrCroak| rhs |$EmptyMode| env)))))
```

—————

5.2.5 defun giveFormalParametersValues

[put p??]
 [get p??]

— defun giveFormalParametersValues —

```
(defun |giveFormalParametersValues| (argl env)
  (dolist (x argl)
    (setq env
```

```
(|put| x '|value|
  (list (|genSomeVariable|) (|get| x '|mode| env) nil) env)))
env)
```

5.2.6 defun macroExpandInPlace

[macroExpand p136]

— defun macroExpandInPlace —

```
(defun |macroExpandInPlace| (form env)
  (let (y)
    (setq y (|macroExpand| form env))
    (if (or (atom form) (atom y))
        y
        (progn
          (rplaca form (car y))
          (rplacd form (cdr y))
          form
          )))))
```

5.2.7 defun macroExpand

[macroExpand p136]

[macroExpandList p137]

— defun macroExpand —

```
(defun |macroExpand| (form env)
  (let (u)
    (cond
      ((atom form)
       (if (setq u (|get| form '|macro| env))
           (|macroExpand| u env)
           form))
      ((and (consp form) (eq (qfirst form) 'def)
            (consp (qrest form))
            (consp (qcddr form))
            (consp (qcdddr form))
            (consp (qcddddr form))
            (eq (qrest (qcddddr form)) nil)))
```

```
(list 'def (|macroExpand| (second form) env)
      (|macroExpandList| (third form) env)
      (|macroExpandList| (fourth form) env)
      (|macroExpand| (fifth form) env)))
(t (|macroExpandList| form env))))
```

5.2.8 defun macroExpandList

[macroExpand p136]
[getdatabase p??]

— defun macroExpandList —

```
(defun |macroExpandList| (lst env)
  (let (tmp)
    (if (and (consp lst) (eq (qrest lst) nil)
          (identp (qfirst lst)) (getdatabase (qfirst lst) 'niladic)
          (setq tmp (|get| (qfirst lst) '|macro| env)))
        (|macroExpand| tmp env)
        (loop for x in lst collect (|macroExpand| x env)))))
```

5.2.9 defun compDefineCategory1

[compDefineCategory2 p142]
[makeCategoryPredicates p138]
[compDefine1 p275]
[mkCategoryPackage p139]
[\$insideCategoryPackageIfTrue p??]
[\$EmptyMode p131]
[\$categoryPredicateList p??]
[\$lisplibCategory p??]
[\$bootStrapMode p??]

— defun compDefineCategory1 —

```
(defun |compDefineCategory1| (df mode env prefix fal)
  (let (|$insideCategoryPackageIfTrue| |$categoryPredicateList| form
        sig sc cat body categoryCapsule d tmp1 tmp3)
    (declare (special |$insideCategoryPackageIfTrue| |$EmptyMode|
                      |$categoryPredicateList| |$lisplibCategory|
```



```

                                |$bootStrapMode|))
;; a category is a DEF form with 4 parts:
;; ((DEF (|BasicType|) ((|Category|)) (NIL)
;;   (|add| (CATEGORY |domain| (SIGNATURE = ((|Boolean|) $ $))
;;         (SIGNATURE ~= ((|Boolean|) $ $)))
;;   (CAPSULE (DEF (~= |x| |y|) ((|Boolean|) $ $) (NIL NIL NIL)
;;   (IF (= |x| |y|) |false| |true|))))))
(setq form (second df))
(setq sig (third df))
(setq sc (fourth df))
(setq body (fifth df))
(setq categoryCapsule
  (when (and (consp body) (eq (qfirst body) '|add|)
            (consp (qrest body)) (consp (qcddr body))
            (eq (qcddr body) nil))
    (setq tmp1 (third body))
    (setq body (second body))
    tmp1))
(setq tmp3 (|compDefineCategory2| form sig sc body mode env prefix fal))
(setq d (first tmp3))
(setq mode (second tmp3))
(setq env (third tmp3))
(when (and categoryCapsule (null |$bootStrapMode|))
  (setq |$insideCategoryPackageIfTrue| t)
  (setq |$categoryPredicateList|
    (|makeCategoryPredicates| form |$lisplibCategory|))
  (setq env (third
    (|compDefine1|
      (|mkCategoryPackage| form cat categoryCapsule) |$EmptyMode| env))))
(list d mode env)))

```

5.2.10 defun makeCategoryPredicates

[\$FormalMapVariableList p242]
 [\$TriangleVariableList p??]
 [\$mvl p??]
 [\$tv1 p??]

— defun makeCategoryPredicates —

```

(defun |makeCategoryPredicates| (form u)
  (labels (
    (fn (u pl)
      (declare (special |$tv1| |$mvl|))
      (cond

```

```

((and (consp u) (eq (qfirst u) '|Join|) (consp (qrest u)))
  (fn (car (reverse (qrest u))) pl))
((and (consp u) (eq (qfirst u) '|has|))
  (|insert| (eqsubstlist |$mvl| |$tv1| u) pl))
((and (consp u) (member (qfirst u) '(signature attribute))) pl)
(atom u) pl)
(t (fn1 u pl)))
(fn1 (u pl)
  (dolist (x u) (setq pl (fn x pl)))
  pl))
(declare (special |$FormalMapVariableList| |$mvl| |$tv1|
  |$TriangleVariableList|))
(setq |$tv1| (take (|#| (cdr form)) |$TriangleVariableList|))
(setq |$mvl| (take (|#| (cdr form)) (cdr |$FormalMapVariableList|)))
(fn u nil)))

```

5.2.11 defun mkCategoryPackage

```

[strconc p??]
[pname p??]
[getdatabase p??]
[abbreviationsSpad2Cmd p??]
[JoinInner p??]
[assoc p??]
[sublislis p??]
[msubst p??]
[$options p??]
[$categoryPredicateList p??]
[$e p??]
[$FormalMapVariableList p242]

```

— defun mkCategoryPackage —

```

(defun |mkCategoryPackage| (form cat def)
  (labels (
    (fn (x oplist)
      (cond
        ((atom x) oplist)
        ((and (consp x) (eq (qfirst x) 'def) (consp (qrest x)))
          (cons (second x) oplist))
        (t
          (fn (cdr x) (fn (car x) oplist))))))
    (gn (cat)
      (cond
        ((and (consp cat) (eq (qfirst cat) 'category)) (cddr cat))

```

```

      ((and (consp cat) (eq (qfirst cat) '|Join|)) (gn (|last| (qrest cat))))
      (t nil))))
(let (|$options| op argl packageName packageAbb nameForDollar packageArgl
      capsuleDefAlist explicitCatPart catvec fullCatOpList op1 sig
      catOpList packageCategory nils packageSig)
  (declare (special |$options| |$categoryPredicateList| |$e|
                    |$FormalMapVariableList|))
  (setq op (car form))
  (setq argl (cdr form))
  (setq packageName (intern (strconc (pname op) "&")))
  (setq packageAbb (intern (strconc (getdatabase op 'abbreviation) "-")))
  (setq |$options| nil)
  (|abbreviationsSpad2Cmd| (list '|domain| packageAbb packageName))
  (setq nameForDollar (car (setdifference '(s a b c d e f g h i) argl)))
  (setq packageArgl (cons nameForDollar argl))
  (setq capsuleDefAlist (fn def nil))
  (setq explicitCatPart (gn cat))
  (setq catvec (|eval| (|mkEvaluableCategoryForm| form)))
  (setq fullCatOpList (elt (|JoinInner| (list catvec) |$e|) 1))
  (setq catOpList
    (loop for x in fullCatOpList do
      (setq op1 (caar x))
      (setq sig (cadar x))
      when (|assoc| op1 capsuleDefAlist)
      collect (list 'signature op1 sig)))
  (when catOpList
    (setq packageCategory
      (cons 'category
        (cons '|domain| (sublislis argl |$FormalMapVariableList| catOpList)))))
  (setq nils (loop for x in argl collect nil))
  (setq packageSig (cons packageCategory (cons form nils)))
  (setq |$categoryPredicateList|
    (msubst nameForDollar '$ |$categoryPredicateList|))
  (msubst nameForDollar '$
    (list 'def (cons packageName packageArgl)
      packageSig (cons nil nils) def))))))

```

5.2.12 defun mkEvaluableCategoryForm

```

[qcar p??]
[qcdr p??]
[mkEvaluableCategoryForm p140]
[compOrCroak p496]
[getdatabase p??]
[get p??]

```

```

[mkq p??]
[$Category p??]
[$e p??]
[$EmptyMode p131]
[$CategoryFrame p??]
[$Category p??]
[$CategoryNames p??]
[$e p??]

```

— defun mkEvaluableCategoryForm —

```

(defun |mkEvaluableCategoryForm| (c)
  (let (op arg1 tmp1 x m)
    (declare (special |$Category| |$e| |$EmptyMode| |$CategoryFrame|
                      |$CategoryNames|))
    (if (consp c)
        (progn
          (setq op (qfirst c))
          (setq arg1 (qrest c))
          (cond
            ((eq op '|Join|)
             (cons '|Join|
                   (loop for x in arg1
                        collect (|mkEvaluableCategoryForm| x))))
            ((eq op '|DomainSubstitutionMacro|)
             (|mkEvaluableCategoryForm| (cadr arg1)))
            ((eq op '|mkCategory|) c)
            ((member op |$CategoryNames|)
             (setq tmp1 (|compOrCroak| c |$EmptyMode| |$e|))
             (setq x (car tmp1))
             (setq m (cadr tmp1))
             (setq |$e| (caddr tmp1))
             (when (equal m |$Category|) x))
            ((or (eq (getdatabase op 'constructorkind) '|category|)
                  (|get| op '|isCategory| |$CategoryFrame|))
             (cons op
                   (loop for x in arg1
                        collect (mkq x))))
            (t
             (setq tmp1 (|compOrCroak| c |$EmptyMode| |$e|))
             (setq x (car tmp1))
             (setq m (cadr tmp1))
             (setq |$e| (caddr tmp1))
             (when (equal m |$Category|) x))))
        (mkq c))))

```

5.2.13 defun compDefineCategory2

```

[addBinding p??]
[getArgumentModeOrMoan p154]
[giveFormalParametersValues p135]
[take p??]
[sublis p??]
[compMakeDeclaration p525]
[nequal p??]
[opOf p??]
[optFunctorBody p??]
[compOrCroak p496]
[mkConstructor p168]
[compile p145]
[lisplibWrite p180]
[removeZeroOne p??]
[mkq p??]
[evalAndRwriteLispForm p168]
[eval p??]
[getParentsFor p??]
[computeAncestorsOf p??]
[constructor? p??]
[augLisplibModemapsFromCategory p155]
[$prefix p??]
[$formalArgList p??]
[$definition p??]
[$form p??]
[$op p??]
[$extraParms p??]
[$lisplibCategory p??]
[$FormalMapVariableList p242]
[$libFile p??]
[$TriangleVariableList p??]
[$lisplib p??]
[$formalArgList p??]
[$insideCategoryIfTrue p??]
[$top-level p??]
[$definition p??]
[$form p??]
[$op p??]
[$extraParms p??]
[$functionStats p??]
[$functorStats p??]
[$frontier p??]
[$getDomainCode p??]
[$addForm p??]

```

```

[$lisplibAbbreviation p??]
[$functorForm p??]
[$lisplibAncestors p??]
[$lisplibCategory p??]
[$lisplibParents p??]
[$lisplibModemap p??]
[$lisplibKind p??]
[$lisplibForm p??]
[$domainShell p??]

```

— **defun compDefineCategory2** —

```

(defun |compDefineCategory2|
  (form signature specialCases body mode env |$prefix| |$formalArgList|)
  (declare (special |$prefix| |$formalArgList|) (ignore specialCases))
  (let (|$insideCategoryIfTrue| $TOP_LEVEL |$definition| |$form| |$op|
        |$extraParms| |$functionStats| |$functorStats| |$frontier|
        |$getDomainCode| |$addForm| argl sargl aList signaturep opp formp
        formalBody formals actuals g fun pairlis parSignature parForm modemap)
    (declare (special |$insideCategoryIfTrue| $top_level |$definition|
                      |$form| |$op| |$extraParms| |$functionStats|
                      |$functorStats| |$frontier| |$getDomainCode|
                      |$addForm| |$lisplibAbbreviation| |$functorForm|
                      |$lisplibAncestors| |$lisplibCategory|
                      |$FormalMapVariableList| |$lisplibParents|
                      |$lisplibModemap| |$lisplibKind| |$lisplibForm|
                      $lisplib |$domainShell| |$libFile|
                      |$TriangleVariableList|))
      ; 1. bind global variables
      (setq |$insideCategoryIfTrue| t)
      (setq $top_level nil)
      (setq |$definition| nil)
      (setq |$form| nil)
      (setq |$op| nil)
      (setq |$extraParms| nil)
      ; 1.1 augment e to add declaration $: <form>
      (setq |$definition| form)
      (setq |$op| (car |$definition|))
      (setq argl (cdr |$definition|))
      (setq env (|addBinding| '$ (list (cons '|mode| |$definition|)) env))
      ; 2. obtain signature
      (setq signaturep
        (cons (car signature)
              (loop for a in argl
                    collect (|getArgumentModeOrMoan| a |$definition| env))))
      (setq env (|giveFormalParametersValues| argl env))
      ; 3. replace arguments by $1,..., substitute into body,
      ;    and introduce declarations into environment
      (setq sargl (take (|#| argl) |$TriangleVariableList|))

```

```

(setq |$form| (cons |$op| sargl))
(setq |$functorForm| |$form|)
(setq |$formalArgList| (append sargl |$formalArgList|))
(setq aList (loop for a in argl for sa in sargl collect (cons a sa)))
(setq formalBody (sublis aList body))
(setq signaturep (sublis aList signaturep))
; Begin lines for category default definitions
(setq |$functionStats| (list 0 0))
(setq |$functorStats| (list 0 0))
(setq |$frontier| 0)
(setq |$getDomainCode| nil)
(setq |$addForm| nil)
(loop for x in sargl for r in (rest signaturep)
  do (setq env (third (|compMakeDeclaration| (list '[:| x r] mode env))))))
; 4. compile body in environment of %type declarations for arguments
(setq opp |$op|)
(when (and (nequal (|opOf| formalBody) '|Join|)
           (nequal (|opOf| formalBody) '|mkCategory|))
  (setq formalBody (list '|Join| formalBody)))
(setq body
  (|optFunctorBody| (car (|compOrCroak| formalBody (car signaturep) env))))
(when |$extraParms|
  (setq actuals nil)
  (setq formals nil)
  (loop for u in |$extraParms| do
    (setq formals (cons (car u) formals))
    (setq actuals (cons (mkq (cdr u)) actuals))))
(setq body
  (list '|sublisV| (list 'pair (list 'quote formals) (cons 'list actuals))
    body)))
; always subst for args after extraparms
(when argl
  (setq body
    (list '|sublisV|
      (list 'pair
        (list 'quote sargl)
        (cons 'list (loop for u in sargl collect (list '|devaluate| u))))
      body)))
(setq body
  (list 'prog1 (list 'let (setq g (gensym)) body)
    (list 'setelt g 0 (|mkConstructor| |$form|))))
(setq fun (|compile| (list opp (list 'lam sargl body))))
; 5. give operator a 'modemap property
(setq pairlis
  (loop for a in argl for v in |$FormalMapVariableList|
    collect (cons a v)))
(setq parSignature (sublis pairlis signaturep))
(setq parForm (sublis pairlis form))
(|lisplibWrite| "compilerInfo"
  (|removeZeroOne|

```

```

    (list 'setq '|$CategoryFrame|
      (list '|put| (list 'quote opp) ''|isCategory| t
        (list '|addModemap| (mkq opp) (mkq parForm)
          (mkq parSignature) t (mkq fun) '|$CategoryFrame|))))
    |$libFile|)
  (unless sargl
    (|evalAndRwriteLispForm| 'niladic
      (list 'makeprop (list 'quote opp) ''niladic t)))
;; 6 put modemaps into InteractiveModemapFrame
(setq |$domainShell| (|eval| (cons opp (mapcar 'mkq sargl))))
(setq |$lisplibCategory| formalBody)
(when $lisplib
  (setq |$lisplibForm| form)
  (setq |$lisplibKind| '|category|)
  (setq modemap (list (cons parForm parSignature) (list t opp)))
  (setq |$lisplibModemap| modemap)
  (setq |$lisplibParents|
    (|getParentsFor| |$op| |$FormalMapVariableList| |$lisplibCategory|))
  (setq |$lisplibAncestors| (|computeAncestorsOf| |$form| nil))
  (setq |$lisplibAbbreviation| (|constructor?| |$op|))
  (setq formp (cons opp sargl))
  (|augLisplibModemapsFromCategory| formp formalBody signaturep))
(list fun '(|Category|) env)))

```

5.2.14 defun compile

```

[member p??]
[getmode p??]
[qcar p??]
[qcdr p??]
[get p??]
[modeEqual p348]
[userError p??]
[encodeItem p150]
[strconc p??]
[nequal p??]
[kar p??]
[encodeFunctionName p148]
[splitEncodedFunctionName p149]
[sayBrightly p??]
[optimizeFunctionDef p203]
[putInLocalDomainReferences p154]
[constructMacro p151]
[spadCompileOrSetq p151]

```



```

[elapsedTime p??]
[addStats p??]
[printStats p??]
[$functionStats p??]
[$macroIfTrue p??]
[$doNotCompileJustPrint p??]
[$insideCapsuleFunctionIfTrue p??]
[$saveableItems p??]
[$lisplibItemsAlreadyThere p??]
[$splitUpItemsAlreadyThere p??]
[$lisplib p??]
[$compileOnlyCertainItems p??]
[$functorForm p??]
[$signatureOfForm p??]
[$suffix p??]
[$prefix p??]
[$signatureOfForm p??]
[$e p??]
[$functionStats p??]
[$savableItems p??]
[$suffix p??]

```

— defun compile —

```

(defun |compile| (u)
  (labels (
    (isLocalFunction (op)
      (let (tmp1)
        (declare (special |$e| |$formalArgList|))
        (and (null (|member| op |$formalArgList|))
          (progn
            (setq tmp1 (|getmode| op |$e|))
            (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)))))))
    (let (op lamExpr DC sig sel opexport opmodes opp parts s tt unew
      optimizedBody stuffToCompile result functionStats)
      (declare (special |$functionStats| |$macroIfTrue| |$doNotCompileJustPrint|
        |$insideCapsuleFunctionIfTrue| |$saveableItems| |$e|
        |$lisplibItemsAlreadyThere| |$splitUpItemsAlreadyThere|
        |$compileOnlyCertainItems| $LISPLIB |$suffix|
        |$signatureOfForm| |$functorForm| |$prefix|
        |$savableItems|))
        (setq op (first u))
        (setq lamExpr (second u))
        (when |$suffix|
          (setq |$suffix| (1+ |$suffix|))
          (setq opp
            (progn
              (setq opexport nil)

```

```

(setq opmodes
(loop for item in (|get| op '|modemap| |$e|)
do
  (setq dc (caar item))
  (setq sig (cdar item))
  (setq sel (cadadr item))
  when (and (eq dc '$)
            (setq opexport t)
            (let ((result t))
              (loop for x in sig for y in |$signatureOfForm|
                do (setq result (|modeEqual| x y)))
              result))
    collect sel))
(cond
((isLocalFunction op)
 (when opexport
  (|userError| (list '|%b| op '|%d| " is local and exported"))
  (intern (strconc (|encodeItem| |$prefix|) ";" (|encodeItem| op))))
(t
 (|encodeFunctionName| op |$functorForm| |$signatureOfForm|
  '|;| |$suffix|))))
(setq u (list opp lamExpr))
(when (and $lisplib |$compileOnlyCertainItems|)
 (setq parts (|splitEncodedFunctionName| (elt u 0) '|;|))
 (cond
 ((eq parts '|inner|)
  (setq |$savableItems| (cons (elt u 0) |$savableItems|)))
 (t
  (setq unew nil)
  (loop for item in |$splitUpItemsAlreadyThere|
  do
    (setq s (first item))
    (setq tt (second item))
    (when
      (and (equal (elt parts 0) (elt s 0))
           (equal (elt parts 1) (elt s 1))
           (equal (elt parts 2) (elt s 2)))
      (setq unew tt)))
  (cond
  ((null unew)
   (|sayBrightly| (list " Error: Item did not previously exist"))
   (|sayBrightly| (cons " Item not saved: " (|bright| (elt u 0))))
   (|sayBrightly|
    (list " What's there is: " |$lisplibItemsAlreadyThere|))
   nil)
  (t
   (|sayBrightly| (list " Renaming " (elt u 0) " as " unew))
   (setq u (cons unew (cdr u)))
   (setq |$savableItems| (cons unew |$saveableItems|))))))
(setq optimizedBody (|optimizeFunctionDef| u))

```

```

(setq stuffToCompile
  (if |$insideCapsuleFunctionIfTrue|
    (|putInLocalDomainReferences| optimizedBody)
    optimizedBody))
(cond
  ((eq |$doNotCompileJustPrint| t)
    (prettyprint stuffToCompile)
    opp)
  (|$macroIfTrue| (|constructMacro| stuffToCompile))
  (t
    (setq result (|spadCompileOrSetq| stuffToCompile))
    (setq functionStats (list 0 (|elapsedTime|)))
    (setq |$functionStats| (|addStats| |$functionStats| functionStats))
    (|printStats| functionStats)
    result))))

```

5.2.15 defun encodeFunctionName

Code for encoding function names inside package or domain [msubst p??]

```

[mkRepetitionAssoc p149]
[encodeItem p150]
[stringimage p??]
[internal p??]
[getAbbreviation p277]
[length p??]
[$lisplib p??]
[$lisplibSignatureAlist p??]
[$lisplibSignatureAlist p??]

```

— defun encodeFunctionName —

```

(defun |encodeFunctionName| (fun package signature sep count)
  (let (packageName arglist signaturep reducedSig n x encodedSig encodedName)
    (declare (special |$lisplibSignatureAlist| $lisplib))
    (setq packageName (car package))
    (setq arglist (cdr package))
    (setq signaturep (msubst '$ package signature))
    (setq reducedSig
      (|mkRepetitionAssoc| (append (cdr signaturep) (list (car signaturep)))))
    (setq encodedSig
      (let ((result ""))
        (loop for item in reducedSig
          do
            (setq n (car item))
            (setq x (cdr item))

```

```

      (setq result
        (strconc result
          (if (eql n 1)
              (|encodeItem| x)
              (strconc (stringimage n) (|encodeItem| x))))))
      result))
(setq encodedName
  (internl (|getAbbreviation| packageName (|#| arglist))
    '|;| (|encodeItem| fun) '|;| encodedSig sep (stringimage count)))
(when $lisplib
  (setq |$lisplibSignatureAlist|
    (cons (cons encodedName signaturep) |$lisplibSignatureAlist|)))
encodedName))

```

5.2.16 defun mkRepititionAssoc

```

[qcar p??]
[qcdr p??]

```

— defun mkRepititionAssoc —

```

(defun |mkRepititionAssoc| (z)
  (labels (
    (mkRepfun (z n)
      (cond
        ((null z) nil)
        ((and (consp z) (eq (qrest z) nil) (list (cons n (qfirst z)))))
        ((and (consp z) (consp (qrest z)) (equal (qsecond z) (qfirst z)))
          (mkRepfun (cdr z) (1+ n)))
        (t (cons (cons n (car z)) (mkRepfun (cdr z) 1))))))
    (mkRepfun z 1)))

```

5.2.17 defun splitEncodedFunctionName

```

[stringimage p??]
[strpos p??]

```

— defun splitEncodedFunctionName —

```

(defun |splitEncodedFunctionName| (encodedName sep)
  (let (sep0 p1 p2 p3 s1 s2 s3 s4)

```

```

; sep0 is the separator used in "encodeFunctionName".
(setq sep0 ";")
(unless (stringp encodedName) (setq encodedName (stringimage encodedName)))
(cond
  ((null (setq p1 (strpos sep0 encodedName 0 "*"))) nil)
  ; This is picked up in compile for inner functions in partial compilation
  ((null (setq p2 (strpos sep0 encodedName (1+ p1) "*"))) '|inner|)
  ((null (setq p3 (strpos sep encodedName (1+ p2) "*"))) nil)
  (t
   (setq s1 (substring encodedName 0 p1))
   (setq s2 (substring encodedName (1+ p1) (- p2 p1 1)))
   (setq s3 (substring encodedName (1+ p2) (- p3 p2 1)))
   (setq s4 (substring encodedName (1+ p3) nil))
   (list s1 s2 s3 s4))))

```

5.2.18 defun encodeItem

```

[getCaps p150]
[identp p??]
[qcar p??]
[pname p??]
[stringimage p??]

```

— defun encodeItem —

```

(defun |encodeItem| (x)
  (cond
    ((consp x) (|getCaps| (qfirst x)))
    ((identp x) (pname x))
    (t (stringimage x))))

```

5.2.19 defun getCaps

```

[stringimage p??]
[maxindex p??]
[l-case p??]
[strconc p??]

```

— defun getCaps —

```

(defun |getCaps| (x)

```

```

(let (s c clist tmp1)
  (setq s (stringimage x))
  (setq clist
    (loop for i from 0 to (maxindex s)
      when (upper-case-p (setq c (elt s i)))
      collect c))
  (cond
    ((null clist) "_")
    (t
     (setq tmp1
       (cons (first clist) (loop for u in (rest clist) collect (l-case u))))
     (let ((result ""))
       (loop for u in tmp1
         do (setq result (strconc result u)))
       result)))))

```

5.2.20 defun constructMacro

constructMacro (form is [nam,[lam,vl,body]]) [stackSemanticError p??]
 [identp p??]

— defun constructMacro —

```

(defun |constructMacro| (form)
  (let (vl body)
    (setq vl (cadadr form))
    (setq body (car (cddadr form)))
    (cond
      ((null (let ((result t))
        (loop for x in vl
          do (setq result (and result (atom x))))
        result))
        (|stackSemanticError| (list '|illegal parameters for macro: | vl) nil))
      (t
       (list 'xlam (loop for x in vl when (identp x) collect x) body)))))

```

5.2.21 defun spadCompileOrSetq

[qcar p??]
 [qcdr p??]
 [contained p??]

```

[sayBrightly p??]
[bright p??]
[LAM,EVALANDFILEACTQ p??]
[mkq p??]
[comp p497]
[compileConstructor p153]
[$insideCapsuleFunctionIfTrue p??]

```

— **defun spadCompileOrSetq** —

```

(defun |spadCompileOrSetq| (form)
  (let (nam lam vl body namp tmp1 e vlp macform)
    (declare (special |$insideCapsuleFunctionIfTrue|))
    (setq nam (car form))
    (setq lam (caadr form))
    (setq vl (cadadr form))
    (setq body (car (cddadr form)))
    (cond
      ((and (consp vl) (progn (setq tmp1 (reverse vl)) t)
        (consp tmp1)
        (progn
          (setq e (qfirst tmp1))
          (setq vlp (qrest tmp1))
          t)
        (progn (setq vlp (nreverse vlp)) t)
        (consp body)
        (progn (setq namp (qfirst body)) t)
        (equal (qrest body) vlp))
      (|LAM,EVALANDFILEACTQ|
       (list 'put (mkq nam) (mkq '|SPADreplace|) (mkq namp)))
      (|sayBrightly|
       (cons " " (append (|bright| nam)
        (cons "is replaced by" (|bright| namp))))))
    ((and (or (atom body)
      (let ((result t))
        (loop for x in body
          do (setq result (and result (atom x))))
        result))
      (consp vl)
      (progn (setq tmp1 (reverse vl)) t)
      (consp tmp1)
      (progn
        (setq e (qfirst tmp1))
        (setq vlp (qrest tmp1))
        t)
      (progn (setq vlp (nreverse vlp)) t)
      (null (contained e body)))
    (setq macform (list 'xlam vlp body))
    (|LAM,EVALANDFILEACTQ|

```

```

      (list 'put (mkq nam) (mkq '|SPADreplace|) (mkq macform)))
    (|sayBrightly| (cons "      " (append (|bright| nam)
      (cons "is replaced by" (|bright| body))))))
    (t nil))
  (if |$insideCapsuleFunctionIfTrue|
    (car (comp (list form)))
    (|compileConstructor| form)))

```

5.2.22 defun compileConstructor

```

[compileConstructor1 p153]
[clearClams p??]

```

— **defun compileConstructor** —

```

(defun |compileConstructor| (form)
  (let (u)
    (setq u (|compileConstructor1| form))
    (|clearClams|
     u))

```

5.2.23 defun compileConstructor1

```

[getdatabase p??]
[compAndDefine p??]
[comp p497]
[clearConstructorCache p??]
[$mutableDomain p??]
[$ConstructorCache p??]
[$clamList p??]
[$clamList p??]

```

— **defun compileConstructor1** —

```

(defun |compileConstructor1| (form)
  (let (|$clamList| fn key vl body1 lambdaOrSlam compForm u)
    (declare (special |$clamList| |$ConstructorCache| |$mutableDomain|))
    (setq fn (car form))
    (setq key (caadr form))
    (setq vl (cadadr form))

```



```

(setq body1 (cddadr form))
(setq |$clamList| nil)
(setq lambdaOrSlam
  (cond
    ((eq (getdatabase fn 'constructorkind) '|category|) 'spadslam)
    (|$mutableDomain| 'lambda)
    (t
      (setq |$clamList|
        (cons (list fn '|$ConstructorCache| '|domainEqualList| '|count|)
              |$clamList|)))
      'lambda)))
(setq compForm (list (list fn (cons lambdaOrSlam (cons vl body1)))))
(if (eq (getdatabase fn 'constructorkind) '|category|)
  (setq u (|compAndDefine| compForm))
  (setq u (comp compForm)))
(|clearConstructorCache| fn)
(car u))

```

5.2.24 defun putInLocalDomainReferences

```

[NRTputInTail p??]
[$QuickCode p??]
[$elt p291]

```

— defun putInLocalDomainReferences —

```

(defun |putInLocalDomainReferences| (def)
  (let (|$elt| opName lam varl body)
    (declare (special |$elt| |$QuickCode|))
    (setq opName (car def))
    (setq lam (caadr def))
    (setq varl (cadadr def))
    (setq body (car (cddadr def)))
    (setq |$elt| (if |$QuickCode| 'qrefelt 'elt))
    (|NRTputInTail| (cddadr def))
    def))

```

5.2.25 defun getArgumentModeOrMoan

```

[getArgumentMode p291]
[stackSemanticError p??]

```

— defun getArgumentModeOrMoan —

```
(defun |getArgumentModeOrMoan| (x form env)
  (or (|getArgumentMode| x env)
      (|stackSemanticError|
       (list '|argument| x '| of | form '| is not declared|) nil)))
```

—————

5.2.26 defun augLisplibModemapsFromCategory

```
[sublis p??]
[mkAlistOfExplicitCategoryOps p156]
[isCategoryForm p??]
[lassoc p??]
[member p??]
[mkpf p??]
[interactiveModemapForm p158]
[$lisplibModemapAlist p??]
[$EmptyEnvironment p??]
[$domainShell p??]
[$PatternVariableList p??]
[$lisplibModemapAlist p??]
```

— defun augLisplibModemapsFromCategory —

```
(defun |augLisplibModemapsFromCategory| (form body signature)
  (let (argl sl opAlist nonCategorySigAlist domainList catPredList op sig
        pred sel predp modemap)
    (declare (special |$lisplibModemapAlist| |$EmptyEnvironment|
                      |$domainShell| |$PatternVariableList|))
    (setq op (car form))
    (setq argl (cdr form))
    (setq sl
      (cons (cons '$ '*1)
            (loop for a in argl for p in (rest |$PatternVariableList|)
                  collect (cons a p))))
    (setq form (sublis sl form))
    (setq body (sublis sl body))
    (setq signature (sublis sl signature))
    (when (setq opAlist (sublis sl (elt |$domainShell| 1)))
      (setq nonCategorySigAlist
        (|mkAlistOfExplicitCategoryOps| (msubst '*1 '$ body)))
      (setq domainList
        (loop for a in (rest form) for m in (rest signature)
```

```

      when (|isCategoryForm| m |$EmptyEnvironment|)
        collect (list a m)))
(setq catPredList
(loop for u in (cons (list '*1 form) domainList)
  collect (cons '|ofCategory| u)))
(loop for entry in opAlist
  when (|member| (caddr entry) (lassoc (caar entry) nonCategorySigAlist))
  do
    (setq op (caar entry))
    (setq sig (caddr entry))
    (setq pred (caddr entry))
    (setq sel (caddr entry))
    (setq predp (mkpf (cons pred catPredList) 'and))
    (setq modemap (list (cons '*1 sig) (list predp sel)))
    (setq |$lisplibModemapAlist|
      (cons (cons op (|interactiveModemapForm| modemap))
        |$lisplibModemapAlist|))))))

```

5.2.27 defun mkAlistOfExplicitCategoryOps

```

[qcar p??]
[qcdr p??]
[keyedSystemError p??]
[union p??]
[mkAlistOfExplicitCategoryOps p156]
[flattenSignatureList p158]
[nreverse0 p??]
[remdup p??]
[assocleft p??]
[isCategoryForm p??]
[$e p??]

```

— defun mkAlistOfExplicitCategoryOps —

```

(defun |mkAlistOfExplicitCategoryOps| (target)
  (labels (
    (atomizeOp (op)
      (cond
        ((atom op) op)
        ((and (consp op) (eq (qrest op) nil)) (qfirst op))
        (t (|keyedSystemError| 'S2GE0016
          (list "mkAlistOfExplicitCategoryOps" "bad signature")))))
    (fn (op u)
      (if (and (consp u) (consp (qfirst u)))
        (if (equal (qcaar u) op)

```

```

      (cons (qcdar u) (fn op (qrest u)))
      (fn op (qrest u))))))
(let (z tmp1 op sig u opList)
(declare (special |$e|))
(when (and (consp target) (eq (qfirst target) '|add|) (consp (qrest target)))
  (setq target (second target)))
(cond
  ((and (consp target) (eq (qfirst target) '|Join|))
   (setq z (qrest target))
   (PROG (tmp1)
    (RETURN
     (DO ((G167566 z (CDR G167566)) (cat nil))
        ((OR (ATOM G167566) (PROGN (setq cat (CAR G167566)) nil))
         tmp1)
      (setq tmp1 (|union| tmp1 (|mkAlistOfExplicitCategoryOps| cat)))))))
  ((and (consp target) (eq (qfirst target) 'category)
   (progn
    (setq tmp1 (qrest target))
    (and (consp tmp1)
         (progn (setq z (qrest tmp1)) t))))
   (setq z (|flattenSignatureList| (cons 'progn z)))
   (setq u
    (prog (G167577)
     (return
      (do ((G167583 z (cdr G167583)) (x nil))
          ((or (atom G167583) (nreverse0 G167577))
           (setq x (car G167583))
           (cond
            ((and (consp x) (eq (qfirst x) 'signature) (consp (qrest x))
             (consp (qcddr x)))
             (setq op (qsecond x))
             (setq sig (qthird x))
             (setq G167577 (cons (cons (atomizeOp op) sig) G167577)))))))
      (setq opList (remdup (assocleft u)))
      (prog (G167593)
       (return
        (do ((G167598 opList (cdr G167598)) (x nil))
            ((or (atom G167598) (nreverse0 G167593))
             (setq x (car G167598))
             (setq G167593 (cons (cons x (fn x u) G167593))))))
        (|isCategoryForm| target |$e|) nil)
       t
      (|keyedSystemError| 'S2GE0016
       (list "mkAlistOfExplicitCategoryOps" "bad signature"))))))))

```

5.2.28 defun flattenSignatureList

[qcar p??]
 [qcdr p??]
 [flattenSignatureList p158]

— **defun flattenSignatureList** —

```
(defun |flattenSignatureList| (x)
  (let (zz)
    (cond
      ((atom x) nil)
      ((and (consp x) (eq (qfirst x) 'signature)) (list x))
      ((and (consp x) (eq (qfirst x) 'if) (consp (qrest x))
            (consp (qcddr x)) (consp (qcdddr x))
            (eq (qcdddr x) nil))
       (append (|flattenSignatureList| (third x))
                (|flattenSignatureList| (fourth x))))
      ((and (consp x) (eq (qfirst x) 'progn))
       (loop for x in (qrest x)
             do
              (if (and (consp x) (eq (qfirst x) 'signature))
                  (setq zz (cons x zz))
                  (setq zz (append (|flattenSignatureList| x) zz))))
       zz)
      (t nil))))
```

—————

5.2.29 defun interactiveModemapForm

Create modemap form for use by the interpreter. This function replaces all specific domains mentioned in the modemap with pattern variables, and predicates [qcar p??]

[qcdr p??]
 [nequal p??]
 [replaceVars p159]
 [modemapPattern p167]
 [substVars p166]
 [fixUpPredicate p160]
 [\$PatternVariableList p??]
 [\$FormalMapVariableList p242]

— **defun interactiveModemapForm** —

```
(defun |interactiveModemapForm| (mm)
  (labels (
```

```

(fn (x)
  (if (and (consp x) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil)
          (nequal (qfirst x) '|isFreeFunction|)
          (atom (qthird x))))
      (list (first x) (second x) (list (third x)))
      x)))
(let (pattern dc sig mmpat patternAlist partial patvars
      domainPredicateList tmp1 pred dependList cond)
(declare (special |$PatternVariableList| |$FormalMapVariableList|))
(setq mm
  (|replaceVars| (copy mm) |$PatternVariableList| |$FormalMapVariableList|))
(setq pattern (car mm))
(setq dc (caar mm))
(setq sig (cdar mm))
(setq pred (cadr mm))
(setq pred
  (prog ()
    (return
      (do ((x pred (cdr x)) (result nil))
          ((atom x) (nreverse0 result))
          (setq result (cons (fn (car x)) result)))))))
(setq tmp1 (|modemapPattern| pattern sig))
(setq mmpat (car tmp1))
(setq patternAlist (cadr tmp1))
(setq partial (caddr tmp1))
(setq patvars (cadddr tmp1))
(setq tmp1 (|substVars| pred patternAlist patvars))
(setq pred (car tmp1))
(setq domainPredicateList (cadr tmp1))
(setq tmp1 (|fixUpPredicate| pred domainPredicateList partial (cdr mmpat)))
(setq pred (car tmp1))
(setq dependList (cdr tmp1))
(setq cond (car pred))
(list mmpat cond)))

```

5.2.30 defun replaceVars

Replace every identifier in oldvars with the corresponding identifier in newvars in the expression x [msubst p??]

— defun replaceVars —

```

(defun |replaceVars| (x oldvars newvars)
  (loop for old in oldvars for new in newvars

```

```

do (setq x (msubst new old x)))
x)

```

5.2.31 defun fixUpPredicate

```

[qcar p??]
[qcdr p??]
[length p??]
[orderPredicateItems p161]
[moveORsOutside p165]

```

— defun fixUpPredicate —

```

(defun |fixUpPredicate| (predClause domainPreds partial sig)
  (let (predicate fn skip predicates tmp1 dependList pred)
    (setq predicate (car predClause))
    (setq fn (cadr predClause))
    (setq skip (cddr predClause))
    (cond
      ((eq (car predicate) 'and)
        (setq predicates (append domainPreds (cdr predicate))))
      ((nequal predicate (mkq t))
        (setq predicates (cons predicate domainPreds)))
      (t
        (setq predicates (or domainPreds (list predicate)))))
    (cond
      ((> (|#| predicates) 1)
        (setq pred (cons 'and predicates))
        (setq tmp1 (|orderPredicateItems| pred sig skip))
        (setq pred (car tmp1))
        (setq dependlist (cdr tmp1))
        tmp1)
      (t
        (setq pred (|orderPredicateItems| (car predicates) sig skip))
        (setq dependList
          (when (and (consp pred) (eq (qfirst pred) '|isDomain|)
                    (consp (qrest pred)) (consp (qcddr pred))
                    (eq (qcdddr pred) nil)
                    (consp (qthird pred))
                    (eq (qcdaddr pred) nil))
            (list (second pred)))))
        (setq pred (|moveORsOutside| pred))
        (when partial (setq pred (cons '|partial| pred)))
        (cons (cons pred (cons fn skip)) dependList)))

```

5.2.32 defun orderPredicateItems

```
[qcar p??]
[qcdr p??]
[signatureTran p161]
[orderPredTran p162]
```

— defun orderPredicateItems —

```
(defun |orderPredicateItems| (pred1 sig skip)
  (let (pred)
    (setq pred (|signatureTran| pred1))
    (if (and (consp pred) (eq (qfirst pred) 'and))
        (|orderPredTran| (qrest pred) sig skip)
        pred)))
```

5.2.33 defun signatureTran

```
[signatureTran p161]
[isCategoryForm p??]
[$e p??]
```

— defun signatureTran —

```
(defun |signatureTran| (pred)
  (declare (special |$e|))
  (cond
    ((atom pred) pred)
    ((and (consp pred) (eq (qfirst pred) '|has|) (CONSP (qrest pred))
          (consp (qcddr pred))
          (eq (qcdddr pred) nil)
          (|isCategoryForm| (third pred) |$e|))
     (list '|ofCategory| (second pred) (third pred)))
    (t
     (loop for p in pred
           collect (|signatureTran| p)))))
```

5.2.34 defun orderPredTran

```
[qcar p??]
[qcdr p??]
[member p??]
[delete p??]
[unionq p??]
[listOfPatternIds p??]
[intersectionq p??]
[setdifference p??]
[insertWOC p??]
[isDomainSubst p164]
```

— defun orderPredTran —

```
(defun |orderPredTran| (oldList sig skip)
  (let (lastDependList somethingDone lastPreds indepvl depvl dependList
        noldList x ids fullDependList newList answer)
    ; --(1) make two kinds of predicates appear last:
    ; ----- (op *target ..) when *target does not appear later in sig
    ; ----- (isDomain *1 ..)
    (SEQ
      (loop for pred in oldList
        do (cond
          ((or (and (consp pred) (consp (qrest pred))
                    (consp (qcddr pred))
                    (eq (qcddr pred) nil)
                    (member (qfirst pred) '(|isDomain| |ofCategory|))
                    (equal (qsecond pred) (car sig))
                    (null (|member| (qsecond pred) (cdr sig)))))
              (and (null skip) (consp pred) (eq (qfirst pred) '(|isDomain|
                    (consp (qrest pred)) (consp (qcddr pred))
                    (eq (qcddr pred) nil)
                    (equal (qsecond pred) '*1))))
              (setq oldList (|delete| pred oldList))
              (setq lastPreds (cons pred lastPreds))))))
    ; --(2a) lastDependList=list of all variables that lastPred forms depend upon
    (setq lastDependList
      (let (result)
        (loop for x in lastPreds
          do (setq result (unionq result (|listOfPatternIds| x))))
        result))
    ; --(2b) dependList=list of all variables that isDom/ofCat forms depend upon
    (setq dependList
      (let (result)
        (loop for x in oldList
          do (when
              (and (consp x)
                   (or (eq (qfirst x) '(|isDomain|) (eq (qfirst x) '(|ofCategory|))
```

```

        (consp (qrest x)) (consp (qcddr x))
        (eq (qcddr x) nil))
      (setq result (unionq result (|listOfPatternIds| (third x))))))
  result))
; --(3a) newList= list of ofCat/isDom entries that don't depend on
(loop for x in oldList
  do
    (cond
      ((and (consp x)
        (or (eq (qfirst x) '|ofCategory|) (eq (qfirst x) '|isDomain|))
        (consp (qrest x)) (consp (qcddr x))
        (eq (qcddr x) nil))
        (setq indepvl (|listOfPatternIds| (second x)))
        (setq depvl (|listOfPatternIds| (third x))))
      (t
        (setq indepvl (|listOfPatternIds| x))
        (setq depvl nil)))
    (when
      (and (null (intersectionq indepvl dependList))
        (intersectionq indepvl lastDependList))
      (setq somethingDone t)
      (setq lastPreds (append lastPreds (list x)))
      (setq oldList (|delete| x oldList))))
; --(3b) newList= list of ofCat/isDom entries that don't depend on
(loop while oldList do
  (loop for x in oldList do
    (cond
      ((and (consp x)
        (or (eq (qfirst x) '|ofCategory|) (eq (qfirst x) '|isDomain|))
        (consp (qrest x))
        (consp (qcddr x)) (eq (qcddr x) nil))
        (setq indepvl (|listOfPatternIds| (second x)))
        (setq depvl (|listOfPatternIds| (third x))))
      (t
        (setq indepvl (|listOfPatternIds| x))
        (setq depvl nil)))
    (when (null (intersectionq indepvl dependList))
      (setq dependList (SETDIFFERENCE dependList depvl))
      (setq newList (APPEND newList (list x))))))
; --(4) noldList= what is left over
(cond
  ((equal (setq noldList (setdifference oldList newList)) oldList)
    (setq newList (APPEND newList oldList))
    (return nil))
  (t
    (setq oldList noldList)))
(loop for pred in newList do
  (when
    (and (consp pred)
      (or (eq (qfirst pred) '|isDomain|) (eq (qfirst x) '|ofCategory|))

```

```

      (consp (qrest pred))
      (consp (qcddr pred))
      (eq (qcdddr pred) nil))
    (setq ids (|listOfPatternIds| (third pred)))
    (when
      (let (result)
        (loop for id in ids do
          (setq result (and result (|member| id fullDependList))))
        result)
      (setq fullDependList (|insertWOC| (second pred) fullDependList)))
    (setq fullDependList (unionq fullDependList ids)))
  (setq newList (append newList lastPreds))
  (setq newList (|isDomainSubst| newList))
  (setq answer
    (cons (cons 'and newList) (intersectionq fullDependList sig))))))

```

5.2.35 defun isDomainSubst

— defun isDomainSubst —

```

(defun |isDomainSubst| (u)
  (labels (
    (findSub (x alist)
      (cond
        ((null alist) nil)
        ((and (consp alist) (consp (qfirst alist))
          (eq (qcaar alist) '|isDomain|)
          (consp (qcddar alist))
          (consp (qcdddr alist))
          (eq (qcdddr alist) nil)
          (equal x (cadar alist)))
          (caddar alist))
        (t (findSub x (cdr alist)))))
    (fn (x alist)
      (let (s)
        (declare (special |$PatternVariableList|))
        (if (atom x)
          (if
            (and (identp x)
              (member x |$PatternVariableList|)
              (setq s (findSub x alist)))
            s
            x)
          (cons (car x)
            (loop for y in (cdr x)
              do (findSub y alist)))))))

```

```

      collect (fn y alist))))))
(let (head tail nhead)
  (if (consp u)
      (progn
        (setq head (qfirst u))
        (setq tail (qrest u))
        (setq nhead
          (cond
            ((and (consp head) (eq (qfirst head) '|isDomain|)
                  (consp (qrest head)) (consp (qcddr head))
                  (eq (qcddr head) nil))
             (list '|isDomain| (second head)
                   (fn (third head) tail)))
            (t head)))
        (cons nhead (|isDomainSubst| (cdr u))))
      u))))

```

5.2.36 defun moveORsOutside

[moveORsOutside p165]

— defun moveORsOutside —

```

(defun |moveORsOutside| (p)
  (let (q x)
    (cond
      ((and (consp p) (eq (qfirst p) 'and))
       (setq q
         (prog (G167169)
           (return
            (do ((G167174 (cdr p) (cdr G167174)) (|r| nil))
              ((or (atom G167174)) (nreverse0 G167169))
              (setq |r| (CAR G167174))
              (setq G167169 (cons (|moveORsOutside| |r|) G167169))))))
       (cond
         ((setq x
           (let (tmp1)
             (loop for r in q
               when (and (consp r) (eq (qfirst r) 'or))
               do (setq tmp1 (or tmp1 r)))
             tmp1))
          (|moveORsOutside|
           (cons 'or
             (let (tmp1)
               (loop for tt in (cdr x)
                 do (setq tmp1 (cons (cons 'and (msubst tt x q)) tmp1))))

```

```

      (nreverse0 tmp1))))))
    (t (cons 'and q))))
  (t p))))

(defun |moveORsOutside| (p)
  (let (q s x tmp1)
    (cond
      ((and (consp p) (eq (qfirst p) 'and))
       (setq q (loop for r in (qrest p) collect (|moveORsOutside| r)))
       (setq tmp1
        (loop for r in q
          when (and (consp r) (eq (qrest r) 'or))
            collect r))
       (setq x (mapcar #'(lambda (a b) (or a b)) tmp1))
       (if x
        (|moveORsOutside|
         (cons 'or
          (loop for tt in (cdr x)
            collect (cons 'and (msubst tt x q)))))
        (cons 'and q))))
    ('t p))))

```

5.2.37 defun substVars

Make pattern variable substitutions. [msubst p??]
 [nsubst p??]
 [contained p??]
 [\$FormalMapVariableList p242]

— defun substVars —

```

(defun |substVars| (pred patternAlist patternVarList)
  (let (patVar value everything replacementVar domainPredicates)
    (declare (special |$FormalMapVariableList|))
    (setq domainPredicates NIL)
    (maplist
     #'(lambda (x)
        (setq patVar (caar x))
        (setq value (cdar x))
        (setq pred (msubst patVar value pred))
        (setq patternAlist (|nsubst| patVar value patternAlist))
        (setq domainPredicates (msubst patVar value domainPredicates))
        (unless (member value |$FormalMapVariableList|)
          (setq domainPredicates
           (cons (list 'isDomain patVar value) domainPredicates))))

```

```

    patternAlist)
  (setq everything (list pred patternAlist domainPredicates))
  (dolist (|var| |$FormalMapVariableList|)
    (cond
      ((contained |var| everything)
        (setq replacementVar (car patternVarList))
        (setq patternVarList (cdr patternVarList))
        (setq pred (msubst replacementVar |var| pred))
        (setq domainPredicates
          (msubst replacementVar |var| domainPredicates))))
    (list pred domainPredicates)))

```

5.2.38 defun modemapPattern

```

[qcar p??]
[qcdr p??]
[rassoc p??]
[$PatternVariableList p??]

```

— defun modemapPattern —

```

(defun |modemapPattern| (mmPattern sig)
  (let (partial patvar patvars mmpat patternAlist)
    (declare (special |$PatternVariableList|))
    (setq patternAlist nil)
    (setq mmpat nil)
    (setq patvars |$PatternVariableList|)
    (setq partial nil)
    (maplist
      #'(lambda (xTails)
        (let ((x (car xTails)))
          (when (and (consp x) (eq (qfirst x) '|Union|)
                     (consp (qrest x)) (consp (qcddr x))
                     (eq (qcddr x) nil)
                     (equal (third x) "failed")
                     (equal xTails sig))
            (setq x (second x))
            (setq partial t))
          (setq patvar (|rassoc| x patternAlist))
          (cond
            ((null (null patvar))
              (setq mmpat (cons patvar mmpat)))
            (t
              (setq patvar (car patvars))
              (setq patvars (cdr patvars))

```

```

      (setq mmpat (cons patvar mmpat))
      (setq patternAlist (cons (cons patvar x) patternAlist))))))
mmPattern)
(list (nreverse mmpat) patternAlist partial patvars)))

```

5.2.39 defun evalAndRwriteLispForm

[eval p??]
[rwriteLispForm p168]

— defun evalAndRwriteLispForm —

```

(defun |evalAndRwriteLispForm| (key form)
  (|eval| form)
  (|rwriteLispForm| key form))

```

5.2.40 defun rwriteLispForm

[\$libFile p??]
[\$lisplib p??]

— defun rwriteLispForm —

```

(defun |rwriteLispForm| (key form)
  (declare (special |$libFile| $lisplib))
  (when $lisplib
    (|rwrite| key form |$libFile|)
    (|LAM,FILEACTQ| key form)))

```

5.2.41 defun mkConstructor

[mkConstructor p168]

— defun mkConstructor —

```

(defun |mkConstructor| (form)

```

```
(cond
  ((atom form) (list '|devaluate| form))
  ((null (rest form)) (list 'quote (list (first form))))
  (t
   (cons 'list
         (cons (mkq (first form))
               (loop for x in (rest form) collect (|mkConstructor| x)))))))
```

5.2.42 defun compDefineCategory

```
[compDefineLisplib p169]
[compDefineCategory1 p137]
[$domainShell p??]
[$lisplibCategory p??]
[$lisplib p??]
[$insideFunctorIfTrue p??]
```

— defun compDefineCategory —

```
(defun |compDefineCategory| (df mode env prefix fal)
  (let (|$domainShell| |$lisplibCategory|)
    (declare (special |$domainShell| |$lisplibCategory| $lisplib
                      |$insideFunctorIfTrue|))
    (setq |$domainShell| nil) ; holds the category of the object being compiled
    (setq |$lisplibCategory| nil)
    (if (and (null |$insideFunctorIfTrue|) $lisplib)
        (|compDefineLisplib| df mode env prefix fal '|compDefineCategory1|)
        (|compDefineCategory1| df mode env prefix fal))))
```

5.2.43 defun compDefineLisplib

```
[sayMSG p??]
[fillerSpaces p??]
[getConstructorAbbreviation p??]
[compileDocumentation p172]
[bright p??]
[finalizeLisplib p174]
[rshut p??]
[lisplibDoRename p172]
[filep p??]
```



```

[rpackfile p??]
[unloadOneConstructor p??]
[localdatabase p??]
[getdatabase p??]
[updateCategoryFrameForCategory p113]
[updateCategoryFrameForConstructor p112]
[$compileDocumentation p172]
[$filep p??]
[$spadLibFT p??]
[$algebraOutputStream p??]
[$newConlist p??]
[$lisplibKind p??]
[$lisplib p??]
[$op p??]
[$lisplibParents p??]
[$lisplibPredicates p??]
[$lisplibCategoriesExtended p??]
[$lisplibForm p??]
[$lisplibKind p??]
[$lisplibAbbreviation p??]
[$lisplibAncestors p??]
[$lisplibModemap p??]
[$lisplibModemapAlist p??]
[$lisplibSlot1 p??]
[$lisplibOperationAlist p??]
[$lisplibSuperDomain p??]
[$libFile p??]
[$lisplibVariableAlist p??]
[$lisplibCategory p??]
[$newConlist p??]

```

— **defun compDefineLisplib** —

```

(defun |compDefineLisplib| (df m env prefix fal fn)
  (let ($LISPLIB |$op| |$lisplibAttributes| |$lisplibPredicates|
        |$lisplibCategoriesExtended| |$lisplibForm| |$lisplibKind|
        |$lisplibAbbreviation| |$lisplibParents| |$lisplibAncestors|
        |$lisplibModemap| |$lisplibModemapAlist| |$lisplibSlot1|
        |$lisplibOperationAlist| |$lisplibSuperDomain| |$libFile|
        |$lisplibVariableAlist| |$lisplibCategory| op libname res ok filearg)
    (declare (special $lisplib |$op| |$lisplibAttributes| |$newConlist|
                      |$lisplibPredicates| |$lisplibCategoriesExtended| | |
                      |$lisplibForm| |$lisplibKind| |$algebraOutputStream|
                      |$lisplibAbbreviation| |$lisplibParents| |$spadLibFT|
                      |$lisplibAncestors| |$lisplibModemap| $filep
                      |$lisplibModemapAlist| |$lisplibSlot1|
                      |$lisplibOperationAlist| |$lisplibSuperDomain|

```

```

        |$libFile| |$lisplibVariableAlist|
        |$lisplibCategory| |$compileDocumentation|))
(when (eq (car df) 'def) (car df))
(setq op (caadr df))
(|sayMSG| (|fillerSpaces| 72 "-"))
(setq $lisplib t)
(setq |$op| op)
(setq |$lisplibAttributes| nil)
(setq |$lisplibPredicates| nil)
(setq |$lisplibCategoriesExtended| nil)
(setq |$lisplibForm| nil)
(setq |$lisplibKind| nil)
(setq |$lisplibAbbreviation| nil)
(setq |$lisplibParents| nil)
(setq |$lisplibAncestors| nil)
(setq |$lisplibModemap| nil)
(setq |$lisplibModemapAlist| nil)
(setq |$lisplibSlot1| nil)
(setq |$lisplibOperationAlist| nil)
(setq |$lisplibSuperDomain| nil)
(setq |$libFile| nil)
(setq |$lisplibVariableAlist| nil)
(setq |$lisplibCategory| nil)
(setq libname (|getConstructorAbbreviation| op))
(cond
  ((and (boundp '|$compileDocumentation|) |$compileDocumentation|)
    (|compileDocumentation| libname))
  (t
    (|sayMSG| (cons "    initializing " (cons |$spadLibFT|
      (append (|bright| libname) (cons "for" (|bright| op))))))
    (|initializeLisplib| libname)
    (|sayMSG|
      (cons "    compiling into " (cons |$spadLibFT| (|bright| libname))))
    (setq ok nil)
    (unwind-protect
      (progn
        (setq res (funcall fn df m env prefix fal))
        (|sayMSG| (cons "    finalizing " (cons |$spadLibFT| (|bright| libname))))
        (|finalizeLisplib| libname)
        (setq ok t))
        (rshut |$libFile|))
      (when ok (|lisplibDoRename| libname))
      (setq filearg ($filep libname |$spadLibFT| 'a))
      (rpackfile filearg)
      (fresh-line |$algebraOutputStream|)
      (|sayMSG| (|fillerSpaces| 72 "-"))
      (|unloadOneConstructor| op libname)
      (localdatabase (list (getdatabase op 'abbreviation)) nil)
      (setq |$newConlist| (cons op |$newConlist|))
      (when (eq |$lisplibKind| '|category|)

```

```
(|updateCategoryFrameForCategory| op)
(|updateCategoryFrameForConstructor| op))
res)))
```

5.2.44 defun compileDocumentation

```
[make-input-filename p??]
[rdefiostream p??]
[lisplibWrite p180]
[finalizeDocumentation p??]
[rshut p??]
[rpackfile p??]
[replaceFile p??]
[$fcopy p??]
[$spadLibFT p??]
[$EmptyMode p131]
[$e p??]
```

— defun compileDocumentation —

```
(defun |compileDocumentation| (libName)
  (let (filename stream)
    (declare (special |$e| |$EmptyMode| |$spadLibFT| $fcopy))
    (setq filename (make-input-filename libName |$spadLibFT|))
    ($fcopy filename (cons libname (list 'doclb)))
    (setq stream
      (rdefiostream (cons (list 'file libName 'doclb) (list (cons 'mode 'o))))))
    (|lisplibWrite| "documentation" (|finalizeDocumentation|) stream)
    (rshut stream)
    (rpackfile (list libName 'doclb))
    (replaceFile (list libName |$spadLibFT|) (list libName 'doclb))
    (list '|dummy| |$EmptyMode| |$e|)))
```

5.2.45 defun lisplibDoRename

```
[replaceFile p??]
[$spadLibFT p??]
```

— defun lisplibDoRename —

```
(defun |lisplibDoRename| (libName)
  (declare (special |$spadLibFT|))
  (replaceFile (list libName |$spadLibFT| 'a) (list libName 'errorlib 'a)))
```

5.2.46 defun initializeLisplib

```
[erase p??]
[writeLib1 p174]
[addoptions p??]
[pathnameTypeId p??]
[LAM,FILEACTQ p??]
[$erase p??]
[$libFile p??]
[$libFile p??]
[$lisplibForm p??]
[$lisplibModemap p??]
[$lisplibKind p??]
[$lisplibModemapAlist p??]
[$lisplibAbbreviation p??]
[$lisplibAncestors p??]
[$lisplibOpAlist p??]
[$lisplibOperationAlist p??]
[$lisplibSuperDomain p??]
[$lisplibVariableAlist p??]
[$lisplibSignatureAlist p??]
[/editfile p??]
[/major-version p??]
[errors p??]
```

— defun initializeLisplib —

```
(defun |initializeLisplib| (libName)
  (declare (special $erase |$libFile| |$lisplibForm|
    |$lisplibModemap| |$lisplibKind| |$lisplibModemapAlist|
    |$lisplibAbbreviation| |$lisplibAncestors|
    |$lisplibOpAlist| |$lisplibOperationAlist|
    |$lisplibSuperDomain| |$lisplibVariableAlist| errors
    |$lisplibSignatureAlist| /editfile /major-version errors))
  ($erase libName 'errorlib 'a)
  (setq errors 0)
  (setq |$libFile| (|writeLib1| libname 'errorlib 'a))
  (addoptions 'file |$libFile|)
  (setq |$lisplibForm| nil)
  (setq |$lisplibModemap| nil)
```

```

(setq |$lisplibKind| nil)
(setq |$lisplibModemapAlist| nil)
(setq |$lisplibAbbreviation| nil)
(setq |$lisplibAncestors| nil)
(setq |$lisplibOpAlist| nil)
(setq |$lisplibOperationAlist| nil)
(setq |$lisplibSuperDomain| nil)
(setq |$lisplibVariableAlist| nil)
(setq |$lisplibSignatureAlist| nil)
(when (eq (|pathnameTypeId| /editfile) 'spad)
  (|LAM,FILEACTQ| 'version (list '/versioncheck /major-version))))

```

5.2.47 defun writeLib1

[rdefiostream p??]

— defun writeLib1 —

```

(defun |writeLib1| (fn ft fm)
  (rdefiostream (cons (list 'file fn ft fm) (list '(mode . output)))))

```

5.2.48 defun finalizeLisplib

```

[lisplibWrite p180]
[removeZeroOne p??]
[namestring p??]
[getConstructorOpsAndAtts p176]
[NRTgenInitialAttributeAlist p??]
[mergeSignatureAndLocalVarAlists p180]
[finalizeDocumentation p??]
[profileWrite p??]
[makeprop p??]
[sayMSG p??]
[$lisplibForm p??]
[$libFile p??]
[$lisplibKind p??]
[$lisplibModemap p??]
[$lisplibCategory p??]
[$/editfile p??]
[$lisplibModemapAlist p??]

```

```

[$lisplibForm p??]
[$lisplibModemap p??]
[$FormalMapVariableList p242]
[$lisplibSuperDomain p??]
[$lisplibSignatureAlist p??]
[$lisplibVariableAlist p??]
[$lisplibAttributes p??]
[$lisplibPredicates p??]
[$lisplibAbbreviation p??]
[$lisplibParents p??]
[$lisplibAncestors p??]
[$lisplibSlot1 p??]
[$profileCompiler p??]
[$spadLibFT p??]
[$lisplibCategory p??]
[$pairlis p??]
[$NRTslot1PredicateList p??]

```

— **defun finalizeLisplib** —

```

(defun |finalizeLisplib| (libName)
  (let (|$pairlis| |$NRTslot1PredicateList| kind opsAndAtts)
    (declare (special |$pairlis| |$NRTslot1PredicateList| |$spadLibFT|
                      |$lisplibForm| |$profileCompiler| |$libFile|
                      |$lisplibSlot1| |$lisplibAncestors| |$lisplibParents|
                      |$lisplibAbbreviation| |$lisplibPredicates|
                      |$lisplibAttributes| |$lisplibVariableAlist|
                      |$lisplibSignatureAlist| |$lisplibSuperDomain|
                      |$FormalMapVariableList| |$lisplibModemap|
                      |$lisplibModemapAlist| /editfile |$lisplibCategory|
                      |$lisplibKind| errors))
      (|lisplibWrite| "constructorForm"
        (|removeZeroOne| |$lisplibForm|) |$libFile|)
      (|lisplibWrite| "constructorKind"
        (setq kind (|removeZeroOne| |$lisplibKind|)) |$libFile|)
      (|lisplibWrite| "constructorModemap"
        (|removeZeroOne| |$lisplibModemap|) |$libFile|)
      (setq |$lisplibCategory| (or |$lisplibCategory| (cadar |$lisplibModemap|)))
      (|lisplibWrite| "constructorCategory" |$lisplibCategory| |$libFile|)
      (|lisplibWrite| "sourceFile" (|namestring| /editfile) |$libFile|)
      (|lisplibWrite| "modemaps"
        (|removeZeroOne| |$lisplibModemapAlist|) |$libFile|)
      (setq opsAndAtts
        (|getConstructorOpsAndAtts| |$lisplibForm| kind |$lisplibModemap|))
      (|lisplibWrite| "operationAlist"
        (|removeZeroOne| (car opsAndAtts)) |$libFile|)
      (when (eq kind '|category|)
        (setq |$pairlis|

```

```

(loop for a in (rest |$lisplibForm|)
  for v in |$FormalMapVariableList|
  collect (cons a v)))
(setq |$NRTslot1PredicateList| nil)
(|NRTgenInitialAttributeAlist| (cdr opsAndAtts)))
(|lisplibWrite| "superDomain"
  (|removeZeroOne| |$lisplibSuperDomain|) |$libFile|)
(|lisplibWrite| "signaturesAndLocals"
  (|removeZeroOne|
    (|mergeSignatureAndLocalVarAlists| |$lisplibSignatureAlist|
      |$lisplibVariableAlist|))
    |$libFile|)
(|lisplibWrite| "attributes"
  (|removeZeroOne| |$lisplibAttributes|) |$libFile|)
(|lisplibWrite| "predicates"
  (|removeZeroOne| |$lisplibPredicates|) |$libFile|)
(|lisplibWrite| "abbreviation" |$lisplibAbbreviation| |$libFile|)
(|lisplibWrite| "parents" (|removeZeroOne| |$lisplibParents|) |$libFile|)
(|lisplibWrite| "ancestors" (|removeZeroOne| |$lisplibAncestors|) |$libFile|)
(|lisplibWrite| "documentation" (|finalizeDocumentation|) |$libFile|)
(|lisplibWrite| "slot1Info" (|removeZeroOne| |$lisplibSlot1|) |$libFile|)
(when |$profileCompiler| (|profileWrite|))
(when (and |$lisplibForm| (null (cdr |$lisplibForm|)))
  (makeprop (car |$lisplibForm|) 'niladic t))
(unless (eql errors 0)
  (|sayMSG| (list " Errors in processing " kind " " libName ":"))
  (|sayMSG| (list " not replacing " |$spadLibFT| " for" libName))))

```

5.2.49 defun getConstructorOpsAndAtts

[getCategoryOpsAndAtts p177]
 [getFunctorOpsAndAtts p179]

— defun getConstructorOpsAndAtts —

```

(defun |getConstructorOpsAndAtts| (form kind modemap)
  (if (eq kind 'category|)
    (|getCategoryOpsAndAtts| form)
    (|getFunctorOpsAndAtts| form modemap)))

```

5.2.50 defun getCategoryOpsAndAtts

```
[transformOperationAlist p177]
[getSlotFromCategoryForm p177]
[getSlotFromCategoryForm p177]
```

— defun getCategoryOpsAndAtts —

```
(defun |getCategoryOpsAndAtts| (catForm)
  (cons (|transformOperationAlist| (|getSlotFromCategoryForm| catForm 1))
        (|getSlotFromCategoryForm| catForm 2)))
```

5.2.51 defun getSlotFromCategoryForm

```
[eval p??]
[take p??]
[systemErrorHere p??]
[$FormalMapVariableList p242]
```

— defun getSlotFromCategoryForm —

```
(defun |getSlotFromCategoryForm| (opargs index)
  (let (op argl u)
    (declare (special |$FormalMapVariableList|))
    (setq op (first opargs))
    (setq argl (rest opargs))
    (setq u
      (|eval| (cons op (mapcar 'mkq (take (|#| argl) |$FormalMapVariableList|)))))
    (if (null (vecp u))
        (|systemErrorHere| "getSlotFromCategoryForm")
        (elt u index))))
```

5.2.52 defun transformOperationAlist

This transforms the operationAlist which is written out onto LISPLIBs. The original form of this list is a list of items of the form:

```
((<op> <signature>) (<condition> (ELT $ n)))
```

The new form is an op-Alist which has entries


```
(<op> . signature-Alist)
```

where signature-Alist has entries

```
(<signature> . item)
```

where item has form

```
(<slotNumber> <condition> <kind>)
```

```
where <kind> =
  NIL => function
  CONST => constant ... and others
```

```
[member p??]
[keyedSystemError p??]
[assoc p??]
[lassq p??]
[insertAlist p??]
[$functionLocations p??]
```

— **defun transformOperationAlist** —

```
(defun |transformOperationAlist| (operationAlist)
  (let (op sig condition implementation eltEtc impOp kind u n signatureItem
        itemList newAlist)
    (declare (special |$functionLocations|))
    (setq newAlist nil)
    (dolist (item operationAlist)
      (setq op (caar item))
      (setq sig (cadar item))
      (setq condition (cadr item))
      (setq implementation (caddr item))
      (setq kind
        (cond
          ((and (consp implementation) (consp (qrest implementation))
                (consp (qcddr implementation))
                (eq (qcdddr implementation) nil)
                (progn (setq n (qthird implementation)) t)
                (|member| (setq eltEtc (qfirst implementation)) '(const elt)))
            eltEtc)
          ((consp implementation)
            (setq impOp (qfirst implementation))
            (cond
              ((eq impOp 'xlam) implementation)
              ((|member| impOp '(const |Subsumed|)) impOp)
              (t (|keyedSystemError| 's2il0025 (list impOp))))))
        ((eq implementation '|mkRecord|) '|mkRecord|)
```

```

      (t (|keyedSystemError| 's2il0025 (list implementation))))
    (when (setq u (|assoc| (list op sig) |$functionLocations|))
      (setq n (cons n (cdr u))))
    (setq signatureItem
      (if (eq kind 'elt)
        (if (eq condition t)
          (list sig n)
          (list sig n condition))
        (list sig n condition kind)))
    (setq itemList (cons signatureItem (lassq op newAlist)))
    (setq newAlist (|insertAlist| op itemList newAlist))
    newAlist))

```

5.2.53 defun getFunctorOpsAndAtts

```

[transformOperationAlist p177]
[getSlotFromFunctor p179]

```

— defun getFunctorOpsAndAtts —

```

(defun |getFunctorOpsAndAtts| (form modemap)
  (cons (|transformOperationAlist| (|getSlotFromFunctor| form 1 modemap))
        (|getSlotFromFunctor| form 2 modemap)))

```

5.2.54 defun getSlotFromFunctor

```

[compMakeCategoryObject p180]
[systemErrorHere p??]
[$e p??]
[$lisplibOperationAlist p??]

```

— defun getSlotFromFunctor —

```

(defun |getSlotFromFunctor| (arg1 slot arg2)
  (declare (ignore arg1))
  (let (tt)
    (declare (special |$e| |$lisplibOperationAlist|))
    (cond
      ((eq1 slot 1) |$lisplibOperationAlist|)
      (t
       (setq tt (or (|compMakeCategoryObject| (cadar arg2) |$e|)

```

```
(|systemErrorHere| "getSlotFromFunctor"))
(elt (car tt) slot))))))
```

5.2.55 defun compMakeCategoryObject

```
[isCategoryForm p??]
[mkEvaluableCategoryForm p140]
[$e p??]
[$Category p??]
```

— defun compMakeCategoryObject —

```
(defun |compMakeCategoryObject| (c |$e|)
  (declare (special |$e|))
  (let (u)
    (declare (special |$Category|))
    (cond
      ((null (|isCategoryForm| c |$e|)) nil)
      ((setq u (|mkEvaluableCategoryForm| c)) (list (|eval| u) |$Category| |$e|))
      (t nil))))
```

5.2.56 defun mergeSignatureAndLocalVarAlists

```
[lassoc p??]
```

— defun mergeSignatureAndLocalVarAlists —

```
(defun |mergeSignatureAndLocalVarAlists| (signatureAlist localVarAlist)
  (loop for item in signatureAlist
    collect
      (cons (first item)
            (cons (rest item)
                  (lassoc (first item) localVarAlist)))))
```

5.2.57 defun lisplibWrite

```
[rwrite128 p??]
[$lisplib p??]
```

— defun lisplibWrite —

```
(defun |lisplibWrite| (prop val filename)
  (declare (special $lisplib))
  (when $lisplib (|rwrite| prop val filename)))
```

5.2.58 defun compDefineFunctor

```
[compDefineLisplib p169]
[compDefineFunctor1 p181]
[$domainShell p??]
[$profileCompiler p??]
[$lisplib p??]
[$profileAlist p??]
```

— defun compDefineFunctor —

```
(defun |compDefineFunctor| (df mode env prefix fal)
  (let (|$domainShell| |$profileCompiler| |$profileAlist|)
    (declare (special |$domainShell| |$profileCompiler| $lisplib |$profileAlist|))
    (setq |$domainShell| nil)
    (setq |$profileCompiler| t)
    (setq |$profileAlist| nil)
    (if $lisplib
      (|compDefineLisplib| df mode env prefix fal '|compDefineFunctor1|)
      (|compDefineFunctor1| df mode env prefix fal))))
```

5.2.59 defun compDefineFunctor1

```
[isCategoryPackageName p??]
[getArgumentModeOrMoan p154]
[getModemap p234]
[giveFormalParametersValues p135]
[compMakeCategoryObject p180]
[sayBrightly p??]
[pp p??]
[strconc p??]
[pname p??]
[disallowNilAttribute p191]
```

```

[remdup p??]
[NRTgenInitialAttributeAlist p??]
[NRTgetLocalIndex p??]
[compMakeDeclaration p525]
[qcar p??]
[qcdr p??]
[augModemapsFromCategoryRep p243]
[augModemapsFromCategory p237]
[sublis p??]
[maxindex p??]
[makeFunctorArgumentParameters p194]
[compFunctorBody p191]
[reportOnFunctorCompilation p192]
[compile p145]
[augmentLisplibModemapsFromFunctor p189]
[reportOnFunctorCompilation p192]
[getParentsFor p??]
[computeAncestorsOf p??]
[constructor? p??]
[nequal p??]
[NRTmakeSlot1Info p??]
[isCategoryPackageName p??]
[lisplibWrite p180]
[mkq p??]
[getdatabase p??]
[NRTgetLookupFunction p??]
[simpBool p??]
[removeZeroOne p??]
[evalAndRwriteLispForm p168]
[$lisplib p??]
[$top-level p??]
[$bootStrapMode p??]
[$CategoryFrame p??]
[$CheckVectorList p??]
[$FormalMapVariableList p242]
[$LocalDomainAlist p??]
[$NRTaddForm p??]
[$NRTaddList p??]
[$NRTattributeAlist p??]
[$NRTbase p??]
[$NRTdeltaLength p??]
[$NRTdeltaListComp p??]
[$NRTdeltaList p??]
[$NRTdomainFormList p??]
[$NRTloadTimeAlist p??]
[$NRTslot1Info p??]

```

```

[$NRTslot1PredicateList p??]
[$Representation p??]
[$addForm p??]
[$attributesName p??]
[$byteAddress p??]
[$byteVec p??]
[$compileOnlyCertainItems p??]
[$condAlist p??]
[$domainShell p??]
[$form p??]
[$functionLocations p??]
[$functionStats p??]
[$functorForm p??]
[$functorLocalParameters p??]
[$functorStats p??]
[$functorSpecialCases p??]
[$functorTarget p??]
[$functorsUsed p??]
[$genFVar p??]
[$genSDVar p??]
[$getDomainCode p??]
[$goGetList p??]
[$insideCategoryPackageIfTrue p??]
[$insideFunctorIfTrue p??]
[$isOpPackageName p??]
[$libFile p??]
[$lisplibAbbreviation p??]
[$lisplibAncestors p??]
[$lisplibCategoriesExtended p??]
[$lisplibCategory p??]
[$lisplibForm p??]
[$lisplibKind p??]
[$lisplibMissingFunctions p??]
[$lisplibModemap p??]
[$lisplibOperationAlist p??]
[$lisplibParents p??]
[$lisplibSlot1 p??]
[$lookupFunction p??]
[$myFunctorBody p??]
[$mutableDomain p??]
[$mutableDomains p??]
[$op p??]
[$pairlis p??]
[$QuickCode p??]
[$setelt p??]
[$signature p??]

```

```

[template p??]
[uncondAlist p??]
[viewNames p??]
[lisplibFunctionLocations p??]

```

— defun compDefineFunctor1 —

```

(defun |compDefineFunctor1| (df mode |$e| |$prefix| |$formalArgList|)
  (declare (special |$e| |$prefix| |$formalArgList|))
  (labels (
    (FindRep (cb)
      (loop while cb do
        (when (atom cb) (return nil))
        (when (and (consp cb) (consp (qfirst cb)) (eq (qcaar cb) 'let)
          (consp (qcddar cb)) (eq (qcadar cb) '|Rep|)
          (consp (qcddar cb)))
        (return (caddar cb)))
      (pop cb))))
    (let (|$addForm| |$viewNames| |$functionStats| |$functorStats|
          |$form| |$op| |$signature| |$functorTarget|
          |$Representation| |$LocalDomainAlist| |$functorForm|
          |$functorLocalParameters| |$CheckVectorList|
          |$getDomainCode| |$insideFunctorIfTrue| |$functorsUsed|
          |$setelt| $TOP_LEVEL |$genFVar| |$genSDVar|
          |$mutableDomain| |$attributesName| |$goGetList|
          |$condAlist| |$uncondAlist| |$NRTslot1PredicateList|
          |$NRTattributeAlist| |$NRTslot1Info| |$NRTbase|
          |$NRTaddForm| |$NRTdeltaList| |$NRTdeltaListComp|
          |$NRTaddList| |$NRTdeltaLength| |$NRTloadTimeAlist|
          |$NRTdomainFormList| |$template| |$functionLocations|
          |$isUpPackageName| |$lookupFunction| |$byteAddress|
          |$byteVec| form signature body originale argl signaturep target ds
          attributeList parSignature parForm
          argPars opp rettype tt bodyp lamOrSlam fun
          operationAlist modemap libFn tmp1)
      (declare (special $lisplib $top_level $bootstrapMode| |$CategoryFrame|
                    |$CheckVectorList| |$FormalMapVariableList| | | | |
                    |$LocalDomainAlist| |$NRTaddForm| |$NRTaddList|
                    |$NRTattributeAlist| |$NRTbase| |$NRTdeltaLength|
                    |$NRTdeltaListComp| |$NRTdeltaList| |$NRTdomainFormList|
                    |$NRTloadTimeAlist| |$NRTslot1Info| |$NRTslot1PredicateList|
                    |$Representation| |$addForm| |$attributesName|
                    |$byteAddress| |$byteVec| |$compileOnlyCertainItems|
                    |$condAlist| |$domainShell| |$form| |$functionLocations|
                    |$functionStats| |$functorForm| |$functorLocalParameters|
                    |$functorStats| |$functorSpecialCases| |$functorTarget|
                    |$functorsUsed| |$genFVar| |$genSDVar| |$getDomainCode|
                    |$goGetList| |$insideCategoryPackageIfTrue|
                    |$insideFunctorIfTrue| |$isUpPackageName| |$libFile|

```

```

    |$lisplibAbbreviation| |$lisplibAncestors| | | | |
    |$lisplibCategoriesExtended| |$lisplibCategory|
    |$lisplibForm| |$lisplibKind| |$lisplibMissingFunctions|
    |$lisplibModemap| |$lisplibOperationAlist| |$lisplibParents|
    |$lisplibSlot1| |$lookupFunction| |$myFunctorBody|
    |$mutableDomain| |$mutableDomains| |$op| |$pairlis|
    |$QuickCode| |$setelt| |$signature| |$template|
    |$uncondAlist| |$viewNames| |$lisplibFunctionLocations|))
(setq form (second df))
(setq signature (third df))
(setq |$functorSpecialCases| (fourth df))
(setq body (fifth df))
(setq |$addForm| nil)
(setq |$viewNames| nil)
(setq |$functionStats| (list 0 0))
(setq |$functorStats| (list 0 0))
(setq |$form| nil)
(setq |$op| nil)
(setq |$signature| nil)
(setq |$functorTarget| nil)
(setq |$Representation| nil)
(setq |$LocalDomainAlist| nil)
(setq |$functorForm| nil)
(setq |$functorLocalParameters| nil)
(setq |$myFunctorBody| body)
(setq |$CheckVectorList| nil)
(setq |$getDomainCode| nil)
(setq |$insideFunctorIfTrue| t)
(setq |$functorsUsed| nil)
(setq |$setelt| (if |$QuickCode| 'qsetrefv 'setelt))
(setq $top_level nil)
(setq |$genFVar| 0)
(setq |$genSDVar| 0)
(setq originale |$e|)
(setq |$op| (first form))
(setq argl (rest form))
(setq |$formalArgList| (append argl |$formalArgList|))
(setq |$pairlis|
  (loop for a in argl for v in |$FormalMapVariableList|
    collect (cons a v)))
(setq |$mutableDomain|
  (OR (|isCategoryPackageName| |$op|)
    (COND
      ((boundp '|$mutableDomains|)
       (member |$op| |$mutableDomains|))
      ('T NIL))))
(setq signaturep
  (cons (car signature)
    (loop for a in argl collect (|getArgumentModeOrMoan| a form |$e|))))
(setq |$form| (cons |$op| argl))

```



```

(setq |$functorForm| |$form|)
(unless (car signaturep)
  (setq signaturep (cdar (|getModemap| |$form| |$e|))))
(setq target (first signaturep))
(setq |$functorTarget| target)
(setq |$e| (|giveFormalParametersValues| arg1 |$e|))
(setq tmp1 (|compMakeCategoryObject| target |$e|))
(if tmp1
  (progn
    (setq ds (first tmp1))
    (setq |$e| (third tmp1))
    (setq |$domainShell| (copy-seq ds))
    (setq |$attributesName| (intern (strconc (pname |$op|) ";attributes")))
    (setq attributeList (|disallowNilAttribute| (elt ds 2)))
    (setq |$goGetList| nil)
    (setq |$condAlist| nil)
    (setq |$uncondAlist| nil)
    (setq |$NRTslot1PredicateList|
      (remdup (loop for x in attributeList collect (second x))))
    (setq |$NRTattributeAlist| (|NRTgenInitialAttributeAlist| attributeList))
    (setq |$NRTslot1Info| nil)
    (setq |$NRTbase| 6)
    (setq |$NRTaddForm| nil)
    (setq |$NRTdeltaList| nil)
    (setq |$NRTdeltaListComp| nil)
    (setq |$NRTaddList| nil)
    (setq |$NRTdeltaLength| 0)
    (setq |$NRTloadTimeAlist| nil)
    (setq |$NRTdomainFormList| nil)
    (setq |$template| nil)
    (setq |$functionLocations| nil)
    (loop for x in arg1 do (|NRTgetLocalIndex| x))
    (setq |$e|
      (third (|compMakeDeclaration| (list '(:| '$ target) mode |$e|)))
      (unless |$insideCategoryPackageIfTrue|
        (if
          (and (consp body) (eq (qfirst body) '|add|)
            (consp (qrest body))
            (consp (qsecond body))
            (consp (qcddr body))
            (eq (qcdddr body) nil)
            (consp (qthird body))
            (eq (qcaaddr body) '|capsule|)
            (member (qcaadr body) '(|List| |Vector|))
            (equal (FindRep (qcdaddr body)) (second body)))
          (setq |$e| (|augModemapsFromCategoryRep| '$
            (second body) (cdaddr body) target |$e|))
          (setq |$e| (|augModemapsFromCategory| '$ '$ target |$e|))))
    (setq |$signature| signaturep)
    (setq operationAlist (sublis |$pairlis| (elt |$domainShell| 1)))

```

```

(setq parSignature (sublis |$pairlis| signaturep))
(setq parForm (sublis |$pairlis| form))
(setq argPars (|makeFunctorArgumentParameters| arg1
              (cdr signaturep) (car signaturep)))
(setq |$functorLocalParameters| arg1)
(setq opp |$op|)
(setq rettype (CAR signaturep))
(setq tt (|compFunctorBody| body rettype |$e| parForm))
(cond
  (|$compileOnlyCertainItems|
   (|reportOnFunctorCompilation|
    (list nil (cons '|Mapping| signaturep) originale)))
  (t
   (setq bodyp (first tt))
   (setq lamOrSlam (if |$mutableDomain| 'lam 'spadslam))
   (setq fun
    (|compile| (sublis |$pairlis| (list opp (list lamOrSlam arg1 bodyp)))))
   (setq operationAlist (sublis |$pairlis| |$lisplibOperationAlist|))
   (cond
    ($lisplib
     (|augmentLisplibModemapsFromFunctor| parForm
      operationAlist parSignature)))
   (|reportOnFunctorCompilation|
    (cond
     ($lisplib
      (setq modemap (list (cons parForm parSignature) (list t opp)))
      (setq |$lisplibModemap| modemap)
      (setq |$lisplibCategory| (cadar modemap))
      (setq |$lisplibParents|
        (|getParentsFor| |$op| |$FormalMapVariableList| |$lisplibCategory|))
      (setq |$lisplibAncestors| (|computeAncestorsOf| |$form| NIL))
      (setq |$lisplibAbbreviation| (|constructor?| |$op|)))
     (setq |$insideFunctorIfTrue| NIL)
     (cond
      ($lisplib
       (setq |$lisplibKind|
        (if (and (consp |$functorTarget|)
                 (eq (qfirst |$functorTarget|) 'category)
                 (consp (qrest |$functorTarget|))
                 (nequal (qsecond |$functorTarget|) '|domain|))
          '|package|
          '|domain|))
       (setq |$lisplibForm| form)
       (cond
        ((null |$bootStrapMode|)
         (setq |$NRTslot1Info| (|NRTmakeSlot1Info|))
         (setq |$isOpPackageName| (|isCategoryPackageName| |$op|))
         (when |$isOpPackageName|
          (|lisplibWrite| "slot1DataBase"
            (list '|updateSlot1DataBase| (mkq |$NRTslot1Info|))

```

```

    |$libFile|))
(setq |$lisplibFunctionLocations|
  (sublis |$pairlis| |$functionLocations|))
(setq |$lisplibCategoriesExtended|
  (sublis |$pairlis| |$lisplibCategoriesExtended|))
(setq libFn (getdatabase opp 'abbreviation))
(setq |$lookupFunction|
  (|NRTgetLookupFunction| |$functorForm|
   (cadar |$lisplibModemap|) |$NRTaddForm|))
(setq |$byteAddress| 0)
(setq |$byteVec| NIL)
(setq |$NRTslot1PredicateList|
  (loop for x in |$NRTslot1PredicateList|
    collect (|simpBool| x)))
(|rewriteLispForm| 'loadTimeStuff|
  (list 'makeprop (mkq |$op|) ''|inforec| (|getInforecCode|))))
(setq |$lisplibSlot1| |$NRTslot1Info|)
(setq |$lisplibOperationAlist| operationAlist)
(setq |$lisplibMissingFunctions| |$CheckVectorList|))
(|lisplibWrite| "compilerInfo"
  (|removeZeroOne|
    (list 'setq '|$CategoryFrame|
      (list '|put| (list 'quote opp) ''|isFunctor|
        (list 'quote operationAlist)
        (list 'addModemap|
          (list 'quote opp)
          (list 'quote parForm)
          (list 'quote parSignature)
          t
          (list 'quote opp)
          (list '|put| (list 'quote opp) ''|mode|
            (list 'quote (cons '|Mapping| parSignature))
            '|$CategoryFrame|))))))
      |$libFile|)
    (unless arg1
      (|evalAndRwriteLispForm| 'niladic
        (list 'makeprop (list 'quote opp) (list 'quote 'niladic) t)))
      (list fun (cons '|Mapping| signaturep) originale))))
(progn
  (|sayBrightly| "    cannot produce category object:")
  (|pp| target)
  nil))))

```

5.2.60 defun augmentLisplibModemapsFromFunctor

```

[formal2Pattern p190]
[mkAlistOfExplicitCategoryOps p156]
[allLASSOCs p190]
[member p??]
[msubst p??]
[mkDatabasePred p191]
[mkpf p??]
[listOfPatternIds p??]
[interactiveModemapForm p158]
[$lisplibModemapAlist p??]
[$PatternVariableList p??]
[$e p??]
[$lisplibModemapAlist p??]
[$e p??]

```

— defun augmentLisplibModemapsFromFunctor —

```

(defun |augmentLisplibModemapsFromFunctor| (form opAlist signature)
  (let (argl nonCategorySigAlist op pred sel predList sig predp z skip modemap)
    (declare (special |$lisplibModemapAlist| |$PatternVariableList| |$e|))
    (setq form (|formal2Pattern| form))
    (setq argl (cdr form))
    (setq opAlist (|formal2Pattern| opAlist))
    (setq signature (|formal2Pattern| signature))
    ; We are going to be EVALing categories containing these pattern variables
    (loop for u in form for v in signature
      do (when (member u |$PatternVariableList|)
          (setq |$e| (|put| u '|mode| v |$e|))))
    (when
      (setq nonCategorySigAlist (|mkAlistOfExplicitCategoryOps| (CAR signature)))
      (loop for entry in opAlist
        do
          (setq op (caar entry))
          (setq sig (cadar entry))
          (setq pred (cadr entry))
          (setq sel (caddr entry))
          (when
            (let (result)
              (loop for catSig in (|allLASSOCs| op nonCategorySigAlist)
                do (setq result (or result (|member| sig catSig))))
              result)
            (setq skip (when (and argl (contained '$ (cdr sig))) 'skip))
            (setq sel (msubst form '$ sel))
            (setq predList
              (loop for a in argl for m in (rest signature)
                when (|member| a |$PatternVariableList|)

```

```

      collect (list a m)))
(setq sig (msubst form '$ sig))
(setq predp
  (mkpf
    (cons pred (loop for y in predList collect (|mkDatabasePred| y)))
    'and))
(setq z (|listOfPatternIds| predList))
(when (some #'(lambda (u) (null (member u z))) arg1)
  (|sayMSG| (list "cannot handle modemap for " op "by pattern match")))
(setq skip 'skip))
(setq modemap (list (cons form sig) (cons predp (cons sel skip))))
(setq |$lisplibModemapAlist|
  (cons
    (cons op (|interactiveModemapForm| modemap))
    |$lisplibModemapAlist|))))))

```

5.2.61 defun allLASSOCs

— defun allLASSOCs —

```

(defun |allLASSOCs| (op alist)
  (loop for value in alist
    when (equal (car value) op)
    collect value))

```

5.2.62 defun formal2Pattern

```

[sublis p??]
[pairList p??]
[$PatternVariableList p??]

```

— defun formal2Pattern —

```

(defun |formal2Pattern| (x)
  (declare (special |$PatternVariableList|))
  (sublis (|pairList| |$FormalMapVariableList| (cdr |$PatternVariableList|)) x))

```

5.2.63 defun mkDatabasePred

```
[isCategoryForm p??]
[$e p??]
```

— defun mkDatabasePred —

```
(defun |mkDatabasePred| (arg)
  (let (a z)
    (declare (special |$e|))
    (setq a (car arg))
    (setq z (cadr arg))
    (if (|isCategoryForm| z |$e|)
        (list '|ofCategory| a z)
        (list '|ofType| a z))))
```

5.2.64 defun disallowNilAttribute

— defun disallowNilAttribute —

```
(defun |disallowNilAttribute| (x)
  (loop for y in x when (and (car y) (nequal (car y) '|nil|))
    collect y))
```

5.2.65 defun compFunctorBody

```
[bootStrapError p192]
[compOrCroak p496]
[/editfile p??]
[$NRTaddForm p??]
[$functorForm p??]
[$bootStrapMode p??]
```

— defun compFunctorBody —

```
(defun |compFunctorBody| (form mode env parForm)
  (declare (ignore parForm))
  (let (tt)
    (declare (special |$NRTaddForm| |$functorForm| |$bootStrapMode| /editfile)))
```

```

(if |$bootStrapMode|
  (list (|bootStrapError| |$functorForm| /editfile) mode env)
  (progn
    (setq tt (|compOrCroak| form mode env))
    (if (and (consp form) (member (qfirst form) '(|add| capsule)))
      tt
      (progn
        (setq |$NRTaddForm|
          (if (and (consp form) (eq (qfirst form) '|SubDomain|)
                (consp (qrest form)) (consp (qcddr form))
                (eq (qcdddr form) nil))
              (qsecond form)
              form))
        tt))))))

```

5.2.66 defun bootStrapError

```

[mkq p??]
[namestring p??]
[mkDomainConstructor p??]

```

— defun bootStrapError —

```

(defun |bootStrapError| (functorForm sourceFile)
  (list 'cond
    (list '|$bootStrapMode|
      (list 'vector (|mkDomainConstructor| functorForm) nil nil nil nil nil))
    (list ''t
      (list '|systemError|
        (list 'list ''|%b| (MKQ (CAR functorForm)) ''|%d| "from" ''|%b|
          (mkq (|namestring| sourceFile)) ''|%d| "needs to be compiled")))))

```

5.2.67 defun reportOnFunctorCompilation

```

[displayMissingFunctions p193]
[sayBrightly p??]
[displaySemanticErrors p??]
[displayWarnings p??]
[addStats p??]
[normalizeStatAndStringify p??]
[$op p??]

```

```
[$functorStats p??]
[$functionStats p??]
[$warningStack p??]
[$semanticErrorStack p??]
```

— **defun reportOnFunctorCompilation** —

```
(defun |reportOnFunctorCompilation| ()
  (declare (special |$op| |$functorStats| |$functionStats|
                    |$warningStack| |$semanticErrorStack|))
  (|displayMissingFunctions|)
  (when |$semanticErrorStack| (|sayBrightly| " "))
  (|displaySemanticErrors|)
  (when |$warningStack| (|sayBrightly| " "))
  (|displayWarnings|)
  (setq |$functorStats| (|addStats| |$functorStats| |$functionStats|))
  (|sayBrightly|
   (cons '|%l|
         (append (|bright| " Cumulative Statistics for Constructor"
                          (list |$op|))))
   (|sayBrightly|
    (cons " Time:"
          (append (|bright| (|normalizeStatAndStringify| (second |$functorStats|))
                    (list "seconds"))))
    (|sayBrightly| " ")
    '|done|)
```

5.2.68 defun displayMissingFunctions

```
[member p??]
[getmode p??]
[sayBrightly p??]
[bright p??]
[formatUnabbreviatedSig p??]
[$env p??]
[$formalArgList p??]
[$CheckVectorList p??]
```

— **defun displayMissingFunctions** —

```
(defun |displayMissingFunctions| ()
  (let (i loc exp)
    (declare (special |$env| |$formalArgList| |$CheckVectorList|))
    (unless |$CheckVectorList|
```



```

(setq loc nil)
(setq exp nil)
(loop for cvl in |$CheckVectorList| do
  (unless (cdr cvl)
    (if (and (null (|member| (caar cvl) |$formalArgList|))
      (consp (|getmode| (caar cvl) |$env|))
      (eq (qfirst (|getmode| (caar cvl) |$env|)) '|Mapping|))
      (push (list (caar cvl) (cadar cvl)) loc)
      (push (list (caar cvl) (cadar cvl)) exp))))
(when loc
  (|sayBrightly| (cons '|%l| (|bright| " Missing Local Functions:"))
    (setq i 0)
    (loop for item in loc do
      (|sayBrightly|
        (cons " [" (cons (incf i) (cons "]"
          (append (|bright| (first item))
            (cons '|: | (|formatUnabbreviatedSig| (second item))))))))))
(when exp
  (|sayBrightly| (cons '|%l| (|bright| " Missing Exported Functions:"))
    (setq i 0)
    (loop for item in exp do
      (|sayBrightly|
        (cons " [" (cons (incf i) (cons "]"
          (append (|bright| (first item))
            (cons '|: | (|formatUnabbreviatedSig| (second item))))))))))

```

5.2.69 defun makeFunctorArgumentParameters

```

[assq p??]
[msubst p??]
[isCategoryForm p??]
[qcar p??]
[qcdr p??]
[genDomainViewList0 p196]
[union p??]
[$ConditionalOperators p??]
[$alternateViewList p??]
[$forceAdd p??]

```

— defun makeFunctorArgumentParameters —

```

(defun |makeFunctorArgumentParameters| (argl sigl target)
  (labels (
    (augmentSig (s ss)
      (let (u)

```

```

(declare (special |$ConditionalOperators|))
(if ss
  (progn
    (loop for u in ss do (push (rest u) |$ConditionalOperators|))
    (if (and (consp s) (eq (qfirst s) '|Join|))
      (progn
        (if (setq u (assq 'category ss))
          (msubst (append u ss) u s)
          (cons '|Join|
            (append (rest s) (list (cons 'category (cons '|package| ss)))))))
      (list '|Join| s (cons 'category (cons '|package| ss))))
    s)))
(fn (a s)
  (declare (special |$CategoryFrame|))
  (if (|isCategoryForm| s |$CategoryFrame|)
    (if (and (consp s) (eq (qfirst s) '|Join|))
      (|genDomainViewList0| a (rest s))
      (list (|genDomainView| a s '|getDomainView|)))
    (list a)))
(findExtras (a target)
  (cond
    ((and (consp target) (eq (qfirst target) '|Join|))
     (reduce #'|union|
       (loop for x in (qrest target)
         collect (findExtras a x))))
    ((and (consp target) (eq (qfirst target) 'category))
     (reduce #'|union|
       (loop for x in (qcddr target)
         collect (findExtras1 a x))))))
(findExtras1 (a x)
  (cond
    ((and (consp x) (or (eq (qfirst x) 'and)) (eq (qfirst x) 'or))
     (reduce #'|union|
       (loop for y in (rest x) collect (findExtras1 a y))))
    ((and (consp x) (eq (qfirst x) 'if)
      (consp (qrest x)) (consp (qcddr x))
      (consp (qcdddr x))
      (eq (qcdddr x) nil))
     (|union| (findExtrasP a (second x))
       (|union|
         (findExtras1 a (third x))
         (findExtras1 a (fourth x))))))
    (t (findExtrasP a x))))
(findExtrasP (a x)
  (cond
    ((and (consp x) (or (eq (qfirst x) 'and)) (eq (qfirst x) 'or))
     (reduce #'|union|
       (loop for y in (rest x) collect (findExtrasP a y))))
    ((and (consp x) (eq (qfirst x) '|has|)
      (consp (qrest x)) (consp (qcddr x))
      (consp (qcdddr x))
      (eq (qcdddr x) nil))
     (|union| (findExtrasP a (second x))
       (|union|
         (findExtras1 a (third x))
         (findExtras1 a (fourth x))))))
    (t (findExtrasP a x))))

```

```

      (eq (qcdddr x) nil))
    (|union| (findExtrasP a (second x))
      (|union|
        (findExtras1 a (third x))
        (findExtras1 a (fourth x))))))
    ((and (consp x) (eq (qfirst x) '|has|)
      (consp (qrest x)) (equal (qsecond x) a)
      (consp (qcddr x))
      (eq (qcddr x) nil)
      (consp (qthird x))
      (eq (qcaaddr x) 'signature))
    (list (third x))))
  )
  (let (|$alternateViewList| |$forceAdd| |$ConditionalOperators|)
    (declare (special |$alternateViewList| |$forceAdd| |$ConditionalOperators|))
    (setq |$alternateViewList| nil)
    (setq |$forceAdd| t)
    (setq |$ConditionalOperators| nil)
    (mapcar #'reduce
      (loop for a in argl for s in sigl do
        (fn a (augmentSig s (findExtras a target)))))))

```

5.2.70 defun genDomainViewList0

[getDomainViewList p??]

— defun genDomainViewList0 —

```

(defun |genDomainViewList0| (id catlist)
  (|genDomainViewList| id catlist t))

```

5.2.71 defun genDomainViewList

[qcdr p??]
 [isCategoryForm p??]
 [genDomainView p197]
 [genDomainViewList p196]
 [\$EmptyEnvironment p??]

— defun genDomainViewList —

```
(defun |genDomainViewList| (id catlist firsttime)
  (declare (special |$EmptyEnvironment|) (ignore firsttime))
  (cond
    ((null catlist) nil)
    ((and (consp catlist) (eq (qrest catlist) nil)
      (null (|isCategoryForm| (first catlist) |$EmptyEnvironment|)))
      nil)
    (t
     (cons
      (|genDomainView| id (first catlist) '|getDomainView|)
      (|genDomainViewList| id (rest catlist) nil))))))
```

5.2.72 defun genDomainView

```
[genDomainOps p198]
[qcar p??]
[qcdr p??]
[augModemapsFromCategory p237]
[mkDomainConstructor p??]
[member p??]
[$e p??]
[$getDomainCode p??]
```

— defun genDomainView —

```
(defun |genDomainView| (name c viewSelector)
  (let (code cd)
    (declare (special |$getDomainCode| |$e|))
    (cond
      ((and (consp c) (eq (qfirst c) 'category) (consp (qrest c)))
       (|genDomainOps| name name c))
      (t
       (setq code
        (if (and (consp c) (eq (qfirst c) '|SubsetCategory|)
          (consp (qrest c)) (consp (qcddr c))
          (eq (qcddr c) nil))
          (second c)
          c))
       (setq |$e| (|augModemapsFromCategory| name nil c |$e|))
       (setq cd
        (list 'let name (list viewSelector name (|mkDomainConstructor| code))))
       (unless (|member| cd |$getDomainCode|)
        (setq |$getDomainCode| (cons cd |$getDomainCode|)))
       name))))
```

5.2.73 defun genDomainOps

```
[getOperationAlist p242]
[substNames p243]
[mkq p??]
[mkDomainConstructor p??]
[addModemap p245]
[$e p??]
[$ConditionalOperators p??]
[$getDomainCode p??]
```

— defun genDomainOps —

```
(defun |genDomainOps| (viewName dom cat)
  (let (siglist oplist cd i)
    (declare (special |$e| |$ConditionalOperators| |$getDomainCode|))
    (setq oplist (|getOperationAlist| dom dom cat))
    (setq siglist (loop for lst in oplist collect (first lst)))
    (setq oplist (|substNames| dom viewName dom oplist))
    (setq cd
      (list 'let viewName
        (list 'mkOpVec| dom
          (cons 'list
            (loop for opsig in siglist
              collect
                (list 'list (mkq (first opsig))
                  (cons 'list
                    (loop for mode in (rest opsig)
                      collect (|mkDomainConstructor| mode))))))))))
    (setq |$getDomainCode| (cons cd |$getDomainCode|))
    (setq i 0)
    (loop for item in oplist do
      (if (|member| (first item) |$ConditionalOperators|)
        (setq |$e| (|addModemap| (caar item) dom (cadar item) nil
          (list 'elt viewName (incf i)) |$e|))
        (setq |$e| (|addModemap| (caar item) dom (cadar item) (second item)
          (list 'elt viewName (incf i)) |$e|))))
    viewName))
```

5.2.74 defun mkOpVec

```

[getPrincipalView p??]
[getOperationAlistFromLisplib p??]
[opOf p??]
[length p??]
[assq p??]
[assoc p??]
[qcar p??]
[qcdr p??]
[sublis p??]
[AssocBarGensym p200]
[msubst p??]
[$FormalMapVariableList p242]
[Undef p??]

```

— defun mkOpVec —

```

(defun |mkOpVec| (dom siglist)
  (let (substargs oplist ops u noplist i tmp1)
    (declare (special |$FormalMapVariableList| |Undef|))
    (setq dom (|getPrincipalView| dom))
    (setq substargs
      (cons (cons '$ (elt dom 0))
            (loop for a in |$FormalMapVariableList| for x in (rest (elt dom 0))
                  collect (cons a x))))
    (setq oplist (|getOperationAlistFromLisplib| (|opOf| (elt dom 0))))
    (setq ops (make-array (|#| siglist)))
    (setq i -1)
    (loop for opSig in siglist do
      (incf i)
      (setq u (assq (first opSig) oplist))
      (setq tmp1 (|assoc| (second opSig) u))
      (cond
        ((and (consp tmp1) (consp (qrest tmp1))
              (consp (qcddr tmp1)) (consp (qcdddr tmp1))
              (eq (qcdddr tmp1) nil)
              (eq (qfourth tmp1) 'elt))
         (setelt ops i (elt dom (second tmp1))))
        (t
         (setq noplist (sublis substargs u))
         (setq tmp1
           (|AssocBarGensym| (msubst (elt dom 0) '$ (second opSig)) noplist))
         (cond
           ((and (consp tmp1) (consp (qrest tmp1)) (consp (qcddr tmp1))
                 (consp (qcdddr tmp1))
                 (eq (qcdddr tmp1) nil)
                 (eq (qfourth tmp1) 'elt))
            (setelt ops i (elt dom (second tmp1))))
           (t
            (setelt ops i (elt dom 0)))))))

```

```

      (setelt ops i (elt dom (second tmp1))))
    (t
      (setelt ops i (cons |Undef| (cons (list (elt dom 0) i) opSig)))))))))
  ops))

```

5.2.75 defun AssocBarGensym

[EqualBarGensym p224]

— defun AssocBarGensym —

```

(defun |AssocBarGensym| (key z)
  (loop for x in z
    do (when (and (consp x) (|EqualBarGensym| key (car x))) (return x))))

```

5.2.76 defun compDefWhereClause

```

[qcar p??]
[qcdr p??]
[getmode p??]
[userError p??]
[concat p??]
[lassoc p??]
[pairList p??]
[union p??]
[listOfIdentifiersIn p??]
[delete p??]
[orderByDependency p202]
[assocleft p??]
[assocright p??]
[comp p497]
[$sigAlist p??]
[$predAlist p??]

```

— defun compDefWhereClause —

```

(defun |compDefWhereClause| (arg mode env)
  (labels (
    (transformType (x)
      (declare (special |$sigAlist|))

```

```

(cond
  ((atom x) x)
  ((and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
        (consp (qcddr x)) (eq (qcdddr x) nil))
   (setq |$sigAlist|
         (cons (cons (second x) (transformType (third x)))
               |$sigAlist|))
   x)
  ((and (consp x) (eq (qfirst x) '|Record|)) x)
  (t
   (cons (first x)
         (loop for y in (rest x)
              collect (transformType y))))))
(removeSuchthat (x)
  (declare (special |$predAlist|))
  (if (and (consp x) (eq (qfirst x) '|\\|') (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
      (progn
        (setq |$predAlist| (cons (cons (second x) (third x)) |$predAlist|))
        (second x))
      x))
(fetchType (a x env form)
  (if x
      x
      (or (|getmodel| a env)
          (|userError| (|concat|
                        "There is no mode for argument" a "of function" (first form))))))
(addSuchthat (x y)
  (let (p)
    (declare (special |$predAlist|))
    (if (setq p (lassoc x |$predAlist|)) (list '|\\| y p) y)))
)
(let (|$sigAlist| |$predAlist| form signature specialCases body sigList
      argList argSigAlist argDepAlist varList whereList formxx signaturex
      defform formx)
  (declare (special |$sigAlist| |$predAlist|))
  ; form is lhs (f a1 ... an) of definition; body is rhs;
  ; signature is (t0 t1 ... tn) where t0= target type, ti=type of ai, i > 0;
  ; specialCases is (NIL l1 ... ln) where li is list of special cases
  ; which can be given for each ti
  ;
  ; removes declarative and assignment information from form and
  ; signature, placing it in list L, replacing form by ("where",form',:L),
  ; signature by a list of NILs (signifying declarations are in e)
  (setq form (second arg))
  (setq signature (third arg))
  (setq specialCases (fourth arg))
  (setq body (fifth arg))
  (setq |$sigAlist| nil)
  (setq |$predAlist| nil)

```



```

; 1. create sigList= list of all signatures which have embedded
;   declarations moved into global variable $sigAlist
(setq sigList
  (loop for a in (rest form) for x in (rest signature)
    collect (transformType (fetchType a x env form))))
; 2. replace each argument of the form (| x p) by x, recording
;   the given predicate in global variable $predAlist
(setq argList
  (loop for a in (rest form)
    collect (removeSuchthat a)))
(setq argSigAlist (append |$sigAlist| (|pairList| argList sigList)))
(setq argDepAlist
  (loop for pear in argSigAlist
    collect
      (cons (car pear)
        (|union| (|listOfIdentifiersIn| (cdr pear))
          (|delete| (car pear)
            (|listOfIdentifiersIn| (lassoc (car pear) |$predAlist|)))))))
; 3. obtain a list of parameter identifiers (x1 .. xn) ordered so that
;   the type of xi is independent of xj if i < j
(setq varList
  (|orderByDependency| (assocleft argDepAlist) (assocright argDepAlist)))
; 4. construct a WhereList which declares and/or defines the xi's in
;   the order constructed in step 3
(setq whereList
  (loop for x in varList
    collect (addSuchthat x (list '|:| x (lassoc x argSigAlist)))))
(setq formxx (cons (car form) argList))
(setq signaturex
  (cons (car signature)
    (loop for x in (rest signature) collect nil)))
(setq defform (list 'def formxx signaturex specialCases body))
(setq formx (cons '|where| (cons defform whereList)))
; 5. compile new ('DEF,("where",form',:WhereList),..) where
;   all argument parameters of form' are bound/declared in WhereList
(|comp| formx mode env)))

```

5.2.77 defun orderByDependency

```

[say p??]
[userError p??]
[intersection p??]
[member p??]
[remdup p??]

```

— defun orderByDependency —

```
(defun |orderByDependency| (vl dl)
  (let (selfDependents fatalError newl orderedVarList vlp dlp)
    (setq selfDependents
      (loop for v in vl for d in dl
        when (member v d)
        collect v))
    (loop for v in vl for d in dl
      when (member v d)
      do (say v "depends on itself")
        (setq fatalError t)))
    (cond
      (fatalError (|userError| "Parameter specification error"))
      (t
        (loop until (null vl) do
          (setq newl
            (loop for v in vl for d in dl
              when (null (|intersection| d vl))
              collect v))
          (if (null newl)
            (setq vl nil) ; force loop exit
            (progn
              (setq orderedVarList (append newl orderedVarList))
              (setq vlp (setdifference vl newl))
              (setq dlp
                (loop for x in vl for d in dl
                  when (|member| x vlp)
                  collect (setdifference d newl)))
              (setq vl vlp)
              (setq dl dlp))))))
        (when (and newl orderedVarList) (remdup (nreverse orderedVarList)))))))
```

—————

5.3 Code optimization routines

5.3.1 defun optimizeFunctionDef

```
[qcar p??]
[qcdr p??]
[rplac p??]
[sayBrightlyI p??]
[optimize p205]
[pp p??]
[bright p??]
```

[\$reportOptimization p??]

— defun optimizeFunctionDef —

```
(defun |optimizeFunctionDef| (def)
  (labels (
    (fn (x g)
      (cond
        ((and (consp x) (eq (qfirst x) 'throw) (consp (qrest x))
          (equal (qsecond x) g))
          (|rplac| (car x) 'return)
          (|rplac| (cdr x)
            (replaceThrowByReturn (qcddr x) g)))
        ((atom x) nil)
        (t
          (replaceThrowByReturn (car x) g)
          (replaceThrowByReturn (cdr x) g))))
    (replaceThrowByReturn (x g)
      (fn x g)
        x)
    (removeTopLevelCatch (body)
      (if (and (consp body) (eq (qfirst body) 'catch) (consp (qrest body))
        (consp (qcddr body)) (eq (qcdddr body) nil))
        (removeTopLevelCatch
          (replaceThrowByReturn
            (qthird body) (qsecond body)))
        body)))
    (let (defp name slamOrLam args body bodyp)
      (declare (special |$reportOptimization|))
      (when |$reportOptimization|
        (|sayBrightlyI| (|bright| "Original LISP code:"))
        (|pp| def))
      (setq defp (|optimize| (copy def)))
      (when |$reportOptimization|
        (|sayBrightlyI| (|bright| "Optimized LISP code:"))
        (|pp| defp)
        (|sayBrightlyI| (|bright| "Final LISP code:")))
      (setq name (car defp))
      (setq slamOrLam (caadr defp))
      (setq args (cadadr defp))
      (setq body (car (cddadr defp)))
      (setq bodyp (removeTopLevelCatch body))
      (list name (list slamOrLam args bodyp))))))
```

— — —

5.3.2 defun optimize

```
[qcar p??]
[qcdr p??]
[optimize p205]
[say p??]
[prettyprint p??]
[rplac p??]
[optIF2COND p207]
[getl p??]
[subrname p208]
```

— defun optimize —

```
(defun |optimize| (x)
  (labels (
    (opt (x)
      (let (argl body a y op)
        (cond
          ((atom x) nil)
          ((eq (setq y (car x)) 'quote) nil)
          ((eq y 'closedfn) nil)
          ((and (consp y) (consp (qfirst y)) (eq (qcaar y) 'xlam)
                (consp (qcдар y)) (consp (qcddar y))
                (eq (qcdddar y) nil))
           (setq argl (qcadar y))
           (setq body (qcaddar y))
           (setq a (qrest y))
           (|optimize| (cdr x))
           (cond
             ((eq argl '|ignore|) (rplac (car x) body))
             (t
              (when (null (<= (length argl) (length a)))
                (say "length mismatch in XLAM expression")
                (prettyprint y))
              (rplac (car x)
                    (|optimize|
                     (|optXLAMCond|
                      (sublis (|pairList| argl a) body)))))))
          ((atom y)
           (|optimize| (cdr x))
           (cond
             ((eq y '|true|) (rplac (car x) '''T))
             ((eq y '|false|) (rplac (car x) nil))))
          ((eq (car y) 'if)
           (rplac (car x) (|optIF2COND| y))
           (setq y (car x))
           (when (setq op (getl (|subrname| (car y)) 'optimize))
             (|optimize| (cdr x))
```

```

      (rplac (car x) (funcall op (|optimize| (car x)))))
    ((setq op (get1 (|subrname| (car y)) 'optimize))
      (|optimize| (cdr x))
      (rplac (car x) (funcall op (|optimize| (car x)))))
    (t
      (rplac (car x) (|optimize| (car x)))
      (|optimize| (cdr x)))))
  (opt x)
  x))

```

5.3.3 defun optXLAMCond

```

[optCONDtail p206]
[optPredicateIfTrue p207]
[optXLAMCond p206]
[qcar p??]
[qcdr p??]
[rplac p??]

```

— defun optXLAMCond —

```

(defun |optXLAMCond| (x)
  (cond
    ((and (consp x) (eq (qfirst x) 'cond) (consp (qrest x))
      (consp (qsecond x)) (consp (qcddadr x))
      (eq (qcddadr x) nil))
      (if (|optPredicateIfTrue| (qcaadr x))
        (qcadadr x)
        (cons 'cond (cons (qsecond x) (|optCONDtail| (qcddr x)))))
      ((atom x) x)
    (t
      (rplac (car x) (|optXLAMCond| (car x)))
      (rplac (cdr x) (|optXLAMCond| (cdr x)))
      x)))

```

5.3.4 defun optCONDtail

```

[optCONDtail p206]
[$true p??]

```

— defun optCONDtail —

```
(defun |optCONDtail| (z)
  (declare (special |$true|))
  (when z
    (cond
      ((|optPredicateIfTrue| (caar z)) (list (list |$true| (cadar z))))
      ((null (cdr z)) (list (car z) (list |$true| (list '|CondError|))))
      (t (cons (car z) (|optCONDtail| (cdr z)))))))
```

5.3.5 defvar \$BasicPredicates

If these predicates are found in an expression the code optimizer routine `optPredicateIfTrue` then `optXLAM` will replace the call with the argument. This is used for predicates that test the type of their argument so that, for instance, a call to `integerp` on an integer will be replaced by that integer if it is true. This represents a simple kind of compile-time type evaluation.

— **initvars** —

```
(defvar |$BasicPredicates| '(integerp stringp floatp))
```

5.3.6 defun optPredicateIfTrue

[`$BasicPredicates` p207]

— **defun optPredicateIfTrue** —

```
(defun |optPredicateIfTrue| (p)
  (declare (special |$BasicPredicates|))
  (cond
    ((and (consp p) (eq (qfirst p) 'quote)) T)
    ((and (consp p) (consp (qrest p)) (eq (qcddr p) nil)
      (member (qfirst p) |$BasicPredicates|) (funcall (qfirst p) (qsecond p)))
      t)
    (t nil)))
```

5.3.7 defun optIF2COND

[`optIF2COND` p207]

[`$true p??`]

— defun optIF2COND —

```

(defun |optIF2COND| (arg)
  (let (a b c)
    (declare (special |$true|))
    (setq a (cadr arg))
    (setq b (caddr arg))
    (setq c (cadddr arg))
    (cond
      ((eq b '|noBranch|) (list 'cond (list (list 'null a ) c)))
      ((eq c '|noBranch|) (list 'cond (list a b)))
      ((and (consp c) (eq (qfirst c) 'if))
        (cons 'cond (cons (list a b) (cdr (|optIF2COND| c)))))
      ((and (consp c) (eq (qfirst c) 'cond))
        (cons 'cond (cons (list a b) (qrest c)))))
    (t
      (list 'cond (list a b) (list |$true| c))))))

```

5.3.8 defun subrname

```

[identp p??]
[compiled-function-p p??]
[mbpip p??]
[bpname p??]

```

— defun subrname —

```

(defun |subrname| (u)
  (cond
    ((identp u) u)
    ((or (compiled-function-p u) (mbpip u)) (bpname u))
    (t nil)))

```

5.3.9 Special case optimizers

Optimization functions are called through the OPTIMIZE property on the symbol property list. The current list is:

call	optCall
seq	optSEQ

<code>eq</code>	<code>optEQ</code>
<code>minus</code>	<code>optMINUS</code>
<code>qsminus</code>	<code>optQSMINUS</code>
<code>-</code>	<code>opt-</code>
<code>lessp</code>	<code>optLESSP</code>
<code>spadcall</code>	<code>optSPADCALL</code>
<code> </code>	<code>optSuchthat</code>
<code>catch</code>	<code>optCatch</code>
<code>cond</code>	<code>optCond</code>
<code> mkRecord </code>	<code>optMkRecord</code>
<code>recordelt</code>	<code>optRECORDELT</code>
<code>setrecordelt</code>	<code>optSETRECORDELT</code>
<code>recordcopy</code>	<code>optRECORDCOPY</code>

Be aware that there are case-sensitivity issues. When found in the s-expression, each symbol in the left column will call a custom optimization routine in the right column. The optimization routines are below. Note that each routine has a special chunk in postvars using `eval-when` to set the property list at load time.

These optimizations are done destructively. That is, they modify the function in-place using `rplac`.

Not all of the optimization routines are called through the property list. Some are called only from other optimization routines, e.g. `optPackageCall`.

5.3.10 `defplist optCall`

— **postvars** —

```
(eval-when (eval load)
  (setf (get '|call| 'optimize) '|optCall|))
```

—————

5.3.11 `defun Optimize` “call” expressions

```
[optimize p205]
[rplac p??]
[optPackageCall p210]
[optCallSpecially p211]
[systemErrorHere p??]
[$QuickCode p??]
[$bootStrapMode p??]
```

— **defun optCall** —


```

(defun |optCall| (x)
  (let (u tmp1 fn a name q r n w)
    (declare (special |$QuickCode| |$bootStrapMode|))
    (setq u (cdr x))
    (setq x (|optimize| (list u)))
    (cond
      ((atom (car x)) (car x))
      (t
       (setq tmp1 (car x))
       (setq fn (car tmp1))
       (setq a (cdr tmp1))
       (cond
         ((atom fn) (rplac (cdr x) a) (rplac (car x) fn))
         ((and (consp fn) (eq (qfirst fn) 'pac)) (|optPackageCall| x fn a))
         ((and (consp fn) (eq (qfirst fn) '|applyFun|)
              (consp (qrest fn)) (eq (qcddr fn) nil))
          (setq name (qsecond fn))
          (rplac (car x) 'spadcall)
          (rplac (cdr x) (append a (cons name nil))))
         x)
        ((and (consp fn) (consp (qrest fn)) (consp (qcddr fn))
              (eq (qcddr fn) nil)
              (member (qfirst fn) '(elt qrefelt const)))
         (setq q (qfirst fn))
         (setq r (qsecond fn))
         (setq n (qthird fn))
         (cond
           ((and (null |$bootStrapMode|) (setq w (|optCallSpecially| q x n r)))
            w)
           ((eq q 'const)
            (list 'spadConstant| r n))
           (t
            (rplac (car x) 'spadcall)
            (when |$QuickCode| (rplaca fn 'qrefelt))
            (rplac (cdr x) (append a (list fn)))
            x)))
        (t (|systemErrorHere| "optCall"))))))))

```

5.3.12 defun optPackageCall

[rplaca p??]
[rplacd p??]

— defun optPackageCall —

```
(defun |optPackageCall| (x arg2 arglist)
```

```
(let (packageVariableOrForm functionName)
  (setq packageVariableOrForm (second arg2))
  (setq functionName (third arg2))
  (rplaca x functionName)
  (rplacd x (append arglist (list packageVariableOrForm)))
  x))
```

5.3.13 defun optCallSpecially

```
[lassoc p??]
[kar p??]
[get p??]
[opOf p??]
[optSpecialCall p212]
[$specialCaseKeyList p??]
[$getDomainCode p??]
[$optimizableConstructorNames p??]
[$e p??]
```

— defun optCallSpecially —

```
(defun |optCallSpecially| (q x n r)
  (declare (ignore q))
  (labels (
    (lookup (a z)
      (let (zp)
        (when z
          (setq zp (car z))
          (setq z (cdr x))
          (if (and (consp zp) (eq (qfirst zp) 'let) (consp (qrest zp))
              (equal (qsecond zp) a) (consp (qcddr zp)))
              (qthird zp)
              (lookup a z))))))
    (lookup a z))))))
  (let (tmp1 op y prop yy)
    (declare (special |$specialCaseKeyList| |$getDomainCode| |$e|
                      |$optimizableConstructorNames|))
    (cond
      ((setq y (lassoc r |$specialCaseKeyList|))
       (|optSpecialCall| x y n))
      ((member (kar r) |$optimizableConstructorNames|)
       (|optSpecialCall| x r n))
      ((and (setq y (|get| r '|value| |$e|))
            (member (|opOf| (car y)) |$optimizableConstructorNames|))
       (|optSpecialCall| x (car y) n))
      ((and (setq y (lookup r |$getDomainCode|))
```

```

(progn
  (setq tmp1 y)
  (setq op (first tmp1))
  (setq y (second tmp1))
  (setq prop (third tmp1))
  tmp1)
(setq yy (lassoc y |$specialCaseKeyList|)))
(|optSpecialCall| x (list op yy prop) n))
(t nil))))

```

5.3.14 defun optSpecialCall

```

[optCallEval p213]
[function p??]
[keyedSystemError p??]
[mkq p??]
[getl p??]
[compileTimeBindingOf p213]
[rplac p??]
[optimize p205]
[rplacw p??]
[rplaca p??]
[$QuickCode p??]
[$Undef p??]

```

— defun optSpecialCall —

```

(defun |optSpecialCall| (x y n)
  (let (yval args tmp1 fn a)
    (declare (special |$QuickCode| |Undef|))
    (setq yval (|optCallEval| y))
    (cond
      ((eq (caaar x) 'const)
        (cond
          ((equal (kar (elt yval n)) (|function| |Undef|))
            (|keyedSystemError| 'S2GE0016
              (list "optSpecialCall" "invalid constant")))
          (t (mkq (elt yval n)))))
      ((setq fn (getl (|compileTimeBindingOf| (car (elt yval n))) '|SPADreplace|))
        (|rplac| (cdr x) (cdar x))
        (|rplac| (car x) fn)
        (when (and (consp fn) (eq (qfirst fn) 'xlam))
          (setq x (car (|optimize| (list x)))))
        (if (and (consp x) (eq (qfirst x) 'equal)) (progn (setq args (qrest x)) t))
        (rplacw x (def-equal args))

```

```

      x))
(t
  (setq tmp1 (car x))
  (setq fn (car tmp1))
  (setq a (cdr tmp1))
  (rplac (car x) 'spadcall)
  (when |$QuickCode| (rplaca fn 'qrefelt))
  (rplac (cdr x) (append a (list fn)))
  x)))

```

5.3.15 defun compileTimeBindingOf

```

[bpname p??]
[keyedSystemError p??]
[moan p??]

```

— defun compileTimeBindingOf —

```

(defun |compileTimeBindingOf| (u)
  (let (name)
    (cond
      ((null (setq name (bpname u)))
       (|keyedSystemError| 'S2000001 (list u)))
      ((eq name '|Undef|)
       (moan "optimiser found unknown function"))
      (t name))))

```

5.3.16 defun optCallEval

```

[qcar p??]
[List p??]
[Integer p??]
[Vector p??]
[PrimitiveArray p??]
[FactoredForm p??]
[Matrix p??]
[eval p??]

```

— defun optCallEval —

```

(defun |optCallEval| (u)

```

```

(cond
  ((and (consp u) (eq (qfirst u) '|List|))
    (|List| (|Integer|)))
  ((and (consp u) (eq (qfirst u) '|Vector|))
    (|Vector| (|Integer|)))
  ((and (consp u) (eq (qfirst u) '|PrimitiveArray|))
    (|PrimitiveArray| (|Integer|)))
  ((and (consp u) (eq (qfirst u) '|FactoredForm|))
    (|FactoredForm| (|Integer|)))
  ((and (consp u) (eq (qfirst u) '|Matrix|))
    (|Matrix| (|Integer|)))
  (t
    (|eval| u))))

```

5.3.17 defplist optSEQ

— postvars —

```

(eval-when (eval load)
  (setf (get 'seq 'optimize) '|optSEQ|))

```

5.3.18 defun optSEQ

— defun optSEQ —

```

(defun |optSEQ| (arg)
  (labels (
    (tryToRemoveSEQ (z)
      (if (and (consp z) (eq (qfirst z) 'seq) (consp (qrest z))
        (eq (qcddr z) nil) (consp (qsecond z))
        (consp (qcadadr z))
        (eq (qcddadr z) nil)
        (member (qcaadr z) '(exit return throw)))
      (qcadadr z)
      z))
    (SEQToCOND (z)
      (let (transform before aft)
        (setq transform
          (loop for x in z

```

```

while
  (and (consp x) (eq (qfirst x) 'cond) (consp (qrest x))
        (eq (qcddr x) nil) (consp (qsecond x))
        (consp (qcaddr x))
        (eq (qcddadr x) nil)
        (consp (qcadadr x))
        (eq (qfirst (qcadadr x)) 'exit)
        (consp (qrest (qcadadr x)))
        (eq (qcddr (qcadadr x)) nil))
collect
  (list (qcaadr x)
        (qsecond (qcadadr x))))
(setq before (take (|#| transform) z))
(setq aft (|after| z before))
(cond
  ((null before) (cons 'seq aft))
  ((null aft)
   (cons 'cond (append transform (list '(t (|conderr|))))))
  (t
   (cons 'cond (append transform
                        (list (list 't (|optSEQ| (cons 'seq aft))))))))
(getRidOfTemps (z))
(let (g x r)
  (cond
    ((null z) nil)
    ((and (consp z) (consp (qfirst z)) (eq (qcaar z) 'let)
          (consp (qcdr z)) (consp (qcddr z))
          (gensymp (qcadr z))
          (> 2 (|numOfOccurencesOf| (qcadr z) (qrest z))))
     (setq g (qcadr z))
     (setq x (qcaddr z))
     (setq r (qrest z))
     (getRidOfTemps (msubst x g r)))
    ((eq (car z) '|/throwAway|)
     (getRidOfTemps (cdr z)))
    (t
     (cons (car z) (getRidOfTemps (cdr z))))))
(tryToRemoveSEQ (SEQToCOND (getRidOfTemps (cdr arg)))))

```

5.3.19 defplist optEQ

— postvars —

```

(eval-when (eval load)
  (setf (get 'eq 'optimize) '|optEQ|))

```

5.3.20 defun optEQ

— defun optEQ —

```
(defun |optEQ| (u)
  (let (z r)
    (cond
      ((and (consp u) (eq (qfirst u) 'eq) (consp (qrest u))
            (consp (qcddr u)) (eq (qcdddr u) nil))
       (setq z (qsecond u))
       (setq r (qthird u))
       ; That undoes some weird work in Boolean to do with the definition of true
       (if (and (numberp z) (numberp r))
           (list 'quote (eq z r))
           u))
      (t u))))
```

5.3.21 defplist optMINUS

— postvars —

```
(eval-when (eval load)
  (setf (get 'minus 'optimize) '|optMINUS|))
```

5.3.22 defun optMINUS

— defun optMINUS —

```
(defun |optMINUS| (u)
  (let (v)
    (cond
      ((and (consp u) (eq (qfirst u) 'minus) (consp (qrest u))
            (eq (qcddr u) nil))
```

```

      (setq v (qsecond u))
      (cond ((numberp v) (- v)) (t u)))
    (t u)))

```

5.3.23 defplist optQSMINUS

— postvars —

```

(eval-when (eval load)
  (setf (get 'qsmminus 'optimize) '|optQSMINUS|))

```

5.3.24 defun optQSMINUS

— defun optQSMINUS —

```

(defun |optQSMINUS| (u)
  (let (v)
    (cond
      ((and (consp u) (eq (qfirst u) 'qsmminus) (consp (qrest u))
            (eq (qcddr u) nil))
       (setq v (qsecond u))
       (cond ((numberp v) (- v)) (t u)))
      (t u))))

```

5.3.25 defplist opt-

— postvars —

```

(eval-when (eval load)
  (setf (get '- 'optimize) '|opt-|))

```

5.3.26 defun opt-

— defun opt- —

```
(defun |opt-| (u)
  (let (v)
    (cond
      ((and (consp u) (eq (qfirst u) '-') (consp (qrest u))
        (eq (qcddr u) NIL))
       (setq v (qsecond u))
       (cond ((numberp v) (- v)) (t u)))
      (t u))))
```

5.3.27 defplist optLESSP

— postvars —

```
(eval-when (eval load)
  (setf (get 'lessp 'optimize) '|optLESSP|))
```

5.3.28 defun optLESSP

— defun optLESSP —

```
(defun |optLESSP| (u)
  (let (a b)
    (cond
      ((and (consp u) (eq (qfirst u) 'lessp) (consp (qrest u))
        (consp (qcddr u))
        (eq (qcdddr u) nil))
       (setq a (qsecond u))
       (setq b (qthird u))
       (if (eql b 0)
          (list 'minusp a)
          (list '> b a)))
      (t u))))
```

5.3.29 defplist optSPADCALL

— postvars —

```
(eval-when (eval load)
  (setf (get 'spadcall 'optimize) '|optSPADCALL|))
```

—————

5.3.30 defun optSPADCALL

[optCall p209]
 [\$InteractiveMode p??]

— defun optSPADCALL —

```
(defun |optSPADCALL| (form)
  (let (fun arg1 tmp1 dom slot)
    (declare (special |$InteractiveMode|))
    (setq arg1 (cdr form))
    (cond
      ; last arg is function/env, but may be a form
      ((null |$InteractiveMode|) form)
      ((and (consp arg1)
            (progn (setq tmp1 (reverse arg1)) t)
            (consp tmp1))
       (setq fun (qfirst tmp1))
       (setq arg1 (qrest tmp1))
       (setq arg1 (nreverse arg1))
       (cond
         ((and (consp fun)
               (or (eq (qfirst fun) 'elt) (eq (qfirst fun) 'lispelt))
               (progn
                 (and (consp (qrest fun))
                     (progn
                       (setq dom (qsecond fun))
                       (and (consp (qcddr fun))
                           (eq (qcdddr fun) nil)
                           (progn
                             (setq slot (qthird fun))
                             t))))))
          (|optCall| (cons '|call| (cons (list 'elt dom slot) arg1))))
         (t form)))
      (t form))))
```

—————

5.3.31 defplist optSuchthat

— postvars —

```
(eval-when (eval load)
  (setf (get '|\\|' 'optimize) '|optSuchthat|))
```

—————

5.3.32 defun optSuchthat

— defun optSuchthat —

```
(defun |optSuchthat| (arg)
  (cons 'suchthat (cdr arg)))
```

—————

5.3.33 defplist optCatch

— postvars —

```
(eval-when (eval load)
  (setf (get 'catch 'optimize) '|optCatch|))
```

—————

5.3.34 defun optCatch

```
[qcar p??]
[qcdr p??]
[rplac p??]
[optimize p205]
[$InteractiveMode p??]
```

— defun optCatch —

```
(defun |optCatch| (x)
  (labels (
```

```

(changeThrowToExit (s g)
  (cond
    ((or (atom s) (member (car s) '(quote seq repeat collect))) nil)
    ((and (consp s) (eq (qfirst s) 'throw) (consp (qrest s))
      (equal (qsecond s) g))
      (|rplac| (car s) 'exit)
      (|rplac| (cdr s) (qcddr s)))
    (t
      (changeThrowToExit (car s) g)
      (changeThrowToExit (cdr s) g))))
(hasNoThrows (a g)
  (cond
    ((and (consp a) (eq (qfirst a) 'throw) (consp (qrest a))
      (equal (qsecond a) g))
      nil)
    ((atom a) t)
    (t
      (and (hasNoThrows (car a) g)
        (hasNoThrows (cdr a) g)))))
(changeThrowToGo (s g)
  (let (u)
    (cond
      ((or (atom s) (eq (car s) 'quote)) nil)
      ((and (consp s) (eq (qfirst s) 'throw) (consp (qrest s))
        (equal (qsecond s) g) (consp (qcddr s))
        (eq (qcdddr s) nil))
        (setq u (qthird s))
        (changeThrowToGo u g)
        (|rplac| (car s) 'progn)
        (|rplac| (cdr s) (list (list 'let (cadr g) u) (list 'go (cadr g)))))
      (t
        (changeThrowToGo (car s) g)
        (changeThrowToGo (cdr s) g)))))
(let (g tmp2 u s tmp6 a)
  (declare (special |$InteractiveMode|))
  (setq g (cadr x))
  (setq a (caddr x))
  (cond
    (|$InteractiveMode| x)
    ((atom a) a)
    (t
      (cond
        ((and (consp a) (eq (qfirst a) 'seq) (consp (qrest a))
          (progn (setq tmp2 (reverse (qrest a))) t)
          (consp tmp2) (consp (qfirst tmp2)) (eq (qcaar tmp2) 'throw)
          (consp (qcadar tmp2))
          (equal (qcadar tmp2) g)
          (consp (qcddar tmp2))
          (eq (qcdddar tmp2) nil))
          (setq u (qcaddar tmp2))

```

```

(setq s (qrest tmp2))
(setq s (nreverse s))
(changeThrowToExit s g)
(|rplac| (cdr a) (append s (list (list 'exit u))))
(setq tmp6 (|optimize| x))
(setq a (caddr tmp6)))
(cond
  ((hasNoThrows a g)
   (|rplac| (car x) (car a))
   (|rplac| (cdr x) (cdr a)))
  (t
   (changeThrowToGo a g)
   (|rplac| (car x) 'seq)
   (|rplac| (cdr x)
    (list (list 'exit a) (cadr g) (list 'exit (cadr g))))))
x))))

```

5.3.35 defplist optCond

— postvars —

```

(eval-when (eval load)
  (setf (get 'cond 'optimize) '|optCond|))

```

5.3.36 defun optCond

```

[qcar p??]
[qcdr p??]
[rplacd p??]
[TruthP p241]
[EqualBarGensym p224]
[rplac p??]

```

— defun optCond —

```

(defun |optCond| (x)
  (let (z p1 p2 c3 c1 c2 a result)
    (setq z (cdr x))
    (when
      (and (consp z) (consp (qrest z)) (eq (qcddr z) nil)

```

```

      (consp (qsecond z)) (consp (qcdadr z))
      (eq (qrest (qcdadr z)) nil)
      (|TruthP| (qcaadr z))
      (consp (qcadadr z))
      (eq (qfirst (qcadadr z)) 'cond))
    (rplacd (cdr x) (qrest (qcadadr z))))
  (cond
    ((and (consp z) (consp (qfirst z)) (consp (qrest z)) (consp (qsecond z)))
      (setq p1 (qcaar z))
      (setq c1 (qcdar z))
      (setq p2 (qcaadr z))
      (setq c2 (qcdadr z))
      (when
        (or (and (consp p1) (eq (qfirst p1) 'null) (consp (qrest p1))
          (eq (qcddr p1) nil)
          (equal (qsecond p1) p2))
          (and (consp p2) (eq (qfirst p2) 'null) (consp (qrest p2))
          (eq (qcddr p2) nil)
          (equal (qsecond p2) p1)))
        (setq z (list (cons p1 c1) (cons 't c2)))
        (rplacd x z))
      (when
        (and (consp c1) (eq (qrest c1) nil) (equal (qfirst c1) 'nil)
          (equal p2 't) (equal (car c2) 't))
        (if (and (consp p1) (eq (qfirst p1) 'null) (consp (qrest p1))
          (eq (qcddr p1) nil))
          (setq result (qsecond p1))
          (setq result (list 'null p1)))))
    (if result
      result
      (cond
        ((and (consp z) (consp (qfirst z)) (consp (qrest z)) (consp (qsecond z))
          (consp (qcddr z)) (eq (qcddr z) nil)
          (consp (qthird z))
          (|TruthP| (qcaaddr z)))
          (setq p1 (qcaar z))
          (setq c1 (qcdar z))
          (setq p2 (qcaadr z))
          (setq c2 (qcdadr z))
          (setq c3 (qcdaddr z))
          (cond
            ((|EqualBarGensym| c1 c3)
              (list 'cond
                (cons (list 'or p1 (list 'null p2)) c1) (cons (list 'quote t) c2)))
            ((|EqualBarGensym| c1 c2)
              (list 'cond (cons (list 'or p1 p2) c1) (cons (list 'quote t) c3)))
            (t x)))
          (t
            (do ((y z (cdr y)))
              ((atom y) nil)

```

```

(do ()
  ((null (and (consp y) (consp (qfirst y)) (consp (qcdar y))
              (eq (qcddar y) nil) (consp (qrest y))
              (consp (qsecond y)) (consp (qcdadr y))
              (eq (qcddadr y) nil)
              (|EqualBarGensym| (qcadar y)
                                (qcadadr y)))))
  nil)
  (setq a (list 'or (qcaar y) (qcaadr y)))
  (rplac (car (car y)) a)
  (rplac (cdr y) (qcddr y)))
x))))

```

5.3.37 defun EqualBarGensym

```

[gensymp p??]
[$GensymAssoc p??]
[$GensymAssoc p??]

```

— defun EqualBarGensym —

```

(defun |EqualBarGensym| (x y)
  (labels (
    (fn (x y)
      (let (z)
        (declare (special |$GensymAssoc|))
        (cond
          ((equal x y) t)
          ((and (gensymp x) (gensymp y))
           (if (setq z (|assoc| x |$GensymAssoc|))
               (if (equal y (cdr z)) t nil)
               (progn
                 (setq |$GensymAssoc| (cons (cons x y) |$GensymAssoc|))
                 t)))
          ((null x) (and (consp y) (eq (qrest y) nil) (gensymp (qfirst y))))
          ((null y) (and (consp x) (eq (qrest x) nil) (gensymp (qfirst x))))
          ((or (atom x) (atom y)) nil)
          (t
           (and (fn (car x) (car y))
                 (fn (cdr x) (cdr y))))))))
    (let (|$GensymAssoc|)
      (declare (special |$GensymAssoc|))
      (setq |$GensymAssoc| NIL)
      (fn x y))))

```

5.3.38 defplist optMkRecord

— postvars —

```
(eval-when (eval load)
  (setf (get '|mkRecord| 'optimize) '|optMkRecord|))
```

5.3.39 defun optMkRecord

[length p??]

— defun optMkRecord —

```
(defun |optMkRecord| (arg)
  (let (u)
    (setq u (cdr arg))
    (cond
      ((and (consp u) (eq (qrest u) nil)) (list 'list (qfirst u)))
      ((eq1 (|#| u) 2) (cons 'cons u))
      (t (cons 'vector u)))))
```

5.3.40 defplist optRECORDELT

— postvars —

```
(eval-when (eval load)
  (setf (get '|recordelt| 'optimize) '|optRECORDELT|))
```

5.3.41 defun optRECORDELT

[keyedSystemError p??]

— defun optRECORDELT —


```
(defun |optRECORDELT| (arg)
  (let (name ind len)
    (setq name (cadr arg))
    (setq ind (caddr arg))
    (setq len (caddr arg))
    (cond
      ((eql len 1)
        (cond
          ((eql ind 0) (list 'qcar name))
          (t (|keyedSystemError| 'S2000002 (list ind)))))
      ((eql len 2)
        (cond
          ((eql ind 0) (list 'qcar name))
          ((eql ind 1) (list 'qcdr name))
          (t (|keyedSystemError| 'S2000002 (list ind)))))
      (t (list 'qvlt name ind)))))
```

5.3.42 defplist optSETRECORDELT

— postvars —

```
(eval-when (eval load)
  (setf (get 'setrecordelt 'optimize) '|optSETRECORDELT|))
```

5.3.43 defun optSETRECORDELT

[keyedSystemError p??]

— defun optSETRECORDELT —

```
(defun |optSETRECORDELT| (arg)
  (let (name ind len expr)
    (setq name (cadr arg))
    (setq ind (caddr arg))
    (setq len (caddr arg))
    (setq expr (car (caddr arg)))
    (cond
      ((eql len 1)
        (if (eql ind 0)
          (list 'progn (list 'rplaca name expr) (list 'qcar name))
```

```

(|keyedSystemError| 'S2000002 (list ind)))
((eql len 2)
 (cond
  ((eql ind 0)
   (list 'progn (list 'rplaca name expr) (list 'qcar name)))
  ((eql ind 1)
   (list 'progn (list 'rplacd name expr) (list 'qcdr name)))
  (t (|keyedSystemError| 'S2000002 (list ind)))))
(t
 (list 'qsetvelt name ind expr))))

```

5.3.44 defplist optRECORDCOPY

— postvars —

```

(eval-when (eval load)
 (setf (get 'recordcopy 'optimize) '|optRECORDCOPY|))

```

5.3.45 defun optRECORDCOPY

— defun optRECORDCOPY —

```

(defun |optRECORDCOPY| (arg)
 (let (name len)
  (setq name (cadr arg))
  (setq len (caddr arg))
  (cond
   ((eql len 1) (list 'list (list 'car name)))
   ((eql len 2) (list 'cons (list 'car name) (list 'cdr name)))
   (t (list 'replace (list 'make-array len) name)))))

```

5.4 Functions to manipulate modemaps

5.4.1 defun addDomain

```
[identp p??]
[qslessp p??]
[getDomainsInScope p230]
[domainMember p236]
[isLiteral p??]
[addNewDomain p231]
[getmode p??]
[isCategoryForm p??]
[isFunctor p229]
[constructor? p??]
[member p??]
[unknownTypeError p229]
```

— defun addDomain —

```
(defun |addDomain| (domain env)
  (let (s name tmp1)
    (cond
      ((atom domain)
        (cond
          ((eq domain '|$EmptyMode|) env)
          ((eq domain '|$NoValueMode|) env)
          ((or (null (identp domain))
              (and (qslessp 2 (|#| (setq s (princ-to-string domain))))
                   (eq (|char| '|#|) (elt s 0))
                   (eq (|char| '|#|) (elt s 1))))
            env)
          ((member domain (|getDomainsInScope| env)) env)
          ((|isLiteral| domain env) env)
          (t (|addNewDomain| domain env))))
      ((eq (setq name (car domain)) '|Category|) env)
      ((|domainMember| domain (|getDomainsInScope| env)) env)
      ((and (progn
              (setq tmp1 (|getmode| name env))
              (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)
                  (consp (qrest tmp1))))
            (|isCategoryForm| (second tmp1) env))
        (|addNewDomain| domain env))
      ((or (|isFunctor| name) (|constructor?| name))
        (|addNewDomain| domain env))
      (t
        (when (and (null (|isCategoryForm| domain env))
                    (null (|member| name '(|Mapping| category))))
          (|unknownTypeError| name))
```

```
env))))
```

5.4.2 defun unknownTypeError

```
[qcar p??]
[stackSemanticError p??]
```

— defun unknownTypeError —

```
(defun |unknownTypeError| (name)
  (let (op)
    (setq name
      (if (and (consp name) (setq op (qfirst name)))
          op
          name))
    (|stackSemanticError| (list '|%b| name '|%d| '|is not a known type|) nil)))
```

5.4.3 defun isFunctor

```
[opOf p??]
[identp p??]
[getdatabase p??]
[get p??]
[constructor? p??]
[updateCategoryFrameForCategory p113]
[updateCategoryFrameForConstructor p112]
[$CategoryFrame p??]
[$InteractiveMode p??]
```

— defun isFunctor —

```
(defun |isFunctor| (x)
  (let (op u prop)
    (declare (special |$CategoryFrame| |$InteractiveMode|))
    (setq op (|opOf| x))
    (cond
      ((null (identp op)) nil)
      (|$InteractiveMode|
       (if (member op '(|Union| |SubDomain| |Mapping| |Record|))
           t
           nil))))
```

```

(member (getdatabase op 'constructorkind) '(|domain| |package|))))
((setq u
  (or (|get| op '|isFunctor| |$CategoryFrame|)
      (member op '(|SubDomain| |Union| |Record|))))
  u)
(|constructor?| op)
(cond
  ((setq prop (|get| op '|isFunctor| |$CategoryFrame|)) prop)
  (t
   (if (eq (getdatabase op 'constructorkind) '|category|)
       (|updateCategoryFrameForCategory| op)
       (|updateCategoryFrameForConstructor| op))
   (|get| op '|isFunctor| |$CategoryFrame|)))
(t nil))))

```

5.4.4 defun getDomainsInScope

The way XLAMs work:

```
((XLAM ($1 $2 $3) (SETELT $1 0 $3)) X "c" V) ==> (SETELT X 0 V)
```

```

[get p??]
[$CapsuleDomainsInScope p??]
[$insideCapsuleFunctionIfTrue p??]

```

— defun getDomainsInScope —

```

(defun |getDomainsInScope| (env)
  (declare (special |$CapsuleDomainsInScope| |$insideCapsuleFunctionIfTrue|))
  (if |$insideCapsuleFunctionIfTrue|
      |$CapsuleDomainsInScope|
      (|get| '|$DomainsInScope| 'special env)))

```

5.4.5 defun putDomainsInScope

```

[getDomainsInScope p230]
[put p??]
[delete p??]
[say p??]
[member p??]

```

```
[$CapsuleDomainsInScope p??]
[$insideCapsuleFunctionIfTrue p??]
```

— **defun putDomainsInScope** —

```
(defun |putDomainsInScope| (x env)
  (let (z newValue)
    (declare (special |$CapsuleDomainsInScope| |$insideCapsuleFunctionIfTrue|))
    (setq z (|getDomainsInScope| env))
    (when (|member| x z) (say "***** Domain: " x " already in scope"))
    (setq newValue (cons x (|delete| x z)))
    (if |$insideCapsuleFunctionIfTrue|
      (progn
        (setq |$CapsuleDomainsInScope| newValue)
        env)
      (|put| '|$DomainsInScope| 'special newValue env))))
```

5.4.6 defun isSuperDomain

```
[isSubset p??]
[lassoc p??]
[opOf p??]
[get p??]
```

— **defun isSuperDomain** —

```
(defun |isSuperDomain| (domainForm domainFormp env)
  (cond
    ((|isSubset| domainFormp domainForm env) t)
    ((and (eq domainForm '|Rep|) (eq domainFormp '$)) t)
    (t (lassoc (|opOf| domainFormp) (|get| domainForm '|SubDomain| env)))))
```

5.4.7 defun addNewDomain

```
[augModemapsFromDomain p232]
```

— **defun addNewDomain** —

```
(defun |addNewDomain| (domain env)
  (|augModemapsFromDomain| domain domain env))
```

5.4.8 defun augModemapsFromDomain

```
[member p??]
[kar p??]
[getDomainsInScope p230]
[getdatabase p??]
[opOf p??]
[addNewDomain p231]
[listOrVectorElementNode p??]
[stripUnionTags p??]
[augModemapsFromDomain1 p232]
[$Category p??]
[$DummyFunctorNames p??]
```

— defun augModemapsFromDomain —

```
(defun |augModemapsFromDomain| (name functorForm env)
  (let (curDomainsInScope u innerDom)
    (declare (special |$Category| |$DummyFunctorNames|))
    (cond
      ((|member| (or (kar name) name) |$DummyFunctorNames|)
       env)
      ((or (equal name |$Category|) (|isCategoryForm| name env))
       env)
      ((|member| name (setq curDomainsInScope (|getDomainsInScope| env)))
       env)
      (t
       (when (setq u (getdatabase (|opOf| functorForm) 'superdomain))
         (setq env (|addNewDomain| (car u) env)))
       (when (setq innerDom (|listOrVectorElementNode| name))
         (setq env (|addDomain| innerDom env)))
       (when (and (consp name) (eq (qfirst name) '|Union|))
         (dolist (d (|stripUnionTags| (qrest name)))
           (setq env (|addDomain| d env))))
       (|augModemapsFromDomain1| name functorForm env))))))
```

5.4.9 defun augModemapsFromDomain1

```
[get1 p??]
[kar p??]
[addConstructorModemaps p234]
```

```
[getmode p??]
[augModemapsFromCategory p237]
[getmodeOrMapping p??]
[substituteCategoryArguments p233]
[stackMessage p??]
```

— **defun augModemapsFromDomain1** —

```
(defun |augModemapsFromDomain1| (name functorForm env)
  (let (mappingForm categoryForm functArgTypes catform)
    (cond
      ((get1 (kar functorForm) '|makeFunctionList|)
        (|addConstructorModemaps| name functorForm env))
      ((and (atom functorForm) (setq catform (|getmode| functorForm env)))
        (|augModemapsFromCategory| name functorForm catform env))
      ((setq mappingForm (|getmodeOrMapping| (kar functorForm) env))
        (when (eq (car mappingForm) '|Mapping|) (car mappingForm))
        (setq categoryForm (cadr mappingForm))
        (setq functArgTypes (cddr mappingForm))
        (setq catform
          (|substituteCategoryArguments| (cdr functorForm) categoryForm))
        (|augModemapsFromCategory| name functorForm catform env))
      (t
        (|stackMessage| (list functorForm '| is an unknown mode|))
        env))))
```

5.4.10 defun substituteCategoryArguments

```
[msubst p??]
[internl p??]
[stringimage p??]
[sublis p??]
```

— **defun substituteCategoryArguments** —

```
(defun |substituteCategoryArguments| (argl catform)
  (let (arglAssoc (i 0))
    (setq argl (msubst '$$ '$ argl))
    (setq arglAssoc
      (loop for a in argl
        collect (cons (internl '|#| (stringimage (incf i))) a)))
    (sublis arglAssoc catform)))
```

5.4.11 defun addConstructorModemaps

```
[putDomainsInScope p230]
[getl p??]
[msubst p??]
[qcar p??]
[qcdr p??]
[addModemap p245]
[$InteractiveMode p??]
```

— defun addConstructorModemaps —

```
(defun |addConstructorModemaps| (name form env)
  (let (|$InteractiveMode| functorName fn tmp1 funList op sig nsig opcode)
    (declare (special |$InteractiveMode|))
    (setq functorName (car form))
    (setq |$InteractiveMode| nil)
    (setq env (|putDomainsInScope| name env))
    (setq fn (getl functorName '|makeFunctionList|))
    (setq tmp1 (funcall fn name form env))
    (setq funList (car tmp1))
    (setq env (cadr tmp1))
    (dolist (item funList)
      (setq op (first item))
      (setq sig (second item))
      (setq opcode (third item))
      (when (and (consp opcode) (consp (qrest opcode))
                  (consp (qcddr opcode))
                  (eq (qcddr opcode) nil)
                  (eq (qfirst opcode) 'elt))
        (setq nsig (msubst '$$$ name sig))
        (setq nsig (msubst '$ '$$$ (msubst '$$ '$ nsig)))
        (setq opcode (list (first opcode) (second opcode) nsig)))
      (setq env (|addModemap| op name sig t opcode env)))
    env))
```

—

5.4.12 defun getModemap

```
[get p??]
[compApplyModemap p??]
[sublis p??]
```

— defun getModemap —

```
(defun |getModemap| (x env)
```

```
(let (u)
  (dolist (modemap (|get| (first x) '|modemap| env))
    (when (setq u (|compApplyModemap| x modemap env nil))
      (return (sublis (third u) modemap))))))
```

5.4.13 defun getUniqueSignature

[getUniqueModemap p235]

— defun getUniqueSignature —

```
(defun |getUniqueSignature| (form env)
  (cdar (|getUniqueModemap| (first form) (|#| (rest form)) env)))
```

5.4.14 defun getUniqueModemap

[getModemapList p235]

[qslessp p??]

[stackWarning p??]

— defun getUniqueModemap —

```
(defun |getUniqueModemap| (op numOfArgs env)
  (let (mml)
    (cond
      ((eq 1 (|#| (setq mml (|getModemapList| op numOfArgs env))))
       (car mml))
      ((qslessp 1 (|#| mml))
       (|stackWarning|
        (list numOfArgs " argument form of: " op " has more than one modemap"))
       (car mml))
      (t nil))))
```

5.4.15 defun getModemapList

[qcar p??]

[qcdr p??]

```
[getModemapListFromDomain p236]
[nreverse0 p??]
[get p??]
```

— defun getModemapList —

```
(defun |getModemapList| (op numOfArgs env)
  (let (result)
    (cond
      ((and (consp op) (eq (qfirst op) '|elt|) (consp (qrest op))
        (consp (qcddr op)) (eq (qcdddr op) nil))
       (|getModemapListFromDomain| (third op) numOfArgs (second op) env))
      (t
       (dolist (term (|get| op '|modemap| env) (nreverse0 result))
         (when (eql numOfArgs (|#| (cddar term))) (push term result)))))))
```

—————

5.4.16 defun getModemapListFromDomain

```
[get p??]
```

— defun getModemapListFromDomain —

```
(defun |getModemapListFromDomain| (op numOfArgs d env)
  (loop for term in (|get| op '|modemap| env)
        when (and (equal (caar term) d) (eql (|#| (cddar term)) numOfArgs))
        collect term))
```

—————

5.4.17 defun domainMember

```
[modeEqual p348]
```

— defun domainMember —

```
(defun |domainMember| (dom domList)
  (let (result)
    (dolist (d domList result)
      (setq result (or result (|modeEqual| dom d)))))
```

—————

5.4.18 defun augModemapsFromCategory

```
[evalAndSub p241]
[compilerMessage p??]
[putDomainsInScope p230]
[addModemapKnown p245]
[$base p??]
```

— **defun augModemapsFromCategory** —

```
(defun |augModemapsFromCategory| (domainName functorform categoryForm env)
  (let (tmp1 op sig cond fnsel)
    (declare (special |$base|))
    (setq tmp1 (|evalAndSub| domainName domainName functorform categoryForm env))
    (|compilerMessage| (list '|Adding| domainName '| modemaps|))
    (setq env (|putDomainsInScope| domainName (second tmp1)))
    (setq |$base| 4)
    (dolist (u (first tmp1))
      (setq op (caar u))
      (setq sig (cadar u))
      (setq cond (cadr u))
      (setq fnsel (caddr u))
      (setq env (|addModemapKnown| op domainName sig cond fnsel env)))
    env))
```

— — —

5.4.19 defun addEltModemap

This is a hack to change selectors from strings to identifiers; and to add flag identifiers as literals in the environment [qcar p??]

```
[qcdr p??]
[makeLiteral p??]
[addModemap1 p246]
[systemErrorHere p??]
[$insideCapsuleFunctionIfTrue p??]
[$e p??]
```

— **defun addEltModemap** —

```
(defun |addEltModemap| (op mc sig pred fn env)
  (let (tmp1 v sel lt id)
    (declare (special |$e| |$insideCapsuleFunctionIfTrue|))
    (cond
      ((and (eq op '|elt|) (consp sig))
       (setq tmp1 (reverse sig))
```

```

(setq sel (qfirst tmp1))
(setq lt (nreverse (qrest tmp1)))
(cond
  ((stringp sel)
   (setq id (intern sel))
   (if |$insideCapsuleFunctionIfTrue|
       (setq |$e| (|makeLiteral| id |$e|))
       (setq env (|makeLiteral| id env)))
   (|addModemap1| op mc (append lt (list id)) pred fn env))
  (t (|addModemap1| op mc sig pred fn env))))
((and (eq op '|setelt|) (consp sig))
 (setq tmp1 (reverse sig))
 (setq v (qfirst tmp1))
 (setq sel (qsecond tmp1))
 (setq lt (nreverse (qcddr tmp1)))
 (cond
  ((stringp sel) (setq id (intern sel))
   (if |$insideCapsuleFunctionIfTrue|
       (setq |$e| (|makeLiteral| id |$e|))
       (setq env (|makeLiteral| id env)))
   (|addModemap1| op mc (append lt (list id v)) pred fn env))
  (t (|addModemap1| op mc sig pred fn env))))
(t (|systemErrorHere| "addEltModemap")))))

```

5.4.20 defun mkNewModemapList

```

[member p??]
[assoc p??]
[qcar p??]
[qcdr p??]
[mergeModemap p239]
[nequal p??]
[nreverse0 p??]
[insertModemap p239]
[$InteractiveMode p??]
[$forceAdd p??]

```

— defun mkNewModemapList —

```

(defun |mkNewModemapList| (mc sig pred fn curModemapList env filenameOrNil)
  (let (map entry oldMap opred result)
    (declare (special |$InteractiveMode| |$forceAdd|))
    (setq entry
      (cons (setq map (cons mc sig)) (cons (list pred fn) filenameOrNil)))
    (cond

```

```

((|member| entry curModemapList) curModemapList)
((and (setq oldMap (|assoc| map curModemapList))
      (consp oldMap) (consp (qrest oldMap))
      (consp (qsecond oldMap))
      (consp (qcadadr oldMap))
      (eq (qcddadr oldMap) nil)
      (equal (qcadadr oldMap) fn))
 (setq opred (qcaadr oldMap))
 (cond
  (|$forceAdd| (|mergeModemap| entry curModemapList env))
  ((eq opred t) curModemapList)
  (t
   (when (and (nequal pred t) (nequal pred opred))
     (setq pred (list 'or pred opred)))
   (dolist (x curModemapList (nreverse0 result))
     (push
      (if (equal x oldMap)
          (cons map (cons (list pred fn) filenameOrNil))
          x)
      result))))))
(|$InteractiveMode|
 (|insertModemap| entry curModemapList))
(t
 (|mergeModemap| entry curModemapList env))))

```

5.4.21 defun insertModemap

— defun insertModemap —

```

(defun |insertModemap| (new mmList)
  (if (null mmList) (list new) (cons new mmList)))

```

5.4.22 defun mergeModemap

```

[isSuperDomain p231]
[TruthP p241]
[$forceAdd p??]

```

— defun mergeModemap —

```

(defun |mergeModemap| (entry modemapList env)
  (let (mc sig pred mcp sigp predp newmm mm)
    (declare (special |$forceAdd|))
    ; break out the condition, signature, and predicate fields of the new entry
    (setq mc (caar entry))
    (setq sig (cdar entry))
    (setq pred (caadr entry))
    (seq
     ; walk across the successive tails of the modemap list
     (do ((mmtail modemapList (cdr mmtail)))
         ((atom mmtail) nil)
         (setq mcp (caaar mmtail))
         (setq sigp (cdaar mmtail))
         (setq predp (caadar mmtail)))
     (cond
      ((or (equal mc mcp) (|isSuperDomain| mcp mc env))
       ; if this is a duplicate condition
       (exit
        (progn
         (setq newmm nil)
         (setq mm modemapList)
         ; copy the unique modemap terms
         (loop while (not (eq mm mmtail)) do
          (setq newmm (cons (car mm) newmm))
          (setq mm (cdr mm)))
         ; if the conditions and signatures are equal
         (when (and (equal mc mcp) (equal sig sigp))
          ; we only need one of these unless the conditions are hairy
          (cond
           ((and (null |$forceAdd|) (|TruthP| predp))
            ; the new predicate buys us nothing
            (setq entry nil)
            (return modemapList))
           ((|TruthP| pred)
            ; the thing we matched against is useless, by comparison
            (setq mmtail (cdr mmtail))))))
         (setq modemapList (nconc (nreverse newmm) (cons entry mmtail)))
         (setq entry nil)
         (return modemapList))))))
    ; if the entry is still defined, add it to the modemap
    (if entry
     (append modemapList (list entry))
     modemapList))))

```

5.4.23 defun TruthP

[qcar p??]

— defun TruthP —

```
(defun |TruthP| (x)
  (cond
    ((null x) nil)
    ((eq x t) t)
    ((and (consp x) (eq (qfirst x) 'quote)) t)
    (t nil)))
```

5.4.24 defun evalAndSub

```
[isCategory p??]
[substNames p243]
[contained p??]
[put p??]
[get p??]
[getOperationAlist p242]
[$lhsOfColon p??]
```

— defun evalAndSub —

```
(defun |evalAndSub| (domainName viewName functorForm form |$e|)
  (declare (special |$e|))
  (let (|$lhsOfColon| opAlist substAlist)
    (declare (special |$lhsOfColon|))
    (setq |$lhsOfColon| domainName)
    (cond
      ((|isCategory| form)
       (list (|substNames| domainName viewName functorForm (elt form 1)) |$e|))
      (t
       (when (contained '$$ form)
         (setq |$e| (|put| '$$ '|mode| (|get| '$ '|mode| |$e|) |$e|)))
       (setq opAlist (|getOperationAlist| domainName functorForm form))
       (setq substAlist (|substNames| domainName viewName functorForm opAlist))
       (list substAlist |$e|))))))
```

5.4.25 defun getOperationAlist

```
[getdatabase p??]
[isFunctor p229]
[systemError p??]
[compMakeCategoryObject p180]
[stackMessage p??]
[$e p??]
[$domainShell p??]
[$insideFunctorIfTrue p??]
[$functorForm p??]
```

— defun getOperationAlist —

```
(defun |getOperationAlist| (name functorForm form)
  (let (u tt)
    (declare (special |$e| |$domainShell| |$insideFunctorIfTrue| |$functorForm|))
    (when (and (atom name) (getdatabase name 'niladic))
      (setq functorform (list functorForm)))
    (cond
      ((and (setq u (|isFunctor| functorForm))
            (null (and |$insideFunctorIfTrue|
                      (equal (first functorForm) (first |$functorForm|)))))
       u)
      ((and |$insideFunctorIfTrue| (eq name '$))
       (if |$domainShell|
          (elt |$domainShell| 1)
          (|systemError| "$ has no shell now")))
      ((setq tt (|compMakeCategoryObject| form |$e|))
       (setq |$e| (third tt))
       (elt (first tt) 1))
      (t
       (|stackMessage| (list '|not a category form: | form))))))
```

—

5.4.26 defvar \$FormalMapVariableList

— initvars —

```
(defvar |$FormalMapVariableList|
  '(\#1 \#2 \#3 \#4 \#5 \#6 \#7 \#8 \#9 \#10 \#11 \#12 \#13 \#14 \#15))
```

—

5.4.27 defun substNames

```
[substq p??]
[isCategoryPackageName p??]
[eqsubstlist p??]
[nreverse0 p??]
[$FormalMapVariableList p242]
```

— **defun substNames** —

```
(defun |substNames| (domainName viewName functorForm opalist)
  (let (nameForDollar sel pos modemapform tmp0 tmp1)
    (declare (special |$FormalMapVariableList|))
    (setq functorForm (substq '$$ '$ functorForm))
    (setq nameForDollar
      (if (|isCategoryPackageName| functorForm)
          (second functorForm)
          domainName))
    ; following calls to SUBSTQ must copy to save RPLAC's in
    ; putInLocalDomainReferences
    (dolist (term
              (eqsubstlist (kdr functorForm) |$FormalMapVariableList| opalist)
              (nreverse0 tmp0)))
      (setq tmp1 (reverse term))
      (setq sel (caar tmp1))
      (setq pos (caddr tmp1))
      (setq modemapform (nreverse (cdr tmp1)))
      (push
        (append
          (substq '$ '$$ (substq nameForDollar '$ modemapform))
          (list
            (list sel viewName (if (eq domainName '$) pos (cadar modemapform))))))
        tmp0))))
```

5.4.28 defun augModemapsFromCategoryRep

```
[evalAndSub p241]
[isCategory p??]
[compilerMessage p??]
[putDomainsInScope p230]
[assoc p??]
[msubst p??]
[addModemap p245]
[$base p??]
```

— defun augModemapsFromCategoryRep —

```
(defun |augModemapsFromCategoryRep|
  (domainName repDefn functorBody categoryForm env)
  (labels (
    (redefinedList (op z)
      (let (result)
        (dolist (u z result)
          (setq result (or result (redefined op u))))))
    (redefined (opname u)
      (let (op z result)
        (when (consp u)
          (setq op (qfirst u))
          (setq z (qrest u))
          (cond
            ((eq op 'def) (equal opname (caar z)))
            ((member op '(progn seq)) (redefinedList opname z))
            ((eq op 'cond)
              (dolist (v z result)
                (setq result (or result (redefinedList opname (cdr v))))))))))
    (let (fnAlist tmp1 repFnAlist catform lhs op sig cond fnset u)
      (declare (special |$base|))
      (setq tmp1 (|evalAndSub| domainName domainName domainName categoryForm env))
      (setq fnAlist (car tmp1))
      (setq env (cadr tmp1))
      (setq tmp1 (|evalAndSub| '|Rep| '|Rep| repDefn (|getmode| repDefn env) env))
      (setq repFnAlist (car tmp1))
      (setq env (cadr tmp1))
      (setq catform
        (if (|isCategory| categoryForm) (elt categoryForm 0) categoryForm))
      (|compilerMessage| (list '|Adding| domainName '| modemaps|))
      (setq env (|putDomainsInScope| domainName env))
      (setq |$base| 4)
      (dolist (term fnAlist)
        (setq lhs (car term))
        (setq op (caar term))
        (setq sig (cadar term))
        (setq cond (cadr term))
        (setq fnset (caddr term))
        (setq u (|assoc| (msubst '|Rep| domainName lhs) repFnAlist))
        (if (and u (null (redefinedList op functorBody)))
          (setq env (|addModemap| op domainName sig cond (caddr u) env))
          (setq env (|addModemap| op domainName sig cond fnset env))))
      env)))
```

5.5 Maintaining Modemaps

5.5.1 defun addModemapKnown

```
[addModemap0 p246]
```

```
[$e p??]
```

```
[$insideCapsuleFunctionIfTrue p??]
```

```
[$CapsuleModemapFrame p??]
```

— defun addModemapKnown —

```
(defun |addModemapKnown| (op mc sig pred fn |$e|)
  (declare (special |$e| |$CapsuleModemapFrame| |$insideCapsuleFunctionIfTrue|))
  (if (eq |$insideCapsuleFunctionIfTrue| t)
      (progn
        (setq |$CapsuleModemapFrame|
          (|addModemap0| op mc sig pred fn |$CapsuleModemapFrame|))
        |$e|)
      (|addModemap0| op mc sig pred fn |$e|)))
```

— — —

5.5.2 defun addModemap

```
[addModemap0 p246]
```

```
[knownInfo p??]
```

```
[$e p??]
```

```
[$InteractiveMode p??]
```

```
[$insideCapsuleFunctionIfTrue p??]
```

```
[$CapsuleModemapFrame p??]
```

```
[$CapsuleModemapFrame p??]
```

— defun addModemap —

```
(defun |addModemap| (op mc sig pred fn |$e|)
  (declare (special |$e| |$CapsuleModemapFrame| |$InteractiveMode|
    |$insideCapsuleFunctionIfTrue|))
  (cond
    (|$InteractiveMode| |$e|)
    (t
      (when (|knownInfo| pred) (setq pred t))
      (cond
        ((eq |$insideCapsuleFunctionIfTrue| t)
          (setq |$CapsuleModemapFrame|
            (|addModemap0| op mc sig pred fn |$CapsuleModemapFrame|))
          |$e|)
```

```
(t
  (|addModemap0| op mc sig pred fn |$e|))))))
```

5.5.3 defun addModemap0

```
[qcar p??]
[addEltModemap p237]
[addModemap1 p246]
[$functorForm p??]
```

— defun addModemap0 —

```
(defun |addModemap0| (op mc sig pred fn env)
  (declare (special |$functorForm|))
  (cond
    ((and (consp |$functorForm|)
          (eq (qfirst |$functorForm|) '|CategoryDefaults|)
          (eq mc '$))
     env)
    ((or (eq op '|elt|) (eq op '|setelt|))
     (|addEltModemap| op mc sig pred fn env))
    (t (|addModemap1| op mc sig pred fn env))))
```

5.5.4 defun addModemap1

```
[msubst p??]
[getProplist p??]
[mkNewModemapList p238]
[lassoc p??]
[augProplist p??]
[unErrorRef p??]
[addBinding p??]
```

— defun addModemap1 —

```
(defun |addModemap1| (op mc sig pred fn env)
  (let (currentProplist newModemapList newProplist newProplistp)
    (when (eq mc '|Rep|) (setq sig (msubst '$ '|Rep| sig)))
    (setq currentProplist (or (|getProplist| op env) nil))
    (setq newModemapList
```

```
(|mkNewModemapList| mc sig pred fn
  (lassoc '|modemap| currentProplist) env nil))
(setq newProplist (|augProplist| currentProplist '|modemap| newModemapList))
(setq newProplistp (|augProplist| newProplist 'fluid t))
(|unErrorRef| op)
(|addBinding| op newProplistp env)))
```

5.6 Indirect called comp routines

In the **compExpression** function there is the code:

```
(if (and (atom (car x)) (setq fn (get1 (car x) 'special))))
  (funcall fn x m e)
  (|compForm| x m e)))
```

5.6.1 defplist compAdd plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|add| 'special) 'compAdd))
```

5.6.2 defun compAdd

The **compAdd** function expects three arguments:

1. the **form** which is an —add— specifying the domain to extend and a set of functions to be added
2. the **mode** a —Join—, which is a set of categories and domains
3. the **env** which is a list of functions and their modemaps

The bulk of the work is performed by a call to **compOrCroak** which compiles the functions in the add form capsule.

The **compAdd** function returns a triple, the result of a call to **compCapsule**.

1. the **compiled capsule** which is a progn form which returns the domain

2. the **mode** from the input argument
3. the **env** prepended with the signatures of the functions in the body of the add.

```
[comp p497]
[qcdr p??]
[qcar p??]
[compSubDomain1 p332]
[nreverse0 p??]
[NRTgetLocalIndex p??]
[compTuple2Record p249]
[compOrCroak p496]
[compCapsule p250]
[/editfile p??]
[$addForm p??]
[$addFormLhs p??]
[$EmptyMode p131]
[$NRTaddForm p??]
[$packagesUsed p??]
[$functorForm p??]
[$bootStrapMode p??]
```

— defun compAdd —

```
(defun compAdd (form mode env)
  (let (|$addForm| |$addFormLhs| code domainForm predicate tmp3 tmp4)
    (declare (special |$addForm| |$addFormLhs| |$EmptyMode| |$NRTaddForm|
                      |$packagesUsed| |$functorForm| |$bootStrapMode| /editfile))
    (setq |$addForm| (second form))
    (cond
      ((eq |$bootStrapMode| t)
        (cond
          ((and (consp |$addForm|) (eq (qfirst |$addForm|) '@Tuple)))
          (setq code nil))
        (t
          (setq tmp3 (|comp| |$addForm| mode env))
          (setq code (first tmp3))
          (setq mode (second tmp3))
          (setq env (third tmp3)) tmp3))
      (list
        (list 'cond
          (list '|$bootStrapMode| code)
          (list 't
            (list '|systemError|
              (list 'list '|%b| (mkq (car |$functorForm|)) '|%d| "from"
                '|%b| (mkq (|namestring| /editfile)) '|%d|
                  "needs to be compiled"))))
            mode env))
```

```

(t
  (setq |$addFormLhs| |$addForm|)
  (cond
    ((and (consp |$addForm|) (eq (qfirst |$addForm|) '|SubDomain|)
      (consp (qrest |$addForm|)) (consp (qcddr |$addForm|))
      (eq (qcdddr |$addForm|) nil)))
     (setq domainForm (second |$addForm|))
     (setq predicate (third |$addForm|))
     (setq |$packagesUsed| (cons domainForm |$packagesUsed|))
     (setq |$NRTaddForm| domainForm)
     (|NRTgetLocalIndex| domainForm)
     ; need to generate slot for add form since all $ go-get
     ; slots will need to access it
     (setq tmp3 (|compSubDomain1| domainForm predicate mode env))
     (setq |$addForm| (first tmp3))
     (setq env (third tmp3)) tmp3)
  (t
    (setq |$packagesUsed|
      (if (and (consp |$addForm|) (eq (qfirst |$addForm|) '|@Tuple|))
        (append (qrest |$addForm|) |$packagesUsed|)
        (cons |$addForm| |$packagesUsed|)))
      (setq |$NRTaddForm| |$addForm|)
      (setq tmp3
        (cond
          ((and (consp |$addForm|) (eq (qfirst |$addForm|) '|@Tuple|))
            (setq |$NRTaddForm|
              (cons '|@Tuple|
                (dolist (x (cdr |$addForm|) (nreverse0 tmp4))
                  (push (|NRTgetLocalIndex| x) tmp4))))
              (|compOrCroak| (|compTuple2Record| |$addForm|) |$EmptyMode| env))
            (t
              (|compOrCroak| |$addForm| |$EmptyMode| env))))
            (setq |$addForm| (first tmp3))
            (setq env (third tmp3))
            tmp3))
      (|compCapsule| (third form) mode env))))

```

5.6.3 defun compTuple2Record

— defun compTuple2Record —

```

(defun |compTuple2Record| (u)
  (let ((i 0))
    (cons '|Record|
      (loop for x in (rest u)

```



```
collect (list '|| (incf i) x))))))
```

5.6.4 defplist compCapsule plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'capsule 'special) '|compCapsule|))
```

5.6.5 defun compCapsule

```
[bootStrapError p192]
[compCapsuleInner p250]
[addDomain p228]
[editfile p??]
[$insideExpressionIfTrue p??]
[$functorForm p??]
[$bootStrapMode p??]
```

— defun compCapsule —

```
(defun |compCapsule| (form mode env)
  (let (|$insideExpressionIfTrue| itemList)
    (declare (special |$insideExpressionIfTrue| |$functorForm| /editfile
                    |$bootStrapMode|))
    (setq itemList (cdr form))
    (cond
      ((eq |$bootStrapMode| t)
       (list (|bootStrapError| |$functorForm| /editfile) mode env))
      (t
       (setq |$insideExpressionIfTrue| nil)
       (|compCapsuleInner| itemList mode (|addDomain| '$ env))))))
```

5.6.6 defun compCapsuleInner

```
[addInformation p??]
[compCapsuleItems p252]
```

```
[processFunctor p251]
[mkpf p??]
[$getDomainCode p??]
[$signature p??]
[$form p??]
[$addForm p??]
[$insideCategoryPackageIfTrue p??]
[$insideCategoryIfTrue p??]
[$functorLocalParameters p??]
```

— **defun compCapsuleInner** —

```
(defun |compCapsuleInner| (form mode env)
  (let (localParList data code)
    (declare (special |$getDomainCode| |$signature| |$form| |$addForm|
                      |$insideCategoryPackageIfTrue| |$insideCategoryIfTrue|
                      |$functorLocalParameters|))
    (setq env (|addInformation| mode env))
    (setq data (cons 'progn form))
    (setq env (|compCapsuleItems| form nil env))
    (setq localParList |$functorLocalParameters|)
    (when |$addForm| (setq data (list 'add |$addForm| data)))
    (setq code
      (if (and |$insideCategoryIfTrue| (null |$insideCategoryPackageIfTrue|))
          data
          (|processFunctor| |$form| |$signature| data localParList env)))
    (cons (mkpf (append |$getDomainCode| (list code))) 'progn (list mode env))))
```

—————

5.6.7 defun processFunctor

```
[error p??]
[buildFunctor p??]
```

— **defun processFunctor** —

```
(defun |processFunctor| (form signature data localParList e)
  (cond
    ((and (consp form) (eq (qrest form) nil)
      (eq (qfirst form) '|CategoryDefaults|))
     (|error| '|CategoryDefaults is a reserved name|))
    (t (|buildFunctor| form signature data localParList e))))
```

—————

5.6.8 defun compCapsuleItems

The variable `data` appears to be unbound at runtime. Optimized code won't check for this but interpreted code fails. We should PROVE that `data` is unbound at runtime but have not done so yet. Rather than remove the code entirely (since there MIGHT be a path where it is used) we check for the runtime bound case and assign `$myFunctorBody` if `data` has a value.

The `compCapsuleInner` function in this file LOOKS like it sets `data` and expects code to manipulate the assigned `data` structure. Since we can't be sure we take the least disruptive course of action.

```
[compSingleCapsuleItem p252]
[$top-level p??]
[$myFunctorBody p??]
[$signatureOfForm p??]
[$suffix p??]
[$e p??]
[$pred p??]
[$e p??]
```

— defun compCapsuleItems —

```
(defun |compCapsuleItems| (itemlist |$predl| |$e|)
  (declare (special |$predl| |$e|))
  (let ($top_level |$myFunctorBody| |$signatureOfForm| |$suffix|)
    (declare (special $top_level |$myFunctorBody| |$signatureOfForm| |$suffix|))
    (setq $top_level nil)
    (setq |$myFunctorBody| nil)
    (when (boundp '|data|) (setq |$myFunctorBody| |data|))
    (setq |$signatureOfForm| nil)
    (setq |$suffix| 0)
    (loop for item in itemlist do
      (setq |$e| (|compSingleCapsuleItem| item |$predl| |$e|)))
    |$e|))
```

5.6.9 defun compSingleCapsuleItem

```
[doit p??]
[$pred p??]
[$e p??]
[macroExpandInPlace p136]
```

— defun compSingleCapsuleItem —

```
(defun |compSingleCapsuleItem| (item |$predl| |$e|)
```

```
(declare (special |$predl| |$e|))
(|doIt| (|macroExpandInPlace| item |$e|) |$predl|)
|$e|)
```

5.6.10 defun doIt

```
[qcar p??]
[qcdr p??]
[lastnode p??]
[compSingleCapsuleItem p252]
[isDomainForm p329]
[stackWarning p??]
[doIt p253]
[compOrCroak p496]
[stackSemanticError p??]
[bright p??]
[member p??]
[kar p??]
[—isFunction p??]
[insert p??]
[opOf p??]
[get p??]
[NRTgetLocalIndex p??]
[sublis p??]
[NRTgetLocalIndexClear p??]
[compOrCroak p496]
[sayBrightly p??]
[formatUnabbreviated p??]
[doItIf p257]
[isMacro p259]
[put p??]
[cannotDo p??]
[$predl p??]
[$e p??]
[$EmptyMode p131]
[$NonMentionableDomainNames p??]
[$functorLocalParameters p??]
[$functorsUsed p??]
[$packagesUsed p??]
[$NRTopt p??]
[$Representation p??]
[$LocalDomainAlist p??]
[$QuickCode p??]
```

```

[$signatureOfForm p??]
[$genno p??]
[$e p??]
[$functorLocalParameters p??]
[$functorsUsed p??]
[$packagesUsed p??]
[$Representation p??]
[$LocalDomainAlist p??]

```

— **defun doIt** —

```

(defun |doIt| (item |$predl|)
  (declare (special |$predl|))
  (prog ($genno x rhs lhsp lhs rhsp rhsCode z tmp1 tmp2 tmp6 op body tt
        functionPart u code)
    (declare (special $genno |$e| |$EmptyMode| |$signatureOfForm|
                      |$QuickCode| |$LocalDomainAlist| |$Representation|
                      |$NRTopt| |$packagesUsed| |$functorsUsed|
                      |$functorLocalParameters| |$NonMentionableDomainNames|))
    (setq $genno 0)
    (cond
     ((and (consp item) (eq (qfirst item) 'seq) (consp (qrest item))
           (progn (setq tmp6 (reverse (qrest item))) t)
           (consp tmp6) (consp (qfirst tmp6))
           (eq (qcaar tmp6) '|exit|)
           (consp (qcдар tmp6))
           (equal (qcadar tmp6) 1)
           (consp (qcddar tmp6))
           (eq (qcdddar tmp6) nil))
      (setq x (qcaddar tmp6))
      (setq z (qrest tmp6))
      (setq z (nreverse z))
      (rplaca item 'progn)
      (rplaca (lastnode item) x)
      (loop for it1 in (rest item)
            do (setq |$e| (|compSingleCapsuleItem| it1 |$predl| |$e|))))
    ((|isDomainForm| item |$e|)
     (setq u (list '|import| (cons (car item) (cdr item))))
     (|stackWarning| (list '|Use: import | (cons (car item) (cdr item))))
     (rplaca item (car u))
     (rplacd item (cdr u))
     (|doIt| item |$predl|))
    ((and (consp item) (eq (qfirst item) 'let) (consp (qrest item))
          (consp (qcddr item)))
     (setq lhs (qsecond item))
     (setq rhs (qthird item))
     (cond
      ((null (progn
               (setq tmp2 (|compOrCroak| item |$EmptyMode| |$e|))

```

```

      (and (consp tmp2)
        (progn
          (setq code (qfirst tmp2))
          (and (consp (qrest tmp2))
            (progn
              (and (consp (qcddr tmp2))
                (eq (qcdddr tmp2) nil)
                (PROGN
                  (setq |$e| (qthird tmp2))
                  t)))))))
(|stackSemanticError|
 (cons '|cannot compile assigned value to| (|bright| lhs))
 nil))
((null (and (consp code) (eq (qfirst code) 'let)
  (progn
    (and (consp (qrest code))
      (progn
        (setq lhsp (qsecond code))
        (and (consp (qcddr code))))))
    (atom (qsecond code))))
 (cond
  ((and (consp code) (eq (qfirst code) 'progn))
   (|stackSemanticError|
    (list '|multiple assignment | item '| not allowed|)
    nil))
  (t
   (rplaca item (car code))
   (rplacd item (cdr code)))))
(t
 (setq lhs lhsp)
 (cond
  ((and (null (|member| (kar rhs) |$NonMentionableDomainNames|))
    (null (member lhs |$functorLocalParameters|)))
   (setq |$functorLocalParameters|
    (append |$functorLocalParameters| (list lhs)))))
  (cond
   ((and (consp code) (eq (qfirst code) 'let)
    (progn
      (setq tmp2 (qrest code))
      (and (consp tmp2)
        (progn
          (setq tmp6 (qrest tmp2))
          (and (consp tmp6)
            (progn
              (setq rhsp (qfirst tmp6))
              t))))))
    (|isDomainForm| rhsp |$e|))
   (cond
    ((|isFunctor| rhsp)
     (setq |$functorsUsed| (|insert| (|opOf| rhsp) |$functorsUsed|)))

```

```

      (setq |$packagesUsed| (|insert| (list (|opOf| rhsp))
      |$packagesUsed|)))
    (cond
      ((eq lhs '|Rep|)
        (setq |$Representation| (elt (|get| '|Rep| '|value| |$e|) 0))
        (cond
          ((eq |$NRTopt| t)
            (|NRTgetLocalIndex| |$Representation|)
            (t nil))))
        (setq |$LocalDomainAlist|
          (cons (cons lhs
            (sublis |$LocalDomainAlist| (elt (|get| lhs '|value| |$e|) 0))
            |$LocalDomainAlist|)))
        (cond
          ((and (consp code) (eq (qfirst code) 'let))
            (rplaca item (if |$QuickCode| 'qsetrefv 'setelt))
            (setq rhsCode rhsp)
            (rplacd item (list '$ (|NRTgetLocalIndexClear| lhs) rhsCode)))
          (t
            (rplaca item (car code))
            (rplacd item (cdr code))))))
      ((and (consp item) (eq (qfirst item) '|:|) (consp (qrest item))
        (consp (qcddr item)) (eq (qcdddr item) nil))
        (setq tmp1 (|compOrCroak| item |$EmptyMode| |$e|))
        (setq |$e| (caddr tmp1))
        tmp1)
      ((and (consp item) (eq (qfirst item) '|import|)
        (loop for dom in (qrest item)
          do (|sayBrightly| (cons " importing " (|formatUnabbreviated| dom))))
        (setq tmp1 (|compOrCroak| item |$EmptyMode| |$e|))
        (setq |$e| (caddr tmp1))
        (rplaca item 'progn)
        (rplacd item nil))
      ((and (consp item) (eq (qfirst item) 'if))
        (|doItIf| item |$pred1| |$e|))
      ((and (consp item) (eq (qfirst item) '|where|) (consp (qrest item))
        (|compOrCroak| item |$EmptyMode| |$e|))
        (|compOrCroak| item |$EmptyMode| |$e|))
      ((and (consp item) (eq (qfirst item) 'mdef))
        (setq tmp1 (|compOrCroak| item |$EmptyMode| |$e|))
        (setq |$e| (caddr tmp1)) tmp1)
      ((and (consp item) (eq (qfirst item) 'def) (consp (qrest item))
        (consp (qsecond item)))
        (setq op (qcaadr item))
        (cond
          ((setq body (|isMacro| item |$e|))
            (setq |$e| (|put| op '|macro| body |$e|)))
          (t
            (setq tt (|compOrCroak| item |$EmptyMode| |$e|))
            (setq |$e| (caddr tt))
            (rplaca item '|CodeDefine|)

```

```

      (rplacd (cadr item) (list |$signatureOfForm|))
      (setq functionPart (list '|dispatchFunction| (car tt)))
      (rplaca (cddr item) functionPart)
      (rplacd (cddr item) nil))))
    ((setq u (|compOrCroak| item |$EmptyModel| |$e|))
     (setq code (car u))
     (setq |$e| (caddr u))
     (rplaca item (car code))
     (rplacd item (cdr code)))
    (t (|cannotDo|))))

```

5.6.11 defun doItIf

```

[comp p497]
[userError p??]
[compSingleCapsuleItem p252]
[getSuccessEnvironment p300]
[localExtras p??]
[rplaca p??]
[rplacd p??]
[$e p??]
[$functorLocalParameters p??]
[$pred1 p??]
[$e p??]
[$functorLocalParameters p??]
[$getDomainCode p??]
[$Boolean p??]

```

— defun doItIf —

```

(defun |doItIf| (item |$pred1| |$e|)
  (declare (special |$pred1| |$e|))
  (labels (
    (localExtras (oldFLP)
      (let (oldFLPp flp1 gv ans nils n)
        (declare (special |$functorLocalParameters| |$getDomainCode|))
        (unless (eq oldFLP |$functorLocalParameters|)
          (setq flp1 |$functorLocalParameters|)
          (setq oldFLPp oldFLP)
          (setq n 0)
          (loop while oldFLPp
            do
              (setq oldFLPp (cdr oldFLPp))
              (setq n (1+ n))))

```



```

(setq nils (setq ans nil))
(loop for u in flp1
  do
    (if (or (atom u)
      (let (result)
        (loop for v in |$getDomainCode|
          do
            (setq result (or result
              (and (consp v) (consp (qrest v))
                (equal (qsecond v) u))))
          result)))
      ; Now we have to add code to compile all the elements of
      ; functorLocalParameters that were added during the conditional compilation
      (setq nils (cons u nils))
      (progn
        (setq gv (gensym))
        (setq ans (cons (list 'let gv u) ans))
        (setq nils (CONS gv nils))))
      (setq n (1+ n)))
    (setq |$functorLocalParameters| (append oldFLP (nreverse nils))
      (nreverse ans))))
(let (p x y olde tmp1 pp xp oldFLP yp)
  (declare (special |$functorLocalParameters| |$Boolean|))
  (setq p (second item))
  (setq x (third item))
  (setq y (fourth item))
  (setq olde |$e|)
  (setq tmp1
    (or (|comp| p |$Boolean| |$e|)
      (|userError| (list "not a Boolean:" p))))
  (setq pp (first tmp1))
  (setq |$e| (third tmp1))
  (setq oldFLP |$functorLocalParameters|)
  (unless (eq x '|noBranch|)
    (|compSingleCapsuleItem| x |$pred1| (|getSuccessEnvironment| p |$e|))
    (setq xp (localExtras oldFLP)))
  (setq oldFLP |$functorLocalParameters|)
  (unless (eq y '|noBranch|)
    (|compSingleCapsuleItem| y |$pred1| (|getInverseEnvironment| p olde))
    (setq yp (localExtras oldFLP)))
  (rplaca item 'cond)
  (rplacd item (list (cons pp (cons x xp)) (cons ''t (cons y yp))))))

```

5.6.12 defun isMacro

```
[qcar p??]
[qcdr p??]
[get p??]
```

— defun isMacro —

```
(defun |isMacro| (x env)
  (let (op args signature body)
    (when
      (and (consp x) (eq (qfirst x) 'def) (consp (qrest x))
           (consp (qsecond x)) (consp (qcddr x))
           (consp (qcdddr x))
           (consp (qcddddr x))
           (eq (qrest (qcddddr x)) nil))
      (setq op (qcaadr x))
      (setq args (qcdadr x))
      (setq signature (qthird x))
      (setq body (qfirst (qcddddr x)))
      (when
        (and (null (|get| op '|modemap| env))
              (null args)
              (null (|get| op '|mode| env))
              (consp signature)
              (eq (qrest signature) nil)
              (null (qfirst signature)))
          body))))
```

—————

5.6.13 defplist compCase plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|case| 'special) '|compCase|))
```

—————

5.6.14 defun compCase

Will the jerk who commented out these two functions please NOT do so again. These functions ARE needed, and case can NOT be done by modemap alone. The reason is that

A case B requires to take A evaluated, but B unevaluated. Therefore a special function is required. You may have thought that you had tested this on “failed” etc., but “failed” evaluates to it’s own mode. Try it on x case \$ next time.

An angry JHD - August 15th., 1984 [addDomain p228]
 [compCase1 p260]
 [coerce p337]

— defun compCase —

```
(defun |compCase| (form mode env)
  (let (mp td)
    (setq mp (third form))
    (setq env (|addDomain| mp env))
    (when (setq td (|compCase1| (second form) mp env)) (|coerce| td mode))))
```

—————

5.6.15 defun compCase1

[comp p497]
 [getModemapList p235]
 [nreverse0 p??]
 [modeEqual p348]
 [\$Boolean p??]
 [\$EmptyMode p131]

— defun compCase1 —

```
(defun |compCase1| (form mode env)
  (let (xp mp ep map tmp3 tmp5 tmp6 u fn)
    (declare (special |$Boolean| |$EmptyMode|))
    (when (setq tmp3 (|comp| form |$EmptyMode| env))
      (setq xp (first tmp3))
      (setq mp (second tmp3))
      (setq ep (third tmp3))
      (when
        (setq u
          (dolist (modemap (|getModemapList| ' |case| 2 ep) (nreverse0 tmp5))
            (setq map (first modemap))
            (when
              (and (consp map) (consp (qrest map)) (consp (qcddr map))
                (consp (qcdddr map))
                (eq (qcdddr map) nil)
                (|modeEqual| (fourth map) mode)
                (|modeEqual| (third map) mp))
              (push (second modemap) tmp5))))
```

```
(when
  (setq fn
    (dolist (onepair u tmp6)
      (when (first onepair) (setq tmp6 (or tmp6 (second onepair))))))
  (list (list '|call| fn xp) |$Boolean| ep))))
```

5.6.16 defplist compCat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Record| 'special) '|compCat|))
```

5.6.17 defplist compCat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Mapping| 'special) '|compCat|))
```

5.6.18 defplist compCat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Union| 'special) '|compCat|))
```

5.6.19 defun compCat

```
[getl p??]
```

— defun compCat —

```

(defun |compCat| (form mode env)
  (declare (ignore mode))
  (let (functorName fn tmp1 tmp2 funList op sig catForm)
    (setq functorName (first form))
    (when (setq fn (get1 functorName '|makeFunctionList|))
      (setq tmp1 (funcall fn form form env))
      (setq funList (first tmp1))
      (setq env (second tmp1))
      (setq catForm
        (list '|Join| '(|SetCategory|)
          (cons 'category
            (cons '|domain|
              (dolist (item funList (nreverse0 tmp2))
                (setq op (first item))
                (setq sig (second item))
                (unless (eq op '=) (push (list 'signature op sig) tmp2)))))))
      (list form catForm env))))

```

5.6.20 defplist compCategory plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'category 'special) '|compCategory|))

```

5.6.21 defun compCategory

```

[resolve p347]
[qcar p??]
[qcdr p??]
[compCategoryItem p263]
[mkExplicitCategoryFunction p265]
[systemErrorHere p??]
[$sigList p??]
[$atList p??]
[$top-level p??]
[$sigList p??]
[$atList p??]

```

— defun compCategory —

```

(defun |compCategory| (form mode env)
  (let ($top_level |$sigList| |$atList| domainOrPackage z rep)
    (declare (special $top_level |$sigList| |$atList|))
    (setq $top_level t)
    (cond
      ((and
        (equal (setq mode (|resolve| mode (list '|Category|)))
              (list '|Category|))
        (consp form)
        (eq (qfirst form) 'category)
        (consp (qrest form)))
        (setq domainOrPackage (second form))
        (setq z (qcddr form))
        (setq |$sigList| nil)
        (setq |$atList| nil)
        (dolist (x z) (|compCategoryItem| x nil))
        (setq rep
          (|mkExplicitCategoryFunction| domainOrPackage |$sigList| |$atList|))
        (list rep mode env))
      (t
        (|systemErrorHere| "compCategory")))))

```

5.6.22 defun compCategoryItem

```

[qcar p??]
[qcdr p??]
[compCategoryItem p263]
[mkpf p??]
|$sigList| p??]
|$atList| p??]

```

— defun compCategoryItem —

```

(defun |compCategoryItem| (x predl)
  (let (p e a b c predlp pred y z op sig)
    (declare (special |$sigList| |$atList|))
    (cond
      ((null x) nil)
      ; 1. if x is a conditional expression, recurse; otherwise, form the predicate
      ((and (consp x) (eq (qfirst x) 'cond)
        (consp (qrest x)) (eq (qcddr x) nil)
        (consp (qsecond x))
        (consp (qcdadr x))
        (eq (qcddadr x) nil))
        (setq p (qcaadr x))

```

```

(setq e (qcadr x))
(setq predlp (cons p predl))
(cond
  ((and (consp e) (eq (qfirst e) 'progn))
    (setq z (qrest e))
    (dolist (y z) (|compCategoryItem| y predlp)))
  (t (|compCategoryItem| e predlp)))
((and (consp x) (eq (qfirst x) 'if) (consp (qrest x))
  (consp (qcddr x)) (consp (qcdddr x))
  (eq (qcdddr x) nil))
  (setq a (qsecond x))
  (setq b (qthird x))
  (setq c (qfourth x))
  (setq predlp (cons a predl))
  (unless (eq b '|noBranch|)
    (cond
      ((and (consp b) (eq (qfirst b) 'progn))
        (setq z (qrest b))
        (dolist (y z) (|compCategoryItem| y predlp)))
      (t (|compCategoryItem| b predlp))))
    (cond
      ((eq c '|noBranch|) nil)
      (t
        (setq predlp (cons (list '|not| a) predl))
        (cond
          ((and (consp c) (eq (qfirst c) 'progn))
            (setq z (qrest c))
            (dolist (y z) (|compCategoryItem| y predlp)))
          (t (|compCategoryItem| c predlp))))))
    (t
      (setq pred (if predl (mkpf predl 'and) t))
      (cond
        ; 2. if attribute, push it and return
        ((and (consp x) (eq (qfirst x) 'attribute)
          (consp (qrest x)) (eq (qcddr x) nil))
          (setq y (qsecond x))
          (push (mkq (list y pred)) |$atList|))
        ; 3. it may be a list, with PROGN as the CAR, and some information as the CDR
        ((and (consp x) (eq (qfirst x) 'progn))
          (setq z (qrest x))
          (dolist (u z) (|compCategoryItem| u predl)))
        (t
          ; 4. otherwise, x gives a signature for a single operator name or a list of
          ; names; if a list of names, recurse
          (cond ((eq (car x) 'signature) (car x)))
          (setq op (cadr x))
          (setq sig (cddr x))
          (cond
            ((null (atom op))
              (dolist (y op)

```



```

        collect x)))
      result)))
    (|wrapDomainSub| parameters body)))

```

5.6.24 defun mustInstantiate

```

[qcar p??]
[getl p??]
[$DummyFunctorNames p??]

```

— defun mustInstantiate —

```

(defun |mustInstantiate| (d)
  (declare (special |$DummyFunctorNames|))
  (and (consp d)
       (null (or (member (qfirst d) |$DummyFunctorNames|)
                  (getl (qfirst d) '|makeFunctionList|)))))

```

5.6.25 defun wrapDomainSub

— defun wrapDomainSub —

```

(defun |wrapDomainSub| (parameters x)
  (list '|DomainSubstitutionMacro| parameters x))

```

5.6.26 defplist compColon plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|:| 'special) '|compColon|))

```

5.6.27 defun compColon

```

[compColonInside p502]
[assoc p??]
[getDomainsInScope p230]
[isDomainForm p329]
[compColon member (vol5)]
[addDomain p228]
[isDomainForm p329]
[isCategoryForm p??]
[unknownTypeError p229]
[compColon p267]
[eqsubstlist p??]
[take p??]
[length p??]
[nreverse0 p??]
[getmode p??]
[systemErrorHere p??]
[put p??]
[makeCategoryForm p270]
[genSomeVariable p??]
[$lhsOfColon p??]
[$noEnv p??]
[$insideFunctorIfTrue p??]
[$bootStrapMode p??]
[$FormalMapVariableList p242]
[$insideCategoryIfTrue p??]
[$insideExpressionIfTrue p??]

```

— defun compColon —

```

(defun |compColon| (form mode env)
  (let (|$lhsOfColon| argf argt tprime mprime r td op argl newTarget a
        signature tmp2 catform tmp3 g2 g5)
    (declare (special |$lhsOfColon| |$noEnv| |$insideFunctorIfTrue|
                      |$bootStrapMode| |$FormalMapVariableList|
                      |$insideCategoryIfTrue| |$insideExpressionIfTrue|))
    (setq argf (second form))
    (setq argt (third form))
    (if |$insideExpressionIfTrue|
        (|compColonInside| argf mode env argt)
        (progn
          (setq |$lhsOfColon| argf)
          (setq argt
            (cond
              ((and (atom argt)
                    (setq tprime (|assoc| argt (|getDomainsInScope| env))))

```

```

    tprime)
  ((and (|isDomainForm| argt env) (null |$insideCategoryIfTrue|))
   (unless (|member| argt (|getDomainsInScope| env))
    (setq env (|addDomain| argt env))))
  argt)
  ((or (|isDomainForm| argt env) (|isCategoryForm| argt env))
   argt)
  ((and (consp argt) (eq (qfirst argt) '|Mapping|)
   (progn
    (setq tmp2 (qrest argt))
    (and (consp tmp2)
     (progn
      (setq mprime (qfirst tmp2))
      (setq r (qrest tmp2))
      t))))
   argt)
  (t
   (|unknownTypeError| argt)
   argt)))
(cond
 ((eq (car argf) 'listof)
  (dolist (x (cdr argf) td)
   (setq td (|compColon| (list '|:| x argt) mode env))
   (setq env (third td))))
 (t
  (setq env
   (cond
    ((and (consp argf)
     (progn
      (setq op (qfirst argf))
      (setq arg1 (qrest argf))
      t)
     (null (and (consp argt) (eq (qfirst argt) '|Mapping|))))
    (setq newTarget
     (eqsubstlist (take (|#| arg1) |$FormalMapVariableList|)
      (dolist (x arg1 (nreverse0 g2))
       (setq g2
        (cons
         (cond
          ((and (consp x) (eq (qfirst x) '|:|)
           (progn
            (setq tmp2 (qrest x))
            (and (consp tmp2)
             (progn
              (setq a (qfirst tmp2))
              (setq tmp3 (qrest tmp2))
              (and (consp tmp3)
               (eq (qrest tmp3) nil)
               (progn
                (setq mode (qfirst tmp3))

```

```

t))))))
      a)
      (t x))
    g2)))
  argt))
  (setq signature
    (cons '|Mapping|
      (cons newTarget
        (dolist (x arg1 (nreverse0 g5))
          (setq g5
            (cons
              (cond
                ((and (consp x) (eq (qfirst x) '|:|)
                  (progn
                    (setq tmp2 (qrest x))
                    (and (consp tmp2)
                      (progn
                        (setq a (qfirst tmp2))
                        (setq tmp3 (qrest tmp2))
                        (and (consp tmp3)
                          (eq (qrest tmp3) nil)
                          (progn
                            (setq mode (qfirst tmp3))
                            t)))))))
                (mode)
              (t
                (or (|getmode| x env)
                  (|systemErrorHere| "compColonOld")))))
            g5))))))
    (|put| op '|mode| signature env))
  (t (|put| argf '|mode| argt env))))
  (cond
    ((and (null |$bootStrapMode|) |$insideFunctorIfTrue|
      (progn
        (setq tmp2 (|makeCategoryForm| argt env))
        (and (consp tmp2)
          (progn
            (setq catform (qfirst tmp2))
            (setq tmp3 (qrest tmp2))
            (and (consp tmp3)
              (eq (qrest tmp3) nil)
              (progn
                (setq env (qfirst tmp3))
                t)))))))
      (setq env
        (|put| argf '|value| (list (|genSomeVariable|) argt |$noEnv|)
          env))))
    (list '|/throwAway| (|getmode| argf env) env ))))))

```

5.6.28 defun makeCategoryForm

[isCategoryForm p??]
 [compOrCroak p496]
 [\$EmptyMode p131]

— defun makeCategoryForm —

```
(defun |makeCategoryForm| (c env)
  (let (tmp1)
    (declare (special |$EmptyMode|))
    (when (|isCategoryForm| c env)
      (setq tmp1 (|compOrCroak| c |$EmptyMode| env))
      (list (first tmp1) (third tmp1)))))
```

5.6.29 defplist compCons plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'cons 'special) '|compCons|))
```

5.6.30 defun compCons

[compCons1 p271]
 [compForm p507]

— defun compCons —

```
(defun |compCons| (form mode env)
  (or (|compCons1| form mode env) (|compForm| form mode env)))
```

5.6.31 defun compCons1

```
[comp p497]
[convert p504]
[qcar p??]
[qcdr p??]
[$EmptyMode p131]
```

— defun compCons1 —

```
(defun |compCons1| (arg mode env)
  (let (mx y my yt mp mr ytp tmp1 x td)
    (declare (special |$EmptyMode|))
    (setq x (second arg))
    (setq y (third arg))
    (when (setq tmp1 (|comp| x |$EmptyMode| env))
      (setq x (first tmp1))
      (setq mx (second tmp1))
      (setq env (third tmp1))
      (cond
        ((null y)
         (|convert| (list (list 'list x) (list '|List| mx) env ) mode))
        (t
         (when (setq yt (|comp| y |$EmptyMode| env))
           (setq y (first yt))
           (setq my (second yt))
           (setq env (third yt))
           (setq td
            (cond
              ((and (consp my) (eq (qfirst my) '|List|) (consp (qrest my)))
               (setq mp (second my))
               (when (setq mr (list '|List| (|resolve| mp mx)))
                 (when (setq ytp (|convert| yt mr))
                   (when (setq tmp1 (|convert| (list x mx (third ytp)) (second mr)))
                     (setq x (first tmp1))
                     (setq env (third tmp1))
                     (cond
                       ((and (consp (car ytp)) (eq (qfirst (car ytp)) 'list))
                        (list (cons 'list (cons x (cdr (car ytp)))) mr env))
                       (t
                        (list (list 'cons x (car ytp)) mr env)))))))
              (t
               (list (list 'cons x y) (list '|Pair| mx my) env ))))
            (|convert| td mode))))))
```

5.6.32 defplist compConstruct plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|construct| 'special) '|compConstruct|))
```

—————

5.6.33 defun compConstruct

```
[modeIsAggregateOf p??]
[compList p506]
[convert p504]
[compForm p507]
[compVector p335]
[getDomainsInScope p230]
```

— defun compConstruct —

```
(defun |compConstruct| (form mode env)
  (let (z y td tp)
    (setq z (cdr form))
    (cond
      ((setq y (|modeIsAggregateOf| '|List| mode env))
       (if (setq td (|compList| z (list '|List| (cadr y)) env))
           (|convert| td mode)
           (|compForm| form mode env)))
      ((setq y (|modeIsAggregateOf| '|Vector| mode env))
       (if (setq td (|compVector| z (list '|Vector| (cadr y)) env))
           (|convert| td mode)
           (|compForm| form mode env)))
      ((setq td (|compForm| form mode env)) td)
      (t
       (dolist (d (|getDomainsInScope| env))
         (cond
           ((and (setq y (|modeIsAggregateOf| '|List| d env))
                  (setq td (|compList| z (list '|List| (cadr y)) env))
                  (setq tp (|convert| td mode)))
            (return tp))
           ((and (setq y (|modeIsAggregateOf| '|Vector| d env))
                  (setq td (|compVector| z (list '|Vector| (cadr y)) env))
                  (setq tp (|convert| td mode)))
            (return tp))))))))))
```

—————

5.6.34 defplist compConstructorCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|ListCategory| 'special) '|compConstructorCategory|))
```

—————

5.6.35 defplist compConstructorCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|RecordCategory| 'special) '|compConstructorCategory|))
```

—————

5.6.36 defplist compConstructorCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|UnionCategory| 'special) '|compConstructorCategory|))
```

—————

5.6.37 defplist compConstructorCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|VectorCategory| 'special) '|compConstructorCategory|))
```

—————

5.6.38 defun compConstructorCategory

```
[resolve p347]
[$Category p??]
```

— defun compConstructorCategory —

```
(defun |compConstructorCategory| (form mode env)
  (declare (special |$Category|))
  (list form (|resolve| |$Category| mode) env))
```

5.6.39 defplist compDefine plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'def 'special) '|compDefine|))
```

5.6.40 defun compDefine

```
[compDefine1 p275]
[$tripleCache p??]
[$tripleHits p??]
[$macroIfTrue p??]
[$packagesUsed p??]
```

— defun compDefine —

```
(defun |compDefine| (form mode env)
  (let (|$tripleCache| |$tripleHits| |$macroIfTrue| |$packagesUsed|)
    (declare (special |$tripleCache| |$tripleHits| |$macroIfTrue|
                      |$packagesUsed|))
    (setq |$tripleCache| nil)
    (setq |$tripleHits| 0)
    (setq |$macroIfTrue| nil)
    (setq |$packagesUsed| nil)
    (|compDefine1| form mode env)))
```

5.6.41 defun compDefine1

[macroExpand p136]
 [isMacro p259]
 [getSignatureFromMode p278]
 [compDefine1 p275]
 [compInternalFunction p279]
 [compDefineAddSignature p133]
 [compDefWhereClause p200]
 [compDefineCategory p169]
 [isDomainForm p329]
 [getTargetFromRhs p135]
 [giveFormalParametersValues p135]
 [addEmptyCapsuleIfNecessary p134]
 [compDefineFunctor p181]
 [stackAndThrow p??]
 [strconc p??]
 [getAbbreviation p277]
 [length p??]
 [compDefineCapsuleFunction p280]
 [\$insideExpressionIfTrue p??]
 [\$formalArgList p??]
 [\$form p??]
 [\$op p??]
 [\$prefix p??]
 [\$insideFunctorIfTrue p??]
 [\$Category p??]
 [\$insideCategoryIfTrue p??]
 [\$insideCapsuleFunctionIfTrue p??]
 [\$ConstructorNames p??]
 [\$NoValueMode p131]
 [\$EmptyMode p131]
 [\$insideWhereIfTrue p??]
 [\$insideExpressionIfTrue p??]

— defun compDefine1 —

```

(defun |compDefine1| (form mode env)
  (let ((|$insideExpressionIfTrue| lhs specialCases sig signature rhs newPrefix
        (tmp1 t))
    (declare (special |$insideExpressionIfTrue| |$formalArgList| |$form|
                      |$op| |$prefix| |$insideFunctorIfTrue| |$Category|
                      |$insideCategoryIfTrue| |$insideCapsuleFunctionIfTrue|
                      |$ConstructorNames| |$NoValueMode| |$EmptyMode|
                      |$insideWhereIfTrue| |$insideExpressionIfTrue|))
    (setq |$insideExpressionIfTrue| nil)
    (setq form (|macroExpand| form env))

```

```

(setq lhs (second form))
(setq signature (third form))
(setq specialCases (fourth form))
(setq rhs (fifth form))
(cond
  ((and |$insideWhereIfTrue|
        (|isMacro| form env)
        (or (equal mode |$EmptyMode|) (equal mode |$NoValueMode|)))
    (list lhs mode (|put| (car lhs) '|macro| rhs env)))
  ((and (null (car signature)) (consp rhs)
        (null (member (qfirst rhs) |$ConstructorNames|))
        (setq sig (|getSignatureFromMode| lhs env)))
    (|compDefine1|
     (list 'def lhs (cons (car sig) (cdr signature)) specialCases rhs)
     mode env))
  (|$insideCapsuleFunctionIfTrue| (|compInternalFunction| form mode env))
  (t
   (when (equal (car signature) |$Category|) (setq |$insideCategoryIfTrue| t))
   (setq env (|compDefineAddSignature| lhs signature env))
   (cond
     ((null (dolist (x (rest signature) tmp1) (setq tmp1 (and tmp1 (null x)))))
      (|compDefWhereClause| form mode env))
     (equal (car signature) |$Category|)
      (|compDefineCategory| form mode env nil |$formalArgList|))
     ((and (|isDomainForm| rhs env) (null |$insideFunctorIfTrue|))
      (when (null (car signature))
        (setq signature
          (cons (|getTargetFromRhs| lhs rhs
                (|giveFormalParametersValues| (cdr lhs) env))
                (cdr signature))))
      (setq rhs (|addEmptyCapsuleIfNecessary| (car signature) rhs))
      (|compDefineFunctor|
       (list 'def lhs signature specialCases rhs)
       mode env NIL |$formalArgList|))
     (null |$form|)
      (|stackAndThrow| (list "bad == form " form)))
   (t
    (setq newPrefix
      (if |$prefix|
        (intern (strconc (|encodeItem| |$prefix|) "," (|encodeItem| |$op|)))
        (|getAbbreviation| |$op| (|#| (cdr |$form|)))))
    (|compDefineCapsuleFunction|
     form mode env newPrefix |$formalArgList|))))))

```

5.6.42 defun getAbbreviation

```
[constructor? p??]
[assq p??]
[mkAbbrev p277]
[rplac p??]
[$abbreviationTable p??]
[$abbreviationTable p??]
```

— defun getAbbreviation —

```
(defun |getAbbreviation| (name c)
  (let (cname x n upc newAbbreviation)
    (declare (special |$abbreviationTable|))
    (setq cname (|constructor?| name))
    (cond
      ((setq x (assq cname |$abbreviationTable|))
       (cond
         ((setq n (assq name (cdr x)))
          (cond
            ((setq upc (assq c (cdr n)))
             (cdr upc))
            (t
             (setq newAbbreviation (|mkAbbrev| x cname))
             (rplac (cdr n) (cons (cons c newAbbreviation) (cdr n)))
             newAbbreviation)))
         (t
          (setq newAbbreviation (|mkAbbrev| x x))
          (rplac (cdr x)
                 (cons (cons name (list (cons c newAbbreviation))) (cdr x)))
          newAbbreviation)))
      (t
       (setq |$abbreviationTable|
             (cons (list cname (list name (cons c cname))) |$abbreviationTable|)
             cname))))))
```

—————

5.6.43 defun mkAbbrev

```
[addSuffix p278]
[alistSize p278]
```

— defun mkAbbrev —

```
(defun |mkAbbrev| (x z)
  (|addSuffix| (|alistSize| (cdr x)) z))
```

5.6.44 defun addSuffix

— defun addSuffix —

```
(defun |addSuffix| (n u)
  (let (s)
    (if (alpha-char-p (elt (spadlet s (stringimage u)) (maxindex s)))
        (intern (strconc s (stringimage n)))
        (internl (strconc s (stringimage '|;|) (stringimage n))))))
```

5.6.45 defun alistSize

— defun alistSize —

```
(defun |alistSize| (c)
  (labels (
    (count (x level)
      (cond
        ((eq level 2) (|#| x))
        ((null x) 0)
        (+ (count (cdar x) (1+ level))
           (count (cdr x) level))))))
    (count c 1)))
```

5.6.46 defun getSignatureFromMode

```
[getmode p??]
[opOf p??]
[qcar p??]
[qcdr p??]
[nequal p??]
[length p??]
[stackAndThrow p??]
```

```
[eqsubstlist p??]
[take p??]
[$FormalMapVariableList p242]
```

— **defun getSignatureFromMode** —

```
(defun |getSignatureFromMode| (form env)
  (let (tmp1 signature)
    (declare (special |$FormalMapVariableList|))
    (setq tmp1 (|getmode| (|op0f| form) env))
    (when (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|))
      (setq signature (qrest tmp1))
      (if (nequal (|#| form) (|#| signature))
          (|stackAndThrow| (list '|Wrong number of arguments: | form))
          (eqsubstlist (cdr form)
                        (take (|#| (cdr form)) |$FormalMapVariableList|
                             signature))))))
```

5.6.47 **defun compInternalFunction**

```
[identp p??]
[stackAndThrow p??]
```

— **defun compInternalFunction** —

```
(defun |compInternalFunction| (df m env)
  (let (form signature specialCases body op argl nbody nf ress)
    (setq form (second df))
    (setq signature (third df))
    (setq specialCases (fourth df))
    (setq body (fifth df))
    (setq op (first form))
    (setq argl (rest form))
    (cond
     ((null (identp op))
      (|stackAndThrow| (list '|Bad name for internal function:| op)))
     ((eql (|#| argl) 0)
      (|stackAndThrow|
       (list '|Argumentless internal functions unsupported:| op )))
     (t
      (setq nbody (list '+-> argl body))
      (setq nf (list 'let (list '|:| op (cons '|Mapping| signature)) nbody))
      (setq ress (|comp| nf m env) ress))))
```

5.6.48 defun compDefineCapsuleFunction

```

[length p??]
[get p??]
[profileRecord p??]
[compArgumentConditions p286]
[addDomain p228]
[giveFormalParametersValues p135]
[getSignature p288]
[put p??]
[stripOffSubdomainConditions p287]
[getArgumentModeOrMoan p154]
[checkAndDeclare p289]
[hasSigInTargetCategory p290]
[stripOffArgumentConditions p287]
[resolve p347]
[member p??]
[getmode p??]
[formatUnabbreviated p??]
[sayBrightly p??]
[compOrCroak p496]
[NRTassignCapsuleFunctionSlot p??]
[mkq p??]
[replaceExitEtc p319]
[addArgumentConditions p285]
[compileCases p283]
[addStats p??]
[$semanticErrorStack p??]
[$DomainsInScope p??]
[$op p??]
[$formalArgList p??]
[$signatureOfForm p??]
[$functionLocations p??]
[$profileCompiler p??]
[$compileOnlyCertainItems p??]
[$returnMode p??]
[$functorStats p??]
[$functionStats p??]
[$form p??]
[$functionStats p??]
[$argumentConditionList p??]
[$finalEnv p??]
[$initCapsuleErrorCount p??]
[$insideCapsuleFunctionIfTrue p??]
[$CapsuleModemapFrame p??]
[$CapsuleDomainsInScope p??]

```

```

[$insideExpressionIfTrue p??]
[$returnMode p??]
[$op p??]
[$formalArgList p??]
[$signatureOfForm p??]
[$functionLocations p??]

```

— **defun compDefineCapsuleFunction** —

```

(defun |compDefineCapsuleFunction| (df m oldE |$prefix| |$formalArgList|)
  (declare (special |$prefix| |$formalArgList|))
  (let (|$form| |$op| |$functionStats| |$argumentConditionList| |$finalEnv|
        |$initCapsuleErrorCount| |$insideCapsuleFunctionIfTrue|
        |$CapsuleModemapFrame| |$CapsuleDomainsInScope|
        |$insideExpressionIfTrue| form signature body tmp1 lineNumber
        specialCases argl identSig argModeList signaturep e rettype tmp2
        localOrExported formattedSig tt catchTag bodyp finalBody fun val)
    (declare (special |$form| |$op| |$functionStats| |$functorStats|
                      |$argumentConditionList| |$finalEnv| |$returnMode|
                      |$initCapsuleErrorCount| |$newCompCompare| |$NoValueMode|
                      |$insideCapsuleFunctionIfTrue|
                      |$CapsuleModemapFrame| |$CapsuleDomainsInScope| | |
                      |$insideExpressionIfTrue| |$compileOnlyCertainItems|
                      |$profileCompiler| |$functionLocations| |$finalEnv|
                      |$signatureOfForm| |$semanticErrorStack|))
      (setq form (second df))
      (setq signature (third df))
      (setq specialCases (fourth df))
      (setq body (fifth df))
      (setq tmp1 specialCases)
      (setq lineNumber (first tmp1))
      (setq specialCases (rest tmp1))
      (setq e oldE)
      ; -1. bind global variables
      (setq |$form| nil)
      (setq |$op| nil)
      (setq |$functionStats| (list 0 0))
      (setq |$argumentConditionList| nil)
      (setq |$finalEnv| nil)
      ; used by ReplaceExitEtc to get a common environment
      (setq |$initCapsuleErrorCount| (|#| |$semanticErrorStack|))
      (setq |$insideCapsuleFunctionIfTrue| t)
      (setq |$CapsuleModemapFrame| e)
      (setq |$CapsuleDomainsInScope| (|get| '|$DomainsInScope| 'special e))
      (setq |$insideExpressionIfTrue| t)
      (setq |$returnMode| m)
      (setq |$op| (first form))
      (setq argl (rest form))
      (setq |$form| (cons |$op| argl))

```



```

(setq argl (|stripOffArgumentConditions| argl))
(setq |$formalArgList| (append argl |$formalArgList|))
; let target and local signatures help determine modes of arguments
(setq argModeList
  (cond
    ((setq identSig (|hasSigInTargetCategory| argl form (car signature) e))
      (setq e (|checkAndDeclare| argl form identSig e))
      (cdr identSig))
    (t
      (loop for a in argl
        collect (|getArgumentModeOrMoan| a form e))))))
(setq argModeList (|stripOffSubdomainConditions| argModeList argl))
(setq signaturep (cons (car signature) argModeList))
(unless identSig
  (setq oldE (|put| |$op| '|mode| (cons '|Mapping| signaturep) oldE)))
; obtain target type if not given
(cond
  ((null (car signaturep))
    (setq signaturep
      (cond
        (identSig identSig)
        (t (|getSignature| |$op| (cdr signaturep) e))))))
  (when signaturep
    (setq e (|giveFormalParametersValues| argl e))
    (setq |$signatureOfForm| signaturep)
    (setq |$functionLocations|
      (cons (cons (list |$op| |$signatureOfForm|) lineNumber)
        |$functionLocations|))
    (setq e (|addDomain| (car signaturep) e))
    (setq e (|compArgumentConditions| e))
    (when |$profileCompiler|
      (loop for x in argl for y in signaturep
        do (|profileRecord| '|arguments| x y)))
    ; 4. introduce needed domains into extendedEnv
    (loop for domain in signaturep
      do (setq e (|addDomain| domain e)))
    ; 6. compile body in environment with extended environment
    (setq rettype (|resolve| (car signaturep) |$returnMode|))
    (setq localOrExported
      (cond
        ((and (null (|member| |$op| |$formalArgList|))
          (progn
            (setq tmp2 (|getmode| |$op| e))
            (and (consp tmp2) (eq (qfirst tmp2) '|Mapping|))))
          '|local|)
        (t '|exported|)))
    ; 6a skip if compiling only certain items but not this one
    ; could be moved closer to the top
    (setq formattedSig (|formatUnabbreviated| (cons '|Mapping| signaturep)))
    (cond

```

```

((and |$compileOnlyCertainItems|
  (null (|member| |$op| |$compileOnlyCertainItems|)))
  (|sayBrightly|
    (cons "    skipping " (cons localOrExported (|bright| |$op|))))
  (list nil (cons '|Mapping| signaturep) oldE))
(t
  (|sayBrightly|
    (cons "    compiling " (cons localOrExported (append (|bright| |$op|)
      (cons ": " formattedSig)))))
    (setq tt (catch '|compCapsuleBody| (|compOrCroak| body rettype e)))
    (|NRTassignCapsuleFunctionSlot| |$op| signaturep)
; A THROW to the above CATCH occurs if too many semantic errors occur
; see stackSemanticError
    (setq catchTag (mkq (gensym)))
    (setq fun
      (progn
        (setq bodyp
          (|replaceExitEtc| (car tt) catchTag '|TAGGEDreturn| |$returnMode|))
        (setq bodyp (|addArgumentConditions| bodyp |$op|))
        (setq finalBody (list 'catch catchTag bodyp))
        (|compileCases|
          (list |$op| (list 'lam (append argl (list '$)) finalBody)
            oldE)))
      (setq |$functorStats| (|addStats| |$functorStats| |$functionStats|))
; 7. give operator a 'value property
      (setq val (list fun signaturep e))
      (list fun (list '|Mapping| signaturep) oldE))))))

```

5.6.49 defun compileCases

```

[eval p??]
[qcar p??]
[qcdr p??]
[msubst p??]
[compile p145]
[getSpecialCaseAssoc p285]
[get p??]
[assocleft p??]
[outerProduct p??]
[assocright p??]
[mkpf p??]
[$getDomainCode p??]
[$insideFunctorIfTrue p??]
[$specialCaseKeyList p??]

```

— defun compileCases —

```

(defun |compileCases| (x |$e|)
  (declare (special |$e|))
  (labels (
    (isEltArgumentIn (Rlist x)
      (cond
        ((atom x) nil)
        ((and (consp x) (eq (qfirst x) 'elt) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
          (or (member (second x) Rlist)
            (isEltArgumentIn Rlist (cdr x))))))
        ((and (consp x) (eq (qfirst x) 'qrefelt) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
          (or (member (second x) Rlist)
            (isEltArgumentIn Rlist (cdr x))))))
    (t
      (or (isEltArgumentIn Rlist (car x))
        (isEltArgumentIn Rlist (cdr x))))))
  (FindNamesFor (r rp)
    (let (v u)
      (declare (special |$getDomainCode|))
      (cons r
        (loop for item in |$getDomainCode|
          do
            (setq v (second item))
            (setq u (third item))
            when (and (equal (second u) r) (|eval| (msubst rp r u)))
              collect v))))))
  (let (|$specialCaseKeyList| specialCaseAssoc listOfDomains listOfAllCases cl)
    (declare (special |$specialCaseKeyList| |$true| |$insideFunctorIfTrue|))
    (setq |$specialCaseKeyList| nil)
    (cond
      ((null (eq |$insideFunctorIfTrue| t)) (|compile| x))
      (t
        (setq specialCaseAssoc
          (loop for y in (|getSpecialCaseAssoc|)
            when (and (null (|get| (first y) '|specialCase| |$e|))
              (isEltArgumentIn (FindNamesFor (first y) (second y)) x))
              collect y))
          (cond
            ((null specialCaseAssoc) (|compile| x))
            (t
              (setq listOfDomains (assocleft specialCaseAssoc))
              (setq listOfAllCases (|outerProduct| (assocright specialCaseAssoc)))
              (setq cl
                (loop for z in listOfAllCases
                  collect
                    (progn

```

```

(setq |$specialCaseKeyList|
  (loop for d in listOfDomains for c in z
    collect (cons d c)))
(cons
  (mkpf
    (loop for d in listOfDomains for c in z
      collect (list 'equal d c))
    'and)
  (list (|compile| (copy x))))))
(setq |$specialCaseKeyList| nil)
(cons 'cond (append cl (list (list |$true| (|compile| x)))))))))

```

5.6.50 defun getSpecialCaseAssoc

```

[$functorForm p??]
[$functorSpecialCases p??]

```

— defun getSpecialCaseAssoc —

```

(defun |getSpecialCaseAssoc| ()
  (declare (special |$functorSpecialCases| |$functorForm|))
  (loop for r in (rest |$functorForm|)
    for z in (rest |$functorSpecialCases|)
    when z
    collect (cons r z)))

```

5.6.51 defun addArgumentConditions

```

[qcar p??]
[qcdr p??]
[mkq p??]
[systemErrorHere p??]
[$true p??]
[$functionName p??]
[$body p??]
[$argumentConditionList p??]
[$argumentConditionList p??]

```

— defun addArgumentConditions —

```

(defun |addArgumentConditions| (|$body| |$functionName|)

```

```

(declare (special |$body| |$functionName| |$argumentConditionList| |$true|))
(labels (
  (fn (clist)
    (let (n untypedCondition typedCondition)
      (cond
        ((and (consp clist) (consp (qfirst clist)) (consp (qcdar clist))
          (consp (qcddar clist))
          (eq (qcdddar clist) nil))
         (setq n (qcaar clist))
         (setq untypedCondition (qcadar clist))
         (setq typedCondition (qcaddar clist))
         (list 'cond
          (list typedCondition (fn (cdr clist)))
          (list |$true|
            (list '|argumentDataError| n
              (mkq untypedCondition) (mkq |$functionName|))))))
        ((null clist) |$body|)
        (t (|systemErrorHere| "addArgumentConditions"))))))
(if |$argumentConditionList|
  (fn |$argumentConditionList|
    |$body|)))

```

5.6.52 defun compArgumentConditions

```

[msubst p??]
[compOrCroak p496]
[$Boolean p??]
[$argumentConditionList p??]
[$argumentConditionList p??]

```

— defun compArgumentConditions —

```

(defun |compArgumentConditions| (env)
  (let (n a x y tmp1)
    (declare (special |$Boolean| |$argumentConditionList|))
    (setq |$argumentConditionList|
      (loop for item in |$argumentConditionList|
        do
          (setq n (first item))
          (setq a (second item))
          (setq x (third item))
          (setq y (msubst a '|#1| x))
          (setq tmp1 (|compOrCroak| y |$Boolean| env))
          (setq env (third tmp1)))

```

```

collect
  (list n x (first tmp1)))
env))

```

5.6.53 defun stripOffSubdomainConditions

```

[qcar p??]
[qcdr p??]
[assoc p??]
[mkpf p??]
[$argumentConditionList p??]
[$argumentConditionList p??]

```

— defun stripOffSubdomainConditions —

```

(defun |stripOffSubdomainConditions| (margl argl)
  (let (pair (i 0))
    (declare (special |$argumentConditionList|))
    (loop for x in margl for arg in argl
      do (incf i)
      collect
        (cond
          ((and (consp x) (eq (qfirst x) '|SubDomain|) (consp (qrest x))
            (consp (qcddr x)) (eq (qcdddr x) nil))
            (cond
              ((setq pair (|assoc| i |$argumentConditionList|))
               (rplac (cadr pair) (mkpf (list (third x) (cadr pair)) 'and))
               (second x))
              (t
               (setq |$argumentConditionList|
                 (cons (list i arg (third x)) |$argumentConditionList|))
                 (second x))))
            (t x))))))

```

5.6.54 defun stripOffArgumentConditions

```

[qcar p??]
[qcdr p??]
[msubst p??]
[$argumentConditionList p??]
[$argumentConditionList p??]

```

— defun stripOffArgumentConditions —

```
(defun |stripOffArgumentConditions| (argl)
  (let (condition (i 0))
    (declare (special |$argumentConditionList|))
    (loop for x in argl
      do (incf i)
      collect
      (cond
        ((and (consp x) (eq (qfirst x) '|\\|') (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
         (setq condition (msubst '|#1| (second x) (third x)))
         (setq |$argumentConditionList|
           (cons (list i (second x) condition) |$argumentConditionList|))
         (second x))
        (t x))))))
```

5.6.55 defun getSignature

Try to return a signature. If there isn't one, complain and return nil. If there are more than one then remove any that are subsumed. If there is still more than one complain else return the only signature. [get p??]

```
[length p??]
[remdup p??]
[knownInfo p??]
[getmode p??]
[qcar p??]
[qcdr p??]
[say p??]
[printSignature p??]
[SourceLevelSubsume p??]
[stackSemanticError p??]
[$e p??]
```

— defun getSignature —

```
(defun |getSignature| (op argModeList |$e|)
  (declare (special |$e|))
  (let (mmList pred u tmp1 dc sig sigl)
    (setq mmList (|get| op '|modemap| |$e|))
    (cond
      ((eq 1
        (|#| (setq sigl (remdup
```

```

(loop for item in mmList
  do
    (setq dc (caar item))
    (setq sig (cdar item))
    (setq pred (caadr item))
    when (and (eq dc '$) (equal (cdr sig) argModeList) (|knownInfo| pred))
      collect sig))))
(car sigl))
((null sigl)
 (cond
  ((progn
    (setq tmp1 (setq u (|getmode| op |$e|)))
    (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)))
   (qrest tmp1))
  (t
   (say "***** USER ERROR *****")
   (say "available signatures for " op ": ")
   (cond
    ((null mmList) (say "    NONE"))
    (t
     (loop for item in mmList
       do (|printSignature| '|      | op (cdar item)))
     (|printSignature| '|NEED | op (cons '? argModeList)))
    nil)))
  (t
   ; Before we complain about duplicate signatures, we should
   ; check that we do not have for example, a partial - as
   ; well as a total one. SourceLevelSubsume should do this
   (loop for u in sigl do
     (loop for v in sigl
       when (null (equal u v))
         do (when (|SourceLevelSubsume| u v) (setq sigl (|delete| v sigl)))))
   (cond
    ((eq 1 (|#| sigl)) (car sigl))
    (t
     (|stackSemanticError|
      (list '|duplicate signatures for | op '|: | argModeList) nil)))))))))

```

5.6.56 defun checkAndDeclare

```

[getArgumentMode p291]
[modeEqual p348]
[put p??]
[sayBrightly p??]
[bright p??]

```


— defun checkAndDeclare —

```
(defun |checkAndDeclare| (argl form sig env)
  (let (m1 stack)
    (loop for a in argl for m in (rest sig)
      do
        (if (setq m1 (|getArgumentMode| a env))
          (if (null (|modeEqual| m1 m))
            (setq stack
              (cons '| | (append (|bright| a)
                (cons "must have type "
                  (cons m
                    (cons " not "
                      (cons m1
                        (cons '|%l| stack))))))))
            (setq env (|put| a '|mode| m env)))
          (when stack
            (|sayBrightly|
              (cons " Parameters of "
                (append (|bright| (car form))
                  (cons " are of wrong type:"
                    (cons '|%l| stack))))))
            env)))
```

—————

5.6.57 defun hasSigInTargetCategory

```
[getArgumentMode p291]
[remdup p??]
[length p??]
[getSignatureFromMode p278]
[stackWarning p??]
[compareMode2Arg p??]
[bright p??]
[$domainShell p??]
```

— defun hasSigInTargetCategory —

```
(defun |hasSigInTargetCategory| (argl form opsig env)
  (labels (
    (fn (opName sig opsig mList form)
      (declare (special |$op|))
      (and
        (and
          (and (equal opName |$op|) (equal (|#| sig) (|#| form)))
```

```

      (or (null opsig) (equal opsig (car sig))))
    (let ((result t))
      (loop for x in mList for y in (rest sig)
        do (setq result (and result (or (null x) (|modeEqual| x y)))))
      result))))
(let (mList potentialSigList c sig)
(declare (special |$domainShell|))
(setq mList
  (loop for x in arg1
    collect (|getArgumentMode| x env)))
(setq potentialSigList
  (remdup
    (loop for item in (elt |$domainShell| 1)
      when (fn (caar item) (cadar item) opsig mList form)
        collect (cadar item))))
(setq c (|#| potentialSigList))
(cond
  ((eql 1 c) (car potentialSigList))
  ((eql 0 c)
    (when (equal (|#| (setq sig (|getSignatureFromMode| form env))) (|#| form))
      sig))
  (> c 1)
    (setq sig (car potentialSigList))
    (|stackWarning|
      (cons '|signature of lhs not unique:|
        (append (|bright| sig) (list '|chosen|))))
    sig)
  (t nil))))

```

5.6.58 defun getArgumentMode

[get p??]

— defun getArgumentMode —

```

(defun |getArgumentMode| (x e)
  (if (stringp x) x (|get| x '|mode| e)))

```

5.6.59 defplist compElt plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|elt| 'special) '|compElt|))
```

5.6.60 defun compElt

```
[compForm p507]
[isDomainForm p329]
[addDomain p228]
[getModemapListFromDomain p236]
[length p??]
[stackMessage p??]
[stackWarning p??]
[convert p504]
[opOf p??]
[getDeltaEntry p??]
[nequal p??]
[$One p??]
[$Zero p??]
```

— defun compElt —

```
(defun |compElt| (form mode env)
  (let (aDomain anOp mmList n modemap sig pred val)
    (declare (special |$One| |$Zero|))
    (setq anOp (third form))
    (setq aDomain (second form))
    (cond
      ((null (and (consp form) (eq (qfirst form) '|elt|)
                  (consp (qrest form)) (consp (qcddr form))
                  (eq (qcdddr form) nil))))
        (|compForm| form mode env))
      ((eq aDomain '|Lisp|)
        (list (cond
              ((equal anOp |$Zero|) 0)
              ((equal anOp |$One|) 1)
              (t anOp))
              mode env))
      ((|isDomainForm| aDomain env)
        (setq env (|addDomain| aDomain env))
        (setq mmList (|getModemapListFromDomain| anOp 0 aDomain env))
        (setq modemap
          (progn
            (setq n (|#| mmList))
            (cond
```

```

((eq 1 n) (elt mmList 0))
((eq 0 n)
 (|stackMessage|
  (list "Operation " '|%b| anOp '|%d| "missing from domain: "
        aDomain nil))
 nil)
(t
 (|stackWarning|
  (list "more than 1 modemap for: " anOp " with dc="
        aDomain " ==>" mmList ))
 (elt mmList 0))))
(when modemap
 (setq sig (first modemap))
 (setq pred (caadr modemap))
 (setq val (cadadr modemap))
 (unless (and (nequal (|#| sig) 2)
              (null (and (consp val) (eq (qfirst val) '|elt|))))
  (setq val (|genDeltaEntry| (cons (|opOf| anOp) modemap)))
  (|convert| (list (list '|call| val) (second sig) env) mode))))
(t
 (|compForm| form mode env))))

```

5.6.61 defplist compExit plist

— postvars —

```

(eval-when (eval load)
 (setf (get '|exit| 'special) '|compExit|))

```

5.6.62 defun compExit

```

[comp p497]
[modifyModeStack p525]
[stackMessageIfNone p??]
[$exitModeStack p??]

```

— defun compExit —

```

(defun |compExit| (form mode env)
 (let (exitForm index m1 u)

```

```

(declare (special |$exitModeStack|))
(setq index (1- (second form)))
(setq exitForm (third form))
(cond
  ((null |$exitModeStack|)
   (|comp| exitForm mode env))
  (t
   (setq m1 (elt |$exitModeStack| index))
   (setq u (|comp| exitForm m1 env))
   (cond
     (u
      (|modifyModeStack| (second u) index)
      (list (list ' |TAGGEDexit| index u) mode env))
     (t
      (|stackMessageIfNone|
       (list ' |cannot compile exit expression| exitForm ' |in mode| m1)))))))

```

5.6.63 defplist compHas plist

— postvars —

```

(eval-when (eval load)
  (setf (get ' |has| 'special) ' |compHas|))

```

5.6.64 defun compHas

```

[chaseInferences p??]
[compHasFormat p295]
[coerce p337]
[$e p??]
[$e p??]
[$Boolean p??]

```

— defun compHas —

```

(defun |compHas| (pred mode |$e|)
  (declare (special |$e| |$Boolean|))
  (let (a b predCode)
    (setq a (second pred))
    (setq b (third pred))

```

```
(setq |$e| (|chaseInferences| pred |$e|))
(setq predCode (|compHasFormat| pred))
(|coerce| (list predCode |$Boolean| |$e|) mode)))
```

5.6.65 defun compHasFormat

```
[take p??]
[length p??]
[sublislis p??]
[comp p497]
[qcar p??]
[qcdr p??]
[mkList p296]
[mkDomainConstructor p??]
[isDomainForm p329]
[$FormalMapVariableList p242]
[$EmptyMode p131]
[$e p??]
[$form p??]
[$EmptyEnvironment p??]
```

— defun compHasFormat —

```
(defun |compHasFormat| (pred)
  (let (olda b argl formals tmp1 a)
    (declare (special |$EmptyEnvironment| |$e| |$EmptyMode|
                      |$FormalMapVariableList| |$form|))
    (when (eq (car pred) '|has|) (car pred))
    (setq olda (second pred))
    (setq b (third pred))
    (setq argl (rest |$form|))
    (setq formals (take (|#| argl) |$FormalMapVariableList|))
    (setq a (sublislis argl formals olda))
    (setq tmp1 (|comp| a |$EmptyMode| |$e|))
    (when tmp1
      (setq a (car tmp1))
      (setq a (sublislis formals argl a))
      (cond
        ((and (consp b) (eq (qfirst b) 'attribute) (consp (qrest b))
              (eq (qcddr b) nil))
         (list '|HasAttribute| a (list 'quote (qsecond b))))
        ((and (consp b) (eq (qfirst b) 'signature) (consp (qrest b))
              (consp (qcddr b)) (eq (qcdddr b) NIL))
         (list '|HasSignature| a
```

```

(|mkList|
  (list (MKQ (qsecond b))
        (|mkList|
          (loop for type in (qthird b)
                collect (|mkDomainConstructor| type))))))
((|isDomainForm| b |$EmptyEnvironment|)
  (list 'equal a b))
(t
  (list '|HasCategory| a (|mkDomainConstructor| b))))))

```

5.6.66 defun mkList

— defun mkList —

```

(defun |mkList| (u)
  (when u (cons 'list u)))

```

5.6.67 defplist compIf plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'if 'special) '|compIf|))

```

5.6.68 defun compIf

```

[canReturn p298]
[intersectionEnvironment p??]
[compBoolean p300]
[compFromIf p297]
[resolve p347]
[coerce p337]
[quotify p??]
[$Boolean p??]

```

— defun compIf —

```
(defun |compIf| (form mode env)
  (labels (
    (environ (bEnv cEnv b c env)
      (cond
        ((|canReturn| b 0 0 t)
          (if (|canReturn| c 0 0 t) (|intersectionEnvironment| bEnv cEnv) bEnv))
        ((|canReturn| c 0 0 t) cEnv)
        (t env))))
    (let (a b c tmp1 xa ma Ea Einv Tb xb mb Eb Tc xc mc Ec xbp x returnEnv)
      (declare (special |$Boolean|))
      (setq a (second form))
      (setq b (third form))
      (setq c (fourth form))
      (when (setq tmp1 (|compBoolean| a |$Boolean| env))
        (setq xa (first tmp1))
        (setq ma (second tmp1))
        (setq Ea (third tmp1))
        (setq Einv (fourth tmp1))
        (when (setq Tb (|compFromIf| b mode Ea))
          (setq xb (first Tb))
          (setq mb (second Tb))
          (setq Eb (third Tb))
          (when (setq Tc (|compFromIf| c (|resolve| mb mode) Einv))
            (setq xc (first Tc))
            (setq mc (second Tc))
            (setq Ec (third Tc))
            (when (setq xbp (|coerce| Tb mc))
              (setq x (list 'if xa (first xbp) xc))
              (setq returnEnv (environ (third xbp) Ec (first xbp) xc env))
              (list x mc returnEnv))))))))))
```

—————

5.6.69 defun compFromIf

[comp p497]

— defun compFromIf —

```
(defun |compFromIf| (a m env)
  (if (eq a '|noBranch|)
    (list '|noBranch| m env)
    (|comp| a m env)))
```


5.6.70 defun canReturn

```
[say p??]
[qcar p??]
[qcdr p??]
[canReturn p298]
[systemErrorHere p??]
```

— defun canReturn —

```
(defun |canReturn| (expr level exitCount ValueFlag)
  (labels (
    (findThrow (gs expr level exitCount ValueFlag)
      (cond
        ((atom expr) nil)
        ((and (consp expr) (eq (qfirst expr) 'throw) (consp (qrest expr))
          (equal (qsecond expr) gs) (consp (qcddr expr))
          (eq (qcdddr expr) nil))
        t)
        ((and (consp expr) (eq (qfirst expr) 'seq))
          (let (result)
            (loop for u in (qrest expr)
              do (setq result
                (or result
                  (findThrow gs u (1+ level) exitCount ValueFlag))))
            result)))
      (t
        (let (result)
          (loop for u in (rest expr)
            do (setq result
              (or result
                (findThrow gs u level exitCount ValueFlag))))
          result))))))
  (let (op gs)
    (cond
      ((atom expr) (and ValueFlag (equal level exitCount)))
      ((eq (setq op (car expr)) 'quote) (and ValueFlag (equal level exitCount)))
      ((eq op '|TAGGEDexit|)
        (cond
          ((and (consp expr) (consp (qrest expr)) (consp (qcddr expr))
            (eq (qcdddr expr) nil))
            (|canReturn| (car (third expr)) level (second expr)
              (equal (second expr) level))))
          ((and (equal level exitCount) (null ValueFlag))
            nil)
          ((eq op 'seq)
```

```

(let (result)
  (loop for u in (rest expr)
    do (setq result (or result (|canReturn| u (1+ level) exitCount nil))))
  result))
((eq op '|TAGGEDreturn|) nil)
((eq op 'catch)
  (cond
    ((findThrow (second expr) (third expr) level
      exitCount ValueFlag)
      t)
    (t
      (|canReturn| (third expr) level exitCount ValueFlag))))
((eq op 'cond)
  (cond
    ((equal level exitCount)
      (let (result)
        (loop for u in (rest expr)
          do (setq result (or result
            (|canReturn| (|last| u) level exitCount ValueFlag))))
        result))
    (t
      (let (outer)
        (loop for v in (rest expr)
          do (setq outer (or outer
            (let (inner)
              (loop for u in v
                do (setq inner
                  (or inner
                    (findThrow gs u level exitCount ValueFlag))))
              inner))))
          outer))))))
((eq op 'if)
  (and (consp expr) (consp (qrest expr)) (consp (qcddr expr))
    (consp (qcdddr expr))
    (eq (qcdddr expr) nil))
  (cond
    ((null (|canReturn| (second expr) 0 0 t))
      (say "IF statement can not cause consequents to be executed")
      (|pp| expr)))
    (or (|canReturn| (second expr) level exitCount nil)
      (|canReturn| (third expr) level exitCount ValueFlag)
      (|canReturn| (fourth expr) level exitCount ValueFlag)))
  ((atom op)
    (let ((result t))
      (loop for u in expr
        do (setq result
          (and result (|canReturn| u level exitCount ValueFlag))))
      result))
  ((and (consp op) (eq (qfirst op) 'xlam) (consp (qrest op))
    (consp (qcddr op)) (eq (qcdddr op) nil))

```

```

(let ((result t))
  (loop for u in expr
    do (setq result
      (and result (|canReturn| u level exitCount ValueFlag))))
  result))
(t (|systemErrorHere| "canReturn"))))

```

5.6.71 defun compBoolean

```

[comp p497]
[getSuccessEnvironment p300]
[getInverseEnvironment p301]

```

— defun compBoolean —

```

(defun |compBoolean| (p mode env)
  (let (tmp1 pp)
    (when (setq tmp1 (OR (|comp| p mode env)))
      (setq pp (car tmp1))
      (setq mode (cadr tmp1))
      (setq env (caddr tmp1))
      (list pp mode (|getSuccessEnvironment| p env)
        (|getInverseEnvironment| p env)))))

```

5.6.72 defun getSuccessEnvironment

```

[qcar p??]
[qcdr p??]
[isDomainForm p329]
[put p??]
[identp p??]
[getProplist p??]
[comp p497]
[consProplistOf p??]
[removeEnv p??]
[addBinding p??]
[get p??]
[$EmptyEnvironment p??]
[$EmptyMode p131]

```

— defun getSuccessEnvironment —

```

(defun |getSuccessEnvironment| (a env)
  (let (id currentProplist tt newProplist x m)
    (declare (special |$EmptyMode| |$EmptyEnvironment|))
    (cond
      ((and (consp a) (eq (qfirst a) '|has|) (CONSP (qrest a))
        (consp (qcddr a)) (eq (qcdddr a) nil))
        (if
          (and (identp (second a)) (|isDomainForm| (third a) |$EmptyEnvironment|))
          (|put| (second a) '|specialCase| (third a) env)
          env))
      ((and (consp a) (eq (qfirst a) '|is|) (consp (qrest a))
        (consp (qcddr a)) (eq (qcdddr a) nil))
        (setq id (qsecond a))
        (setq m (qthird a))
        (cond
          ((and (identp id) (|isDomainForm| m |$EmptyEnvironment|))
            (setq env (|put| id '|specialCase| m env))
            (setq currentProplist (|getProplist| id env))
            (setq tt (|comp| m |$EmptyMode| env))
            (when tt
              (setq env (caddr tt))
              (setq newProplist
                (|consProplistOf| id currentProplist '|value|
                  (cons m (cdr (|removeEnv| tt))))))
              (|addBinding| id newProplist env)))
          (t env)))
      ((and (consp a) (eq (qfirst a) '|case|) (consp (qrest a))
        (consp (qcddr a)) (eq (qcdddr a) nil)
        (identp (qsecond a)))
        (setq x (qsecond a))
        (setq m (qthird a))
        (|put| x '|condition| (cons a (|get| x '|condition| env)) env))
      (t env))))

```

5.6.73 defun getInverseEnvironment

```

[qcar p??]
[qcdr p??]
[identp p??]
[isDomainForm p329]
[put p??]
[get p??]
[member p??]
[mkpf p??]
[delete p??]

```

```
[getUnionMode p303]
[$EmptyEnvironment p??]
```

— **defun getInverseEnvironment** —

```
(defun |getInverseEnvironment| (a env)
  (let (op argl x m oldpred tmp1 zz newpred)
    (declare (special |$EmptyEnvironment|))
    (cond
      ((atom a) env)
      (t
       (setq op (car a))
       (setq argl (cdr a))
       (cond
         ((eq op '|has|)
          (setq x (car argl))
          (setq m (cadr argl))
          (cond
            ((and (identp x) (|isDomainForm| m |$EmptyEnvironment|))
             (|put| x '|specialCase| m env))
            (t env)))
         ((and (consp a) (eq (qfirst a) '|case|) (consp (qrest a))
              (consp (qcddr a)) (eq (qcdddr a) nil)
              (identp (qsecond a)))
          (setq x (qsecond a))
          (setq m (qthird a))
          (setq tmp1 (|get| x '|condition| env))
          (cond
            ((and tmp1 (consp tmp1) (eq (qrest tmp1) nil) (consp (qfirst tmp1))
              (eq (qcaar tmp1) '|or|) (|member| a (qcdr tmp1)))
             (setq oldpred (qcdr tmp1))
             (|put| x '|condition| (list (mkpf (|delete| a oldpred) '|or|) env))
             (t
              (setq tmp1 (|getUnionMode| x env))
              (setq zz (|delete| m (qrest tmp1)))
              (loop for u in zz
                when (and (consp u) (eq (qfirst u) '|:|)
                        (consp (qrest u)) (equal (qsecond u) m))
                do (setq zz (|delete| u zz)))
              (setq newpred
                (mkpf (loop for mp in zz collect (list '|case| x mp)) '|or|)
                (|put| x '|condition|
                  (cons newpred (|get| x '|condition| env) env)))
              (t env)))))))))
```

5.6.74 defun getUnionMode

```
[isUnionMode p303]
[getmode p??]
```

— defun getUnionMode —

```
(defun |getUnionMode| (x env)
  (let (m)
    (setq m (when (atom x) (|getmode| x env)))
    (when m (|isUnionMode| m env))))
```

—————

5.6.75 defun isUnionMode

```
[getmode p??]
[get p??]
```

— defun isUnionMode —

```
(defun |isUnionMode| (m env)
  (let (mp v tmp1)
    (cond
      ((and (consp m) (eq (qfirst m) '|Union|)) m)
      ((progn
         (setq tmp1 (setq mp (|getmode| m env)))
         (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)
              (consp (qrest tmp1)) (eq (qcddr tmp1) nil)
              (consp (qsecond tmp1))
              (eq (qcaadr tmp1) '|UnionCategory|)))
         (second mp))
      ((setq v (|get| (if (eq m '$) '|Rep| m) '|value| env))
       (when (and (consp (car v)) (eq (qfirst (car v)) '|Union|)) (car v))))))
```

—————

5.6.76 defplist compImport plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|import| 'special) '|compImport|))
```

5.6.77 defun compImport

```
[addDomain p228]
[$NoValueMode p131]
```

— defun compImport —

```
(defun |compImport| (form mode env)
  (declare (ignore mode))
  (declare (special |$NoValueMode|))
  (dolist (dom (cdr form)) (setq env (|addDomain| dom env)))
  (list '|/throwAway| |$NoValueMode| env))
```

5.6.78 defplist compIs plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|is| 'special) '|compIs|))
```

5.6.79 defun compIs

```
[comp p497]
[coerce p337]
[$Boolean p??]
[$EmptyMode p131]
```

— defun compIs —

```
(defun |compIs| (form mode env)
  (let (a b aval am tmp1 bval bm td)
    (declare (special |$Boolean| |$EmptyMode|))
    (setq a (second form))
    (setq b (third form))
    (when (setq tmp1 (|comp| a |$EmptyMode| env))
      (setq aval (first tmp1)))
```

```

(setq am (second tmp1))
(setq env (third tmp1))
(when (setq tmp1 (|comp| b |$EmptyMode| env))
  (setq bval (first tmp1))
  (setq bm (second tmp1))
  (setq env (third tmp1))
  (setq td (list (list '|domainEqual| aval bval) |$Boolean| env ))
  (|coerce| td mode))))

```

5.6.80 defplist compJoin plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|Join| 'special) '|compJoin|))

```

5.6.81 defun compJoin

```

[nreverse0 p??]
[compForMode p307]
[stackSemanticError p??]
[nreverse0 p??]
[isCategoryForm p??]
[union p??]
[compJoin,getParms p??]
[qcar p??]
[qcdr p??]
[wrapDomainSub p266]
[convert p504]
[$Category p??]

```

— defun compJoin —

```

(defun |compJoin| (form mode env)
  (labels (
    (getParms (y env)
      (cond
        ((atom y)
          (when (|isDomainForm| y env) (list y)))

```



```

      ((and (consp y) (eq (qfirst y) 'length)
            (consp (qrest y)) (eq (qcddr y) nil))
       (list y (second y)))
      (t (list y)))) )
(let (argl catList pl tmp3 tmp4 tmp5 body parameters catListp td)
(declare (special |$Category|))
(setq argl (cdr form))
(setq catList
  (dolist (x argl (nreverse0 tmp3))
    (push (car (or (|compForMode| x |$Category| env) (return '|failed|)))
      tmp3)))
(cond
  ((eq catList '|failed|)
   (|stackSemanticError| (list '|cannot form Join of: | argl) nil))
  (t
   (setq catListp
     (dolist (x catList (nreverse0 tmp4))
       (setq tmp4
         (cons
          (cond
            ((|isCategoryForm| x env)
             (setq parameters
              (|union|
               (dolist (y (cdr x) tmp5)
                 (setq tmp5 (append tmp5 (getParms y env))))
               parameters)))
            (t
             x))
          ((and (consp x) (eq (qfirst x) '|DomainSubstitutionMacro|)
                (consp (qrest x)) (consp (qcddr x))
                (eq (qcdddr x) nil))
           (setq pl (second x))
           (setq body (third x))
           (setq parameters (|union| pl parameters)) body)
            ((and (consp x) (eq (qfirst x) '|mkCategory|))
             x)
            ((and (atom x) (equal (|getmode| x env) |$Category|))
             x)
            (t
             (|stackSemanticError| (list '|invalid argument to Join: | x) nil)
             x))
          tmp4))))
   (setq td (list (|wrapDomainSub| parameters (cons '|Join| catListp))
                  |$Category| env))
   (|convert| td mode))))))

```

5.6.82 defun compForMode

[comp p497]

[\$compForModeIfTrue p??]

— **defun compForMode** —

```
(defun |compForMode| (x m e)
  (let (|$compForModeIfTrue|)
    (declare (special |$compForModeIfTrue|))
    (setq |$compForModeIfTrue| t)
    (|comp| x m e)))
```

—————

5.6.83 defplist compLambda plist— **postvars** —

```
(eval-when (eval load)
  (setf (get '|+->' 'special) '|compLambda|))
```

—————

5.6.84 defun compLambda

[qcar p??]

[qcdr p??]

[argsToSig p524]

[compAtSign p343]

[stackAndThrow p??]

— **defun compLambda** —

```
(defun |compLambda| (form mode env)
  (let (v1 body tmp1 tmp2 tmp3 target args arg1 sig1 ress)
    (setq v1 (second form))
    (setq body (third form))
    (cond
      ((and (consp v1) (eq (qfirst v1) '|:|))
       (progn
        (setq tmp1 (qrest v1))
        (and (consp tmp1)
```

```

      (progn
        (setq args (qfirst tmp1))
        (setq tmp2 (qrest tmp1))
        (and (consp tmp2)
              (eq (qrest tmp2) nil)
              (progn
                (setq target (qfirst tmp2))
                t))))))
    (when (and (consp args) (eq (qfirst args) '|@Tuple|))
      (setq args (qrest args)))
    (cond
      ((listp args)
       (setq tmp3 (|argsToSig| args))
       (setq arg1 (first tmp3))
       (setq sig1 (second tmp3))
       (cond
         (sig1
          (setq ress
                (compAtSign
                 (list '@
                      (list '+-> arg1 body)
                      (cons '|Mapping| (cons target sig1))) mode env))
                ress)
          (t (|stackAndThrow| (list '|compLambda| form )))))
        (t (|stackAndThrow| (list '|compLambda| form )))))
      (t (|stackAndThrow| (list '|compLambda| form ))))))

```

5.6.85 defplist compLeave plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|leave| 'special) '|compLeave|))

```

5.6.86 defun compLeave

```

[comp p497]
[modifyModeStack p525]
[$exitModeStack p??]
[$leaveLevelStack p??]

```

— **defun compLeave** —

```
(defun |compLeave| (form mode env)
  (let (level x index u)
    (declare (special |$exitModeStack| |$leaveLevelStack|))
    (setq level (second form))
    (setq x (third form))
    (setq index
      (- (1- (|#| |$exitModeStack|)) (elt |$leaveLevelStack| (1- level))))
    (when (setq u (|comp| x (elt |$exitModeStack| index) env))
      (|modifyModeStack| (second u) index)
      (list (list 'TAGGEDexit| index u) mode env ))))
```

5.6.87 defplist compMacro plist

— **postvars** —

```
(eval-when (eval load)
  (setf (get 'mdef 'special) '|compMacro|))
```

5.6.88 defun compMacro

```
[qcar p??]
[formatUnabbreviated p??]
[sayBrightly p??]
[put p??]
[macroExpand p136]
[$macroIfTrue p??]
[$NoValueMode p131]
[$EmptyMode p131]
```

— **defun compMacro** —

```
(defun |compMacro| (form mode env)
  (let (|$macroIfTrue| lhs signature specialCases rhs prhs)
    (declare (special |$macroIfTrue| |$NoValueMode| |$EmptyMode|))
    (setq |$macroIfTrue| t)
    (setq lhs (second form))
```

```

(setq signature (third form))
(setq specialCases (fourth form))
(setq rhs (fifth form))
(setq prhs
  (cond
    ((and (consp rhs) (eq (qfirst rhs) 'category))
      (list "-- the constructor category"))
    ((and (consp rhs) (eq (qfirst rhs) '|Join|))
      (list "-- the constructor category"))
    ((and (consp rhs) (eq (qfirst rhs) 'capsule))
      (list "-- the constructor capsule"))
    ((and (consp rhs) (eq (qfirst rhs) '|add|))
      (list "-- the constructor capsule"))
    (t (|formatUnabbreviated| rhs))))
(|sayBrightly|
  (cons "    processing macro definition"
    (cons '|%b|
      (append (|formatUnabbreviated| lhs)
        (cons " ==> "
          (append prhs (list '|%d|)))))))
(when (or (equal mode |$EmptyMode|) (equal mode |$NoValueMode|))
  (list '|/throwAway| |$NoValueMode|
    (|put| (CAR lhs) '|macro| (|macroExpand| rhs env) env))))

```

5.6.89 defplist compPretend plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|pretend| 'special) '|compPretend|))

```

5.6.90 defun compPretend

```

[addDomain p228]
[comp p497]
[opOf p??]
[nequal p??]
[stackSemanticError p??]
[stackWarning p??]
[$newCompilerUnionFlag p??]

```

[`$EmptyMode` p131]

— **defun compPretend** —

```
(defun |compPretend| (form mode env)
  (let (x tt warningMessage td tp)
    (declare (special |$newCompilerUnionFlag| |$EmptyMode|))
    (setq x (second form))
    (setq tt (third form))
    (setq env (|addDomain| tt env))
    (when (setq td (or (|comp| x tt env) (|comp| x |$EmptyMode| env)))
      (when (equal (second td) tt)
        (setq warningMessage (list '|pretend| tt '| -- should replace by @|)))
      (cond
        ((and |$newCompilerUnionFlag|
              (eq (|opOf| (second td)) '|Union|)
              (nequal (|opOf| mode) '|Union|))
          (|stackSemanticError|
           (list '|cannot pretend | x '| of mode | (second td) '| to mode | mode)
           nil))
        (t
         (setq td (list (first td) tt (third td)))
         (when (setq tp (|coerce| td mode))
           (when warningMessage (|stackWarning| warningMessage)
             tp))))))
```

—————

5.6.91 **defplist compQuote** plist

— **postvars** —

```
(eval-when (eval load)
  (setf (get 'quote 'special) '|compQuote|))
```

—————

5.6.92 **defun compQuote**

— **defun compQuote** —

```
(defun |compQuote| (form mode env)
  (list form mode env))
```

5.6.93 defplist compReduce plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'reduce 'special) '|compReduce|))
```

5.6.94 defun compReduce

```
[compReduce1 p312]
[$formalArgList p??]
```

— defun compReduce —

```
(defun |compReduce| (form mode env)
  (declare (special |$formalArgList|))
  (|compReduce1| form mode env |$formalArgList|))
```

5.6.95 defun compReduce1

```
[systemError p??]
[nreverse0 p??]
[compIterator p??]
[comp p497]
[parseTran p93]
[getIdentity p??]
[msubst p??]
[$sideEffectsList p??]
[$until p??]
[$initList p??]
[$Boolean p??]
[$e p??]
[$endTestList p??]
```

— defun compReduce1 —

```

(defun |compReduce1| (form mode env |$formalArgList|)
  (declare (special |$formalArgList|))
  (let (|$sideEffectsList| |$until| |$initList| |$endTestList| collectForm
        collectOp body op itl acc afterFirst bodyVal part1 part2 part3 id
        identityCode untilCode finalCode tmp1 tmp2)
    (declare (special |$sideEffectsList| |$until| |$initList| |$Boolean| |$e|
                      |$endTestList|))
    (setq op (second form))
    (setq collectForm (fourth form))
    (setq collectOp (first collectForm))
    (setq tmp1 (reverse (cdr collectForm)))
    (setq body (first tmp1))
    (setq itl (nreverse (cdr tmp1)))
    (when (stringp op) (setq op (intern op)))
    (cond
      ((null (member collectOp '(collect collectv collectvec)))
       (|systemError| (list '|illegal reduction form:| form)))
      (t
       (setq |$sideEffectsList| nil)
       (setq |$until| nil)
       (setq |$initList| nil)
       (setq |$endTestList| nil)
       (setq |$e| env)
       (setq itl
        (dolist (x itl (nreverse0 tmp2))
          (setq tmp1 (or (|compIterator| x |$e|) (return '|failed|)))
          (setq |$e| (second tmp1))
          (push (elt tmp1 0) tmp2)))
        (unless (eq itl '|failed|)
          (setq env |$e|)
          (setq acc (gensym))
          (setq afterFirst (gensym))
          (setq bodyVal (gensym))
          (when (setq tmp1 (|comp| (list 'let bodyVal body ) mode env))
            (setq part1 (first tmp1))
            (setq mode (second tmp1))
            (setq env (third tmp1))
            (when (setq tmp1 (|comp| (list 'let acc bodyVal) mode env))
              (setq part2 (first tmp1))
              (setq env (third tmp1))
              (when (setq tmp1
                (|comp| (list 'let acc (|parseTran| (list op acc bodyVal)))
                  mode env))
                (setq part3 (first tmp1))
                (setq env (third tmp1))
                (when (setq identityCode
                  (if (setq id (|getIdentity| op env))
                    (car (|comp| id mode env))
                    (list '|IdentityError| (mkq op))))
                  (setq finalCode

```



```

(cons 'progn
  (cons (list 'let afterFirst nil)
    (cons
      (cons 'repeat
        (append itl
          (list
            (list 'progn part1
              (list 'if afterFirst part3
                (list 'progn part2 (list 'let afterFirst (mkq t)))) nil))))
        (list (list 'if afterFirst acc identityCode ))))))
    (when |$until|
      (setq tmp1 (|comp| |$until| |$Boolean| env))
      (setq untilCode (first tmp1))
      (setq env (third tmp1))
      (setq finalCode
        (msubst (list 'until untilCode) '|$until| finalCode)))
      (list finalCode mode env ))))))))

```

5.6.96 defplist compRepeatOrCollect plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'collect 'special) '|compRepeatOrCollect|))

```

5.6.97 defplist compRepeatOrCollect plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'repeat 'special) '|compRepeatOrCollect|))

```

5.6.98 defun compRepeatOrCollect

```

[length p??]
[compIterator p??]

```

```

[modeIsAggregateOf p??]
[stackMessage p??]
[compOrCroak p496]
[comp p497]
[msubst p??]
[coerceExit p342]
[ p??]
[ p??]
[$until p??]
[$Boolean p??]
[$NoValueMode p131]
[$exitModeStack p??]
[$leaveLevelStack p??]
[$formalArgList p??]

```

— defun compRepeatOrCollect —

```

(defun |compRepeatOrCollect| (form mode env)
  (labels (
    (fn (form |$exitModeStack| |$leaveLevelStack| |$formalArgList| env)
      (declare (special |$exitModeStack| |$leaveLevelStack| |$formalArgList|))
      (let (|$until| body itl xp targetMode repeatOrCollect bodyMode bodyp mp tmp1
            untilCode ep itlp formp u mpp tmp2)
        (declare (special |$Boolean| |$until| |$NoValueMode| ))
        (setq |$until| nil)
        (setq repeatOrCollect (car form))
        (setq tmp1 (reverse (cdr form)))
        (setq body (car tmp1))
        (setq itl (nreverse (cdr tmp1)))
        (setq itlp
          (dolist (x itl (nreverse0 tmp2))
            (setq tmp1 (or (|compIterator| x env) (return '|failed|)))
            (setq xp (first tmp1))
            (setq env (second tmp1))
            (push xp tmp2)))
        (unless (eq itlp '|failed|)
          (setq targetMode (car |$exitModeStack|))
          (setq bodyMode
            (if (eq repeatOrCollect 'collect)
              (cond
                ((eq targetMode '|$EmptyMode|)
                 '|$EmptyMode|)
                ((setq u (|modeIsAggregateOf| '|List| targetMode env))
                 (second u))
                ((setq u (|modeIsAggregateOf| '|PrimitiveArray| targetMode env))
                 (setq repeatOrCollect 'collectv)
                 (second u))
                ((setq u (|modeIsAggregateOf| '|Vector| targetMode env))

```

```

      (setq repeatOrCollect 'collectvec)
      (second u))
    (t
      (|stackMessage| "Invalid collect bodytype")
      '|failed|))
    |$NoValueMode|))
  (unless (eq bodyMode '|failed|)
    (when (setq tmp1 (|compOrCroak| body bodyMode env))
      (setq bodyp (first tmp1))
      (setq mp (second tmp1))
      (setq ep (third tmp1))
      (when |$until|
        (setq tmp1 (|comp| |$until| |$Boolean| ep))
        (setq untilCode (first tmp1))
        (setq ep (third tmp1))
        (setq itlp (msubst (list 'until untilCode) '|$until| itlp)))
      (setq formp (cons repeatOrCollect (append itlp (list bodyp))))
      (setq mpp
        (cond
          ((eq repeatOrCollect 'collect)
            (if (setq u (|modeIsAggregateOf| '|List| targetMode env))
              (car u)
              (list '|List| mp)))
          ((eq repeatOrCollect 'collectv)
            (if (setq u (|modeIsAggregateOf| '|PrimitiveArray| targetMode env))
              (car u)
              (list '|PrimitiveArray| mp)))
          ((eq repeatOrCollect 'collectvec)
            (if (setq u (|modeIsAggregateOf| '|Vector| targetMode env))
              (car u)
              (list '|Vector| mp)))
          (t mp)))
      (|coerceExit| (list formp mpp ep) targetMode))))))
  (declare (special |$exitModeStack| |$leaveLevelStack| |$formalArgList|))
  (fn form
    (cons mode |$exitModeStack|)
    (cons (|#| |$exitModeStack|) |$leaveLevelStack|)
    |$formalArgList|
    env)))

```

5.6.99 defplist compReturn plist

— postvars —

```
(eval-when (eval load)
```

```
(setf (get '|return| 'special) '|compReturn|))
```

5.6.100 defun compReturn

```
[stackSemanticError p??]
[nequal p??]
[userError p??]
[resolve p347]
[comp p497]
[modifyModeStack p525]
[$exitModeStack p??]
[$returnMode p??]
```

— defun compReturn —

```
(defun |compReturn| (form mode env)
  (let (level x index u xp mp ep)
    (declare (special |$returnMode| |$exitModeStack|))
    (setq level (second form))
    (setq x (third form))
    (cond
      ((null |$exitModeStack|)
       (|stackSemanticError|
        (list '|the return before| '|%b| x '|%d| '|is unnecessary|) nil)
       nil)
      ((nequal level 1)
       (|userError| "multi-level returns not supported"))
      (t
       (setq index (max 0 (1- (|#| |$exitModeStack|))))
       (when (>= index 0)
        (setq |$returnMode|
              (|resolve| (elt |$exitModeStack| index) |$returnMode|)))
       (when (setq u (|comp| x |$returnMode| env))
        (setq xp (first u))
        (setq mp (second u))
        (setq ep (third u))
        (when (>= index 0)
         (setq |$returnMode| (|resolve| mp |$returnMode|))
         (|modifyModeStack| mp index))
        (list (list '|TAGGEDreturn| 0 u) mode ep))))))
```

5.6.101 defplist compSeq plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'seq 'special) '|compSeq|))
```

—————

5.6.102 defun compSeq

```
[compSeq1 p318]
[$exitModeStack p??]
```

— defun compSeq —

```
(defun |compSeq| (form mode env)
  (declare (special |$exitModeStack|))
  (|compSeq1| (cdr form) (cons mode |$exitModeStack|) env))
```

—————

5.6.103 defun compSeq1

```
[nreverse0 p??]
[compSeqItem p320]
[mkq p??]
[replaceExitEtc p319]
[$exitModeStack p??]
[$insideExpressionIfTrue p??]
[$finalEnv p??]
[$NoValueMode p131]
```

— defun compSeq1 —

```
(defun |compSeq1| (form |$exitModeStack| env)
  (declare (special |$exitModeStack|))
  (let (|$insideExpressionIfTrue| |$finalEnv| tmp1 tmp2 c catchTag newform)
    (declare (special |$insideExpressionIfTrue| |$finalEnv| |$NoValueMode|))
    (setq |$insideExpressionIfTrue| nil)
    (setq |$finalEnv| nil)
    (when
      (setq c (dolist (x form (nreverse0 tmp2))
```

```

      (setq |$insideExpressionIfTrue| nil)
      (setq tmp1 (|compSeqItem| x |$NoValueMode| env))
      (unless tmp1 (return nil))
      (setq env (third tmp1))
      (push (first tmp1) tmp2)))
(setq catchTag (mkq (gensym)))
(setq newform
  (cons 'seq
    (|replaceExitEtc| c catchTag '|TAGGEDexit| (elt |$exitModeStack| 0))))
(list (list 'catch catchTag newform)
      (elt |$exitModeStack| 0) |$finalEnv|)))

```

5.6.104 defun replaceExitEtc

```

[qcar p??]
[qcdr p??]
[rplac p??]
[replaceExitEtc p319]
[intersectionEnvironment p??]
[convertOrCroak p320]
|$finalEnv p??]
|$finalEnv p??]

```

— defun replaceExitEtc —

```

(defun |replaceExitEtc| (x tag opFlag opMode)
  (declare (special |$finalEnv|))
  (cond
    ((atom x) nil)
    ((and (consp x) (eq (qfirst x) 'quote)) nil)
    ((and (consp x) (equal (qfirst x) opFlag) (consp (qrest x))
      (consp (qcddr x)) (eq (qcdddr x) nil))
      (|rplac| (caaddr x) (|replaceExitEtc| (caaddr x) tag opFlag opMode))
      (cond
        ((eq1 (second x) 0)
          (setq |$finalEnv|
            (if |$finalEnv|
              (|intersectionEnvironment| |$finalEnv| (third (third x)))
              (third (third x))))
          (|rplac| (car x) 'throw)
          (|rplac| (cadr x) tag)
          (|rplac| (caddr x) (car (|convertOrCroak| (caddr x) opMode))))
        (t
          (|rplac| (cadr x) (1- (cadr x))))))
    ((and (consp x) (consp (qrest x)) (consp (qcddr x))

```

```

      (eq (qcdddr x) nil)
      (member (qfirst x) '(|TAGGEDreturn| |TAGGEDexit|)))
    (|rplac| (car (caddr x))
      (|replaceExitEtc| (car (caddr x)) tag opFlag opMode)))
  (t
    (|replaceExitEtc| (car x) tag opFlag opMode)
    (|replaceExitEtc| (cdr x) tag opFlag opMode)))
  x)

```

5.6.105 defun convertOrCroak

[convert p504]
[userError p??]

— defun convertOrCroak —

```

(defun |convertOrCroak| (tt m)
  (let (u)
    (if (setq u (|convert| tt m))
      u
      (|userError|
        (list '|CANNOT CONVERT: | (first tt) '|%1| '| OF MODE: | (second tt)
          '|%1| '| TO MODE: | m '|%1|))))))

```

5.6.106 defun compSeqItem

[comp p497]
[macroExpand p136]

— defun compSeqItem —

```

(defun |compSeqItem| (form mode env)
  (|comp| (|macroExpand| form env) mode env))

```

5.6.107 defplist compSetq plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'let 'special) '|compSetq|))
```

5.6.108 defplist compSetq plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'setq 'special) '|compSetq|))
```

5.6.109 defun compSetq

[compSetq1 p321]

— defun compSetq —

```
(defun |compSetq| (form mode env)
  (|compSetq1| (second form) (third form) mode env))
```

5.6.110 defun compSetq1

[setqSingle p326]
 [compSetq1 identp (vol5)]
 [compMakeDeclaration p525]
 [compSetq p321]
 [qcar p??]
 [qcdr p??]
 [setqMultiple p322]
 [setqSetelt p326]
 [\$EmptyMode p131]

— defun compSetq1 —

```
(defun |compSetq1| (form val mode env)
  (let (x y ep op z)
```



```

(declare (special |$EmptyMode|))
(cond
  ((identp form) (|setqSingle| form val mode env))
  ((and (consp form) (eq (qfirst form) '|:|) (consp (qrest form))
        (consp (qcddr form)) (eq (qcdddr form) nil))
   (setq x (second form))
   (setq y (third form))
   (setq ep (third (|compMakeDeclaration| form |$EmptyMode| env)))
   (|compSetq| (list 'let x val) mode ep))
  ((consp form)
   (setq op (qfirst form))
   (setq z (qrest form))
   (cond
     ((eq op 'cons)      (|setqMultiple| (|uncons| form) val mode env))
     ((eq op '|@Tuple|) (|setqMultiple| z val mode env))
     (t                  (|setqSetelt| form val mode env))))))

```

5.6.111 defun uncons

[uncons p322]

— defun uncons —

```

(defun |uncons| (x)
  (cond
    ((atom x) x)
    ((and (consp x) (eq (qfirst x) 'cons) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
     (cons (second x) (|uncons| (third x)))))

```

5.6.112 defun setqMultiple

[nreverse0 p??]
 [qcar p??]
 [qcdr p??]
 [stackMessage p??]
 [setqMultipleExplicit p324]
 [genVariable p??]
 [addBinding p??]
 [compSetq1 p321]
 [convert p504]

```
[put p??]
[genSomeVariable p??]
[length p??]
[mkprogn p??]
[$EmptyMode p131]
[$NoValueMode p131]
[$noEnv p??]
```

— defun `setqMultiple` —

```
(defun |setqMultiple| (nameList val m env)
  (labels (
    (decompose (tt len env)
      (declare (ignore len))
      (let (tmp1 z)
        (declare (special |$EmptyMode|))
        (cond
          ((and (consp tt) (eq (qfirst tt) '|Record|)
            (progn (setq z (qrest tt)) t))
          (loop for item in z
            collect (cons (second item) (third item))))
        (progn
          (setq tmp1 (|comp| tt |$EmptyMode| env))
          (and (consp tmp1) (consp (qrest tmp1)) (consp (qsecond tmp1))
            (eq (qcaadr tmp1) '|RecordCategory|)
            (consp (qcddr tmp1)) (eq (qcdddr tmp1) nil)))
          (loop for item in z
            collect (cons (second item) (third item))))
        (t (|stackMessage| (list '|no multiple assigns to mode: | tt))))))
    (let (g m1 tt x mp selectorModePairs tmp2 assignList)
      (declare (special |$noEnv| |$EmptyMode| |$NoValueMode|))
      (cond
        ((and (consp val) (eq (qfirst val) 'cons) (equal m |$NoValueMode|))
          (|setqMultipleExplicit| nameList (|uncons| val) m env))
        ((and (consp val) (eq (qfirst val) '|@Tuple|) (equal m |$NoValueMode|))
          (|setqMultipleExplicit| nameList (qrest val) m env))
        ; 1 create a gensym, %add to local environment, compile and assign rhs
        (t
          (setq g (|genVariable|))
          (setq env (|addBinding| g nil env))
          (setq tmp2 (|compSetq1| g val |$EmptyMode| env))
          (when tmp2
            (setq tt tmp2)
            (setq m1 (cadr tmp2))
            (setq env (|put| g 'mode m1 env))
            (setq tmp2 (|convert| tt m))
            ; 1.1 --exit if result is a list
            (when tmp2
              (setq x (first tmp2))
```

```

(setq mp (second tmp2))
(setq env (third tmp2))
(cond
  ((and (consp m1) (eq (qfirst m1) '|List|) (consp (qrest m1))
        (eq (qcddr m1) nil))
   (loop for y in nameList do
     (setq env
       (|put| y '|value| (list (|genSomeVariable|) (second m1) |$noEnv|)
         env)))
   (|convert| (list (list 'progn x (list 'let nameList g) g) mp env) m))
  (t
; 2 --verify that the #nameList = number of parts of right-hand-side
   (setq selectorModePairs
     (decompose m1 (|#| nameList) env))
   (when selectorModePairs
     (cond
       ((nequal (|#| nameList) (|#| selectorModePairs))
        (|stackMessage|
          (list val '| must decompose into |
            (|#| nameList) '| components| )))
       (t
; 3 --generate code
        (setq assignList
          (loop for x in nameList
            for item in selectorModePairs
            collect (car
              (progn
                (setq tmp2
                  (or (|compSetq1| x (list '|elt| g (first item))
                    (rest item) env)
                    (return '|failed|))))
                (setq env (third tmp2))
                tmp2))))
          (unless (eq assignList '|failed|)
            (list (mkprogn (cons x (append assignList (list g)))) mp env)
            )))))))))))

```

5.6.113 defun setqMultipleExplicit

```

[nequal p??]
[stackMessage p??]
[genVariable p??]
[compSetq1 p321]
[last p??]
[$EmptyMode p131]

```

[*\$NoValueMode* p131]

— **defun** *setqMultipleExplicit* —

```
(defun |setqMultipleExplicit| (nameList valList m env)
  (declare (ignore m))
  (let (gensymList assignList tmp1 reAssignList)
    (declare (special |$NoValueMode| |$EmptyMode|))
    (cond
      ((nequal (|#| nameList) (|#| valList))
        (|stackMessage|
          (list '|Multiple assignment error; # of items in: | nameList
            '|must = # in: | valList)))
      (t
        (setq gensymList
          (loop for name in nameList
            collect (|genVariable|)))
        (setq assignList
          (loop for g in gensymList
            for val in valList
            collect (progn
              (setq tmp1
                (or (|compSetq1| g val |$EmptyMode| env)
                  (return '|failed|)))
              (setq env (third tmp1))
              tmp1)))
        (unless (eq assignList '|failed|)
          (setq reAssignList
            (loop for g in gensymList
              for name in nameList
              collect (progn
                (setq tmp1
                  (or (|compSetq1| name g |$EmptyMode| env)
                    (return '|failed|)))
                (setq env (third tmp1))
                tmp1)))
          (unless (eq reAssignList '|failed|)
            (list
              (cons 'progn
                (append
                  (loop for tt in assignList
                    collect (car tt))
                  (loop for tt in reAssignList
                    collect (car tt))))
              |$NoValueMode| (third (|last| reAssignList))))))))))
```

5.6.114 defun setqSetelt

[comp p497]

— **defun setqSetelt** —

```
(defun |setqSetelt| (form val mode env)
  (|comp| (cons '|setelt| (cons (car form) (append (cdr form) (list val))))
    mode env))
```

—————

5.6.115 defun setqSingle

```
[setqSingle getProplist (vol5)]
[getmode p??]
[get p??]
[nequal p??]
[maxSuperType p329]
[comp p497]
[getmode p??]
[assignError p328]
[convert p504]
[setqSingle identp (vol5)]
[profileRecord p??]
[consProplistOf p??]
[removeEnv p??]
[setqSingle addBinding (vol5)]
[isDomainForm p329]
[isDomainInScope p??]
[stackWarning p??]
[augModemapsFromDomain1 p232]
[NRTassocIndex p??]
[isDomainForm p329]
[outputComp p328]
[$insideSetqSingleIfTrue p??]
[$QuickLet p??]
[$form p??]
[$profileCompiler p??]
[$EmptyMode p131]
[$NoValueMode p131]
```

— **defun setqSingle** —

```
(defun |setqSingle| (form val mode env)
```

```

(let (|$insideSetqSingleIfTrue| currentProplist mpp maxmpp td x mp tp key
      newProplist ep k newform)
  (declare (special |$insideSetqSingleIfTrue| |$QuickLet| |$form|
                    |$profileCompiler| |$EmptyMode| |$NoValueMode|))
  (setq |$insideSetqSingleIfTrue| t)
  (setq currentProplist (|getProplist| form env))
  (setq mpp
    (or (|get| form '|mode| env) (|getmode| form env)
      (if (equal mode |$NoValueMode|) |$EmptyMode| mode)))
  (when (setq td
    (cond
      ((setq td (|comp| val mpp env))
       td)
      ((and (null (|get| form '|mode| env))
            (nequal mpp (setq maxmpp (|maxSuperType| mpp env)))
            (setq td (|comp| val maxmpp env)))
       td)
      ((and (setq td (|comp| val |$EmptyMode| env))
            (|getmode| (second td) env))
       (|assignError| val (second td) form mpp))))
    (when (setq tp (|convert| td mode))
      (setq x (first tp))
      (setq mp (second tp))
      (setq ep (third tp))
      (when (and |$profileCompiler| (identp form))
        (setq key (if (member form (cdr |$form|)) '|arguments| '|locals|)
          (|profileRecord| key form (second td)))
        (setq newProplist
          (|consProplistOf| form currentProplist '|value|
            (|removeEnv| (cons val (cdr td)))))
        (setq ep (if (consp form) ep (|addBinding| form newProplist ep)))
        (when (|isDomainForm| val ep)
          (when (|isDomainInScope| form ep)
            (|stackWarning|
              (list '|domain valued variable| '|%b| form '|%d|
                '|has been reassigned within its scope| )))
            (setq ep (|augModemapsFromDomain1| form val ep)))
        (if (setq k (|NRTassocIndex| form))
          (setq newform (list 'setelt '$ k x))
          (setq newform
            (if |$QuickLet|
              (list 'let form x)
              (list 'let form x
                (if (|isDomainForm| x ep)
                  (list 'elt form 0)
                  (car (|outputComp| form ep)))))))
          (list newform mp ep))))

```

```

      result)))
    (nreverse0 result)))
  |$Expression| env))
((and (setq v (|get| x '|value| env))
      (consp (cadr v)) (eq (qfirst (cadr v)) '|Union|))
  (list (list '|coerceUn2E| x (cadr v)) |$Expression| env))
  (t (list x |$Expression| env)))))

```

5.6.118 defun maxSuperType

[get p??]
[maxSuperType p329]

— defun maxSuperType —

```

(defun |maxSuperType| (m env)
  (let (typ)
    (if (setq typ (|get| m '|SuperDomain| env))
        (|maxSuperType| typ env)
        m)))

```

5.6.119 defun isDomainForm

[kar p??]
[qcar p??]
[qcdr p??]
[isFunctor p229]
[isCategoryForm p??]
[isDomainConstructorForm p330]
[\$SpecialDomainNames p??]

— defun isDomainForm —

```

(defun |isDomainForm| (d env)
  (let (tmp1)
    (declare (special |$SpecialDomainNames|))
    (or (member (kar d) |$SpecialDomainNames|) (|isFunctor| d)
        (and (progn
              (setq tmp1 (|getmode| d env))
              (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|) (consp (qrest tmp1)))))

```



```

(|isCategoryForm| (qsecond tmp1) env))
(|isCategoryForm| (|getmode| d env) env)
(|isDomainConstructorForm| d env)))

```

5.6.120 defun isDomainConstructorForm

```

[qcar p??]
[qcdr p??]
[isCategoryForm p??]
[eqsubstlist p??]
[$FormalMapVariableList p242]

```

— defun isDomainConstructorForm —

```

(defun |isDomainConstructorForm| (d env)
  (let (u)
    (declare (special |$FormalMapVariableList|))
    (when
      (and (consp d)
           (setq u (|get| (qfirst d) '|value| env))
           (consp u)
           (consp (qrest u))
           (consp (qsecond u))
           (eq (qcaadr u) '|Mapping|)
           (consp (qcdadr u)))
      (|isCategoryForm|
       (eqsubstlist (rest d) |$FormalMapVariableList| (cadadr u)) env))))

```

5.6.121 defplist compString plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|String| 'special) '|compString|))

```

5.6.122 defun compString

[resolve p347]

[\$StringCategory p??]

— **defun compString** —

```
(defun |compString| (form mode env)
  (declare (special |$StringCategory|))
  (list form (|resolve| |$StringCategory| mode) env))
```

—

5.6.123 defplist compSubDomain plist— **postvars** —

```
(eval-when (eval load)
  (setf (get '|SubDomain| 'special) '|compSubDomain|))
```

—

5.6.124 defun compSubDomain

[compSubDomain1 p332]

[compCapsule p250]

[\$addFormLhs p??]

[\$NRTaddForm p??]

[\$addForm p??]

[\$addFormLhs p??]

— **defun compSubDomain** —

```
(defun |compSubDomain| (form mode env)
  (let (|$addFormLhs| |$addForm| domainForm predicate tmp1)
    (declare (special |$addFormLhs| |$addForm| |$NRTaddForm| |$addFormLhs|))
    (setq domainForm (second form))
    (setq predicate (third form))
    (setq |$addFormLhs| domainForm)
    (setq |$addForm| nil)
    (setq |$NRTaddForm| domainForm)
    (setq tmp1 (|compSubDomain1| domainForm predicate mode env))
    (setq |$addForm| (first tmp1)))
```

```
(setq env (third tmp1))
(|compCapsule| (list 'capsule) mode env)))
```

5.6.125 defun compSubDomain1

```
[compMakeDeclaration p525]
[addDomain p228]
[compOrCroak p496]
[stackSemanticError p??]
[lispize p333]
[evalAndRwriteLispForm p168]
[$CategoryFrame p??]
[$op p??]
[$lisplibSuperDomain p??]
[$Boolean p??]
[$EmptyMode p131]
```

— defun compSubDomain1 —

```
(defun |compSubDomain1| (domainForm predicate mode env)
  (let (u prefixPredicate opp dFp)
    (declare (special |$CategoryFrame| |$op| |$lisplibSuperDomain| |$Boolean|
                      |$EmptyMode|))
    (setq env (third
      (|compMakeDeclaration| (list '|:| '|#1| domainForm)
        |$EmptyMode| (|addDomain| domainForm env))))
    (setq u (|compOrCroak| predicate |$Boolean| env))
    (unless u
      (|stackSemanticError|
        (list '|predicate:| predicate
          '| cannot be interpreted with #1: | domainForm) nil))
    (setq prefixPredicate (|lispize| (first u)))
    (setq |$lisplibSuperDomain| (list domainForm predicate))
    (|evalAndRwriteLispForm| '|evalOnLoad2|
      (list 'setq '|$CategoryFrame|
        (list '|put|
          (setq opp (list 'quote |$op|))
          '|SuperDomain|
          (setq dFp (list 'quote domainForm))
          (list '|put| dFp '|SubDomain|
            (list 'cons (list 'quote (cons |$op| prefixPredicate))
              (list 'delasc opp (list '|get| dFp '|SubDomain| '|$CategoryFrame|))))
          '|$CategoryFrame|))))
      (list domainForm mode env)))
```

5.6.126 defun lispize

[optimize p205]

— defun lispize —

```
(defun |lispize| (x)
  (car (|optimize| (list x))))
```

5.6.127 defplist compSubsetCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|SubsetCategory| 'special) '|compSubsetCategory|))
```

5.6.128 defun compSubsetCategory**TPDHERE:** See LocalAlgebra for an example call [put p??]

[comp p497]

[msubst p??]

[\$lhsOfColon p??]

— defun compSubsetCategory —

```
(defun |compSubsetCategory| (form mode env)
  (let (cat r)
    (declare (special |$lhsOfColon|))
    (setq cat (second form))
    (setq r (third form))
    ; --1. put "Subsets" property on R to allow directly coercion to subset;
    ; -- allow automatic coercion from subset to R but not vice versa
    (setq env (|put| r '|Subsets| (list (list |$lhsOfColon| '|isFalse|)) env))
    ; --2. give the subset domain modemaps of cat plus 3 new functions
    (|comp|
      (list '|Join| cat
            (msubst |$lhsOfColon| '$
```

```

      (list 'category '|domain|
        (list 'signature '|coerce| (list r '$))
        (list 'signature '|lift| (list r '$))
        (list 'signature '|reduce| (list '$ r))))
mode env)))

```

5.6.129 defplist compSuchthat plist

— postvars —

```

(eval-when (eval load)
  (setf (get '| 'special) '|compSuchthat|))

```

5.6.130 defun compSuchthat

```

[comp p497]
[put p??]
[$Boolean p??]

```

— defun compSuchthat —

```

(defun |compSuchthat| (form mode env)
  (let (x p xp mp tmp1 pp)
    (declare (special |$Boolean|))
    (setq x (second form))
    (setq p (third form))
    (when (setq tmp1 (|comp| x mode env))
      (setq xp (first tmp1))
      (setq mp (second tmp1))
      (setq env (third tmp1))
      (when (setq tmp1 (|comp| p |$Boolean| env))
        (setq pp (first tmp1))
        (setq env (third tmp1))
        (setq env (|put| xp '|condition| pp env))
        (list xp mp env))))))

```

5.6.131 defplist compVector plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'vector 'special) '|compVector|))
```

—

5.6.132 defun compVector

```
; null l => [$EmptyVector,m,e]
; T1:= [[.mUnder,e]:= comp(x,mUnder,e) or return "failed" for x in l]
; T1="failed" => nil
; [["VECTOR",:[T.expr for T in T1]],m,e]
```

```
[comp p497]
[$EmptyVector p??]
```

— defun compVector —

```
(defun |compVector| (form mode env)
  (let (tmp1 tmp2 t0 failed (newmode (second mode)))
    (declare (special |$EmptyVector|))
    (if (null form)
      (list |$EmptyVector| mode env)
      (progn
        (setq t0
          (do ((t3 form (cdr t3)) (x nil))
              ((or (atom t3) failed) (unless failed (nreverse0 tmp2)))
            (setq x (car t3))
            (if (setq tmp1 (|comp| x newmode env))
              (progn
                (setq newmode (second tmp1))
                (setq env (third tmp1))
                (push tmp1 tmp2))
              (setq failed t))))))
        (unless failed
          (list (cons 'vector
                     (loop for texpr in t0 collect (car texpr))) mode env))))))
```

—

5.6.133 `defplist compWhere plist`— `postvars` —

```
(eval-when (eval load)
  (setf (get '|where| 'special) '|compWhere|))
```

—————

5.6.134 `defun compWhere`

```
[comp p497]
[macroExpand p136]
[deltaContour p??]
[addContour p??]
[$insideExpressionIfTrue p??]
[$insideWhereIfTrue p??]
[$EmptyMode p131]
```

— `defun compWhere` —

```
(defun |compWhere| (form mode eInit)
  (let (|$insideExpressionIfTrue| |$insideWhereIfTrue| newform exprList e
        eBefore tmp1 x eAfter del eFinal)
    (declare (special |$insideExpressionIfTrue| |$insideWhereIfTrue|
                      |$EmptyMode|))
    (setq newform (second form))
    (setq exprlist (cddr form))
    (setq |$insideExpressionIfTrue| nil)
    (setq |$insideWhereIfTrue| t)
    (setq e eInit)
    (when (dolist (item exprList t)
              (setq tmp1 (|comp| item |$EmptyMode| e))
              (unless tmp1 (return nil))
              (setq e (third tmp1))))
    (setq |$insideWhereIfTrue| nil)
    (setq tmp1 (|comp| (|macroExpand| newform (setq eBefore e)) mode e))
    (when tmp1
      (setq x (first tmp1))
      (setq mode (second tmp1))
      (setq eAfter (third tmp1))
      (setq del (|deltaContour| eAfter eBefore))
      (if del
        (setq eFinal (|addContour| del eInit))
        (setq eFinal eInit))
      (list x mode eFinal))))
```

5.7 Functions for coercion

5.7.1 defun coerce

The function `coerce` is used by the old compiler for coercions. The function `coerceInteractive` is used by the interpreter. One should always call the correct function, since the representation of basic objects may not be the same. [keyedSystemError p??]

```
[rplac p??]
[msubst p??]
[coerceEasy p338]
[coerceSubset p338]
[coerceHard p339]
[isSomeDomainVariable p??]
[stackMessage p??]
[$InteractiveMode p??]
[$Rep p??]
[$fromCoerceable p??]
```

— defun coerce —

```
(defun |coerce| (tt mode)
  (labels (
    (fn (x m1 m2)
      (list '|Cannot coerce| '|%b| x '|%d| '|%l| '|
            of mode| '|%b| m1
            '|%d| '|%l| '|
            to mode| '|%b| m2 '|%d|)))
    (let (tp)
      (declare (special |$fromCoerceable$| |$Rep| |$InteractiveMode|))
      (if |$InteractiveMode|
        (|keyedSystemError| 'S2GE0016
         (list "coerce" "function coerce called from the interpreter."))
        (progn
         (|rplac| (cadr tt) (msubst '$ |$Rep| (cadr tt)))
         (cond
          ((setq tp (|coerceEasy| tt mode)) tp)
          ((setq tp (|coerceSubset| tt mode)) tp)
          ((setq tp (|coerceHard| tt mode)) tp)
          ((or (eq (car tt) '|$fromCoerceable$|) (|isSomeDomainVariable| mode)) nil)
          (t (|stackMessage| (fn (first tt) (second tt) mode))))))))))
```


5.7.2 defun coerceEasy

```
[modeEqualSubst p348]
[$EmptyMode p131]
[$Exit p??]
[$NoValueMode p131]
[$Void p??]
```

— defun coerceEasy —

```
(defun |coerceEasy| (tt m)
  (declare (special |$EmptyMode| |$Exit| |$NoValueMode| |$Void|))
  (cond
    ((equal m |$EmptyMode|) tt)
    ((or (equal m |$NoValueMode|) (equal m |$Void|))
     (list (car tt) m (third tt)))
    ((equal (second tt) m) tt)
    ((equal (second tt) |$NoValueMode|) tt)
    ((equal (second tt) |$Exit|)
     (list
      (list 'progn (car tt) (list '|userError| "Did not really exit."))
      m (third tt)))
    ((or (equal (second tt) |$EmptyMode|)
         (|modeEqualSubst| (second tt) m (third tt)))
     (list (car tt) m (third tt)))))
```

—————

5.7.3 defun coerceSubset

```
[isSubset p??]
[lassoc p??]
[get p??]
[opOf p??]
[eval p??]
[msubst p??]
[isSubset p??]
[maxSuperType p329]
```

— defun coerceSubset —

```
(defun |coerceSubset| (arg1 mp)
  (let (x m env pred)
    (setq x (first arg1))
    (setq m (second arg1))
    (setq env (third arg1))
```

```
(cond
  ((or (|isSubset| m mp env) (and (eq m '|Rep|) (eq mp '$)))
    (list x mp env))
  ((and (consp m) (eq (qfirst m) '|SubDomain|)
    (consp (qrest m)) (equal (qsecond m) mp))
    (list x mp env))
  ((and (setq pred (lassoc (|opOf| mp) (|get| (|opOf| m) '|SubDomain| env)))
    (integerp x) (|eval| (msubst x '|#1| pred)))
    (list x mp env))
  ((and (setq pred (|isSubset| mp (|maxSuperType| m env) env))
    (integerp x) (|eval| (msubst x '* pred)))
    (list x mp env))
  (t nil))))
```

5.7.4 defun coerceHard

```
[modeEqual p348]
[get p??]
[getmode p??]
[isCategoryForm p??]
[extendsCategoryForm p??]
[coerceExtraHard p340]
[$e p??]
[$e p??]
[$String p330]
[$bootStrapMode p??]
```

— defun coerceHard —

```
(defun |coerceHard| (tt m)
  (let (|$e| mp tmp1 mpp)
    (declare (special |$e| |$String| |$bootStrapMode|))
    (setq |$e| (third tt))
    (setq mp (second tt))
    (cond
      ((and (stringp mp) (|modeEqual| m |$String|))
        (list (car tt) m |$e|))
      ((or (|modeEqual| mp m)
        (and (or (progn
          (setq tmp1 (|get| mp '|value| |$e|))
          (and (consp tmp1)
            (progn (setq mpp (qfirst tmp1)) t)))
          (progn
            (setq tmp1 (|getmode| mp |$e|))
            (and (consp tmp1)
```

```

      (eq (qfirst tmp1) '|Mapping|)
      (and (consp (qrest tmp1))
            (eq (qcddr tmp1) nil)
            (progn (setq mpp (qsecond tmp1)) t))))))
    (|modeEqual| mpp m))
  (and (or (progn
            (setq tmp1 (|get| m '|value| |$e|))
            (and (consp tmp1)
                  (progn (setq mpp (qfirst tmp1)) t)))
          (progn
            (setq tmp1 (|getmode| m |$e|))
            (and (consp tmp1)
                  (eq (qfirst tmp1) '|Mapping|)
                  (and (consp (qrest tmp1))
                        (eq (qcddr tmp1) nil)
                        (progn (setq mpp (qsecond tmp1)) t))))))
        (|modeEqual| mpp mp)))
  (list (car tt) m (third tt)))
((and (stringp (car tt)) (equal (car tt) m))
  (list (car tt) m |$e|))
((|isCategoryForm| m |$e|)
  (cond
    ((eq |$bootStrapMode| t)
     (list (car tt) m |$e|))
    ((|extendsCategoryForm| (car tt) (cadr tt) m)
     (list (car tt) m |$e|))
    (t (|coerceExtraHard| tt m))))
(t (|coerceExtraHard| tt m))))

```

5.7.5 defun coerceExtraHard

```

[autoCoerceByModemap p345]
[isUnionMode p303]
[qcar p??]
[qcdr p??]
[hasType p341]
[member p??]
[autoCoerceByModemap p345]
[coerce p337]
[$Expression p??]

```

— defun coerceExtraHard —

```

(defun |coerceExtraHard| (tt m)
  (let (x mp e tmp1 z ta tp tpp)

```

```

(declare (special |$Expression|))
(setq x (first tt))
(setq mp (second tt))
(setq e (third tt))
(cond
  ((setq tp (|autoCoerceByModemap| tt m)) tp)
  ((and (progn
        (setq tmp1 (|isUnionMode| mp e))
        (and (consp tmp1) (eq (qfirst tmp1) '|Union|)
              (progn
                (setq z (qrest tmp1)) t)))
        (setq ta (|hasType| x e))
        (|member| ta z)
        (setq tp (|autoCoerceByModemap| tt ta))
        (setq tpp (|coerce| tp m)))
        tpp)
   ((and (consp mp) (eq (qfirst mp) '|Record|) (equal m |$Expression|))
    (list (list '|coerceRe2E| x (list 'elt (copy mp) 0)) m e))
  (t nil)))

```

5.7.6 defun hasType

[get p??]

— defun hasType —

```

(defun |hasType| (x e)
  (labels (
    (fn (x)
      (cond
        ((null x) nil)
        ((and (consp x) (consp (qfirst x)) (eq (qcaar x) '|case|)
              (consp (qcddar x)) (consp (qcddar x))
              (eq (qcdddar x) nil))
          (qcaddar x))
        (t (fn (cdr x))))))
    (fn (|get| x '|condition| e))))

```

5.7.7 defun coerceable

[pmatch p??]

[sublis p??]

```
[coerce p337]
[$fromCoerceable p??]
```

— **defun coerceable** —

```
(defun |coerceable| (m mp env)
  (let (sl)
    (declare (special |$fromCoerceable$|))
    (cond
      ((equal m mp) m)
      ((setq sl (|pmatch| mp m)) (sublis sl mp))
      ((|coerce| (list '|$fromCoerceable$| m env) mp) mp)
      (t nil))))
```

—————

5.7.8 defun coerceExit

```
[resolve p347]
[replaceExitEsc p??]
[coerce p337]
[$exitMode p??]
```

— **defun coerceExit** —

```
(defun |coerceExit| (arg1 mp)
  (let (x m e catchTag xp)
    (declare (special |$exitMode|))
    (setq x (first arg1))
    (setq m (second arg1))
    (setq e (third arg1))
    (setq mp (|resolve| m mp))
    (setq xp
      (|replaceExitEtc| x
        (setq catchTag (mkq (gensym))) '|TAGGEDexit| |$exitMode|))
    (|coerce| (list (list 'catch catchTag xp) m e) mp)))
```

—————

5.7.9 defplist compAtSign plist

— **postvars** —

```
(eval-when (eval load)
```

```
(setf (get '|@| 'special) 'compAtSign))
```

5.7.10 defun compAtSign

```
[addDomain p228]
[comp p497]
[coerce p337]
```

— defun compAtSign —

```
(defun compAtSign (form mode env)
  (let ((newform (second form)) (mprime (third form)) tmp)
    (setq env (|addDomain| mprime env))
    (when (setq tmp (|comp| newform mprime env)) (|coerce| tmp mode))))
```

5.7.11 defplist compCoerce plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|::| 'special) '|compCoerce|))
```

5.7.12 defun compCoerce

```
[addDomain p228]
[getmode p??]
[compCoerce1 p344]
[coerce p337]
```

— defun compCoerce —

```
(defun |compCoerce| (form mode env)
  (let (newform newmode tmp1 tmp4 z td)
    (setq newform (second form))
    (setq newmode (third form))
```

```

(setq env (|addDomain| newmode env))
(setq tmp1 (|getmode| newmode env))
(cond
  ((setq td (|compCoerce1| newform newmode env))
   (|coerce| td mode))
  ((and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)
        (consp (qrest tmp1)) (eq (qcddr tmp1) nil)
        (consp (qsecond tmp1))
        (eq (qcaadr tmp1) '|UnionCategory|)))
   (setq z (qcaddr tmp1))
   (when
    (setq td
      (dolist (mode1 z tmp4)
        (setq tmp4 (or tmp4 (|compCoerce1| newform mode1 env))))))
    (|coerce| (list (car td) newmode (third td)) mode))))

```

5.7.13 defun compCoerce1

```

[comp p497]
[resolve p347]
[coerce p337]
[coerceByModemap p345]
[msubst p??]
[mkq p??]

```

— defun compCoerce1 —

```

(defun |compCoerce1| (form mode env)
  (let (m1 td tp gg pred code)
    (declare (special |$String| |$EmptyMode|))
    (when (setq td (or (|comp| form mode env) (|comp| form |$EmptyMode| env)))
      (setq m1 (if (stringp (second td)) |$String| (second td)))
      (setq mode (|resolve| m1 mode))
      (setq td (list (car td) m1 (third td)))
      (cond
        ((setq tp (|coerce| td mode)) tp)
        ((setq tp (|coerceByModemap| td mode)) tp)
        ((setq pred (|isSubset| mode (second td) env))
         (setq gg (gensym))
         (setq pred (msubst gg '* pred))
         (setq code
           (list 'prog1
             (list 'let gg (first td))
             (cons '|check-subtype| (cons pred (list (mkq mode) gg))))))
         (list code mode (third td))))))

```

5.7.14 defun coerceByModemap

```
[qcar p??]
[qcdr p??]
[modeEqual p348]
[isSubset p??]
[genDeltaEntry p??]
```

— defun coerceByModemap —

```
(defun |coerceByModemap| (arg1 mp)
  (let (x m env map cexpr u mm fn)
    (setq x (first arg1))
    (setq m (second arg1))
    (setq env (third arg1))
    (setq u
      (loop for modemap in (|getModemapList| ' |coerce| 1 env)
        do
          (setq map (first modemap))
          (setq cexpr (second modemap))
          when
            (and (consp map) (consp (qrest map))
                 (consp (qcddr map))
                 (eq (qcdddr map) nil)
                 (or (|modeEqual| (second map) mp) (|isSubset| (second map) mp env))
                 (or (|modeEqual| (third map) m) (|isSubset| m (third map) env)))
            collect modemap))
    (when u
      (setq mm (first u))
      (setq fn (|genDeltaEntry| (cons ' |coerce| mm)))
      (list (list ' |call| fn x) mp env))))
```

5.7.15 defun autoCoerceByModemap

```
[qcar p??]
[qcdr p??]
[getModemapList p235]
[modeEqual p348]
[member p??]
[get p??]
```



```
[stackMessage p??]
[$fromCoerceable p??]
```

— **defun autoCoerceByModemap** —

```
(defun |autoCoerceByModemap| (arg1 target)
  (let (x source e map cexpr u fn y)
    (declare (special |$fromCoerceable$|))
    (setq x (first arg1))
    (setq source (second arg1))
    (setq e (third arg1))
    (setq u
      (loop for modemap in (|getModemapList| '|autoCoerce| 1 e)
        do
          (setq map (first modemap))
          (setq cexpr (second modemap))
          when
            (and (consp map) (consp (qrest map)) (consp (qcddr map))
              (eq (qcdddr map) nil)
              (|modeEqual| (second map) target)
              (|modeEqual| (third map) source))
            collect cexpr))
    (when u
      (setq fn
        (let (result)
          (loop for item in u
            do
              (when (first item) (setq result (or result (second item))))))
        result))
    (when fn
      (cond
        ((and (consp source) (eq (qfirst source) '|Union|)
          (|member| target (qrest source)))
          (cond
            ((and (setq y (|get| x '|condition| e))
              (let (result)
                (loop for u in y do
                  (setq result
                    (or result
                      (and (consp u) (eq (qfirst u) '|case|) (consp (qrest u))
                        (consp (qcddr u))
                        (eq (qcdddr u) nil)
                        (equal (qthird u) target))))))
              result))
            (list (list '|call| fn x) target e))
          ((eq x '|$fromCoerceable$|) nil)
          (t
            (|stackMessage|
              (list '|cannot coerce: | x '|%1| '| of mode: | source
                '|%1| '| to: | target '| without a case statement|))))))
```

```
(t
  (list (list '|call| fn x) target e))))))
```

5.7.16 defun resolve

```
[nequal p??]
[modeEqual p348]
[mkUnion p347]
[$String p330]
[$EmptyMode p131]
[$NoValueMode p131]
```

— defun resolve —

```
(defun |resolve| (din dout)
  (declare (special |$String| |$EmptyMode| |$NoValueMode|))
  (cond
    ((or (equal din |$NoValueMode|) (equal dout |$NoValueMode|)) |$NoValueMode|)
    ((equal dout |$EmptyMode|) din)
    ((and (nequal din dout) (or (stringp din) (stringp dout)))
     (cond
       ((|modeEqual| dout |$String|) dout)
       ((|modeEqual| din |$String|) nil)
       (t (|mkUnion| din dout))))
    (t dout)))
```

5.7.17 defun mkUnion

```
[qcar p??]
[qcdr p??]
[union p??]
[$Rep p??]
```

— defun mkUnion —

```
(defun |mkUnion| (a b)
  (declare (special |$Rep|))
  (cond
    ((and (eq b '$) (consp |$Rep|) (eq (qfirst |$Rep|) '|Union|))
     (qrest |$Rep|))
```

```

((and (consp a) (eq (qfirst a) '|Union|))
 (cond
  ((and (consp b) (eq (qfirst b) '|Union|))
   (cons '|Union| (|union| (qrest a) (qrest b))))
  (t (cons '|Union| (|union| (list b) (qrest a))))))
((and (consp b) (eq (qfirst b) '|Union|))
 (cons '|Union| (|union| (list a) (qrest b))))
(t (list '|Union| a b))))

```

5.7.18 defun This orders Unions

This orders Unions

— **defun modeEqual** —

```

(defun |modeEqual| (x y)
  (let (xl yl)
    (cond
     ((or (atom x) (atom y)) (equal x y))
     ((nequal (|#| x) (|#| y)) nil)
     ((and (consp x) (eq (qfirst x) '|Union|) (consp y) (eq (qfirst y) '|Union|))
      (setq xl (qrest x))
      (setq yl (qrest y))
      (loop for a in xl do
        (loop for b in yl do
          (when (|modeEqual| a b)
            (setq xl (|delete| a xl))
            (setq yl (|delete| b yl))
            (return nil))))))
     (unless (or xl yl) t))
    (t
     (let ((result t))
       (loop for u in x for v in y
         do (setq result (and result (|modeEqual| u v))))
       result))))

```

5.7.19 defun modeEqualSubst

```

[modeEqual p348]
[modeEqualSubst p348]
[length p??]

```

— **defun modeEqualSubst** —

```
(defun |modeEqualSubst| (m1 m env)
  (let (mp op z1 z2)
    (cond
      ((|modeEqual| m1 m) t)
      ((atom m1)
       (when (setq mp (car (|get| m1 '|value| env)))
         (|modeEqual| mp m)))
      ((and (consp m1) (consp m) (equal (qfirst m) (qfirst m1))
        (equal (|#| (qrest m1)) (|#| (qrest m))))
       (setq op (qfirst m1))
       (setq z1 (qrest m1))
       (setq z2 (qrest m))
       (let ((result t))
         (loop for xm1 in z1 for xm2 in z2
           do (setq result (and result (|modeEqualSubst| xm1 xm2 env))))
       result))
    (t nil))))
```

5.7.20 compilerDoitWithScreenedLisplib

```
compilerDoitWithScreenedLisplib [embed p??]
[rewrite p??]
[compilerDoit p478]
[unembed p??]
[$saveableItems p??]
[$libFile p??]
```

— defun compilerDoitWithScreenedLisplib —

```
(defun |compilerDoitWithScreenedLisplib| (constructor fun)
  (declare (special |$saveableItems| |$libFile|))
  (embed 'rewrite
    '(lambda (key value stream)
      (cond
        ((and (eq stream |$libFile|)
          (not (member key |$saveableItems|)))
         value)
        ((not nil) (rewrite key value stream))))))
  (unwind-protect
    (|compilerDoit| constructor fun)
    (unembed 'rewrite)))
```

Chapter 6

Post Transformers

6.1 Direct called postparse routines

6.1.1 defun postTransform

```
[postTran p352]  
[postTransform identp (vol5)]  
[postTransformCheck p355]  
[aplTran p387]
```

— defun postTransform —

```
(defun |postTransform| (y)  
  (let (x tmp1 tmp2 tmp3 tmp4 tmp5 tt l u)  
    (setq x y)  
    (setq u (|postTran| x))  
    (when  
      (and (consp u) (eq (qfirst u) '|@Tuple|))  
      (progn  
        (setq tmp1 (qrest u))  
        (and (consp tmp1)  
              (progn (setq tmp2 (reverse tmp1)) t)  
              (consp tmp2)  
              (progn  
                (setq tmp3 (qfirst tmp2))  
                (and (consp tmp3)  
                      (eq (qfirst tmp3) '|:|')  
                      (progn  
                        (setq tmp4 (qrest tmp3))  
                        (and (consp tmp4)  
                              (progn  
                                (setq y (qfirst tmp4))
```

```

                (setq tmp5 (qrest tmp4))
                (and (consp tmp5)
                     (eq (qrest tmp5) nil)
                     (progn (setq tt (qfirst tmp5)) t))))))
            (progn (setq l (qrest tmp2)) t)
            (progn (setq l (nreverse l)) t)))
        (dolist (x l t) (unless (identp x) (return nil))))
    (setq u (list '|:| (cons 'listof (append l (list y)) tt)))
    (|postTransformCheck| u)
    (|aplTran| u)))

```

6.1.2 defun postTran

```

[postAtom p353]
[postTran p352]
[qcar p??]
[qcdr p??]
[unTuple p395]
[postTranList p354]
[postForm p356]
[postOp p353]
[postScriptsForm p354]

```

— defun postTran —

```

(defun |postTran| (x)
  (let (op f tmp1 a tmp2 tmp3 b y)
    (if (atom x)
        (|postAtom| x)
        (progn
         (setq op (car x))
         (cond
          ((and (atom op) (setq f (get1 op '|postTran|)))
           (funcall f x))
          ((and (consp op) (eq (qfirst op) '|elt|))
           (progn
            (setq tmp1 (qrest op))
            (and (consp tmp1)
                 (progn
                  (setq a (qfirst tmp1))
                  (setq tmp2 (qrest tmp1))
                  (and (consp tmp2)
                       (eq (qrest tmp2) nil)
                       (progn (setq b (qfirst tmp2)) t))))))
            (cons (|postTran| op) (cdr (|postTran| (cons b (cdr x)))))))

```

```

((and (consp op) (eq (qfirst op) '|Scripts|))
  (|postScriptsForm| op
    (dolist (y (rest x) tmp3)
      (setq tmp3 (append tmp3 (|unTuple| (|postTran| y))))))
  ((nequal op (setq y (|postOp| op)))
    (cons y (|postTranList| (cdr x)))
    (t (|postForm| x))))))

```

6.1.3 defun postOp

— defun postOp —

```

(defun |postOp| (x)
  (declare (special $boot))
  (cond
    ((eq x '|:=|) (if $boot 'spadlet 'let))
    ((eq x '|:-|) 'letd)
    ((eq x '|Attribute|) 'attribute)
    (t x)))

```

6.1.4 defun postAtom

[\$boot p??]

— defun postAtom —

```

(defun |postAtom| (x)
  (declare (special $boot))
  (cond
    ($boot x)
    ((eq1 x 0) '|Zero|)
    ((eq1 x 1) '|One|)
    ((eq x t) 't$)
    ((and (identp x) (getdatabase x 'niladic)) (list x))
    (t x)))

```

6.1.5 defun postTranList

[postTran p352]

— defun postTranList —

```
(defun |postTranList| (x)
  (loop for y in x collect (|postTran| y)))
```

—————

6.1.6 defun postScriptsForm

[getScriptName p391]

[length p??]

[postTranScripts p354]

— defun postScriptsForm —

```
(defun |postScriptsForm| (form arg1)
  (let ((op (second form)) (a (third form)))
    (cons (|getScriptName| op a (|#| arg1))
          (append (|postTranScripts| a) arg1))))
```

—————

6.1.7 defun postTranScripts

[postTranScripts p354]

[postTran p352]

— defun postTranScripts —

```
(defun |postTranScripts| (a)
  (labels (
    (fn (x)
      (if (and (consp x) (eq (qfirst x) '|@Tuple|))
          (qrest x)
          (list x))))
    (let (tmp1 tmp2 tmp3)
      (cond
        ((and (consp a) (eq (qfirst a) '|PrefixSC|))
         (progn
          (setq tmp1 (qrest a))
```

```

      (and (consp tmp1) (eq (qrest tmp1) nil))))
    (|postTranScripts| (qfirst tmp1)))
  ((and (consp a) (eq (qfirst a) '|;|))
   (dolist (y (qrest a) tmp2)
    (setq tmp2 (append tmp2 (|postTranScripts| y)))))
  ((and (consp a) (eq (qfirst a) '|,|))
   (dolist (y (qrest a) tmp3)
    (setq tmp3 (append tmp3 (fn (|postTran| y))))))
  (t (list (|postTran| a)))))

```

6.1.8 defun postTransformCheck

[postcheck p355]
 [\$defOp p??]

— defun postTransformCheck —

```

(defun |postTransformCheck| (x)
  (let (|$defOp|)
    (declare (special |$defOp|))
    (setq |$defOp| nil)
    (|postcheck| x)))

```

6.1.9 defun postcheck

[setDefOp p386]
 [postcheck p355]

— defun postcheck —

```

(defun |postcheck| (x)
  (cond
    ((atom x) nil)
    ((and (consp x) (eq (qfirst x) 'def) (consp (qrest x)))
     (|setDefOp| (qsecond x))
     (|postcheck| (qcddr x)))
    ((and (consp x) (eq (qfirst x) 'quote)) nil)
    (t (|postcheck| (car x)) (|postcheck| (cdr x)))))

```

6.1.10 defun postError

```
[nequal p??]
[bumperrorcount p457]
[$defOp p??]
[$InteractiveMode p??]
[$postStack p??]
```

— defun postError —

```
(defun |postError| (msg)
  (let (xmsg)
    (declare (special |$defOp| |$postStack| |$InteractiveMode|))
    (bumperrorcount '|precompilation|)
    (setq xmsg
      (if (and (nequal |$defOp| '|$defOp|) (null |$InteractiveMode|))
          (cons |$defOp| (cons ": " msg))
          msg))
    (push xmsg |$postStack|)
    nil))
```

6.1.11 defun postForm

```
[postTranList p354]
[internal p??]
[postTran p352]
[postError p356]
[bright p??]
[$boot p??]
```

— defun postForm —

```
(defun |postForm| (u)
  (let (op argl arglp numOfArgs opp x)
    (declare (special $boot))
    (seq
      (setq op (car u))
      (setq argl (cdr u))
      (setq x
        (cond
          ((atom op)
           (setq arglp (|postTranList| argl))
           (setq opp
             (seq
```

```

(exit op)
(when $boot (exit op))
(when (or (get1 op '|Led|) (get1 op '|Nud|) (eq op 'in)) (exit op))
(setq numOfArgs
  (cond
    ((and (consp arglp) (eq (qrest arglp) nil) (consp (qfirst arglp))
      (eq (qcaar arglp) '@Tuple|))
      (|#| (qcdar arglp)))
    (t 1)))
(internal '* (princ-to-string numOfArgs) (pname op)))
(cons opp arglp))
((and (consp op) (eq (qfirst op) '|Scripts|))
  (append (|postTran| op) (|postTranList| argl)))
(t
  (setq u (|postTranList| u))
  (cond
    ((and (consp u) (consp (qfirst u)) (eq (qcaar u) '@Tuple|))
      (|postError|
        (cons " "
          (append (|bright| u)
            (list "is illegal because tuples cannot be applied!" '|%1|
              " Did you misuse infix dot?")))))
    (t u)))
(cond
  ((and (consp x) (consp (qrest x)) (eq (qcddr x) nil)
    (consp (qsecond x)) (eq (qcaadr x) '@Tuple|))
    (cons (car x) (qcdadr x)))
  (t x))))

```

6.2 Indirect called postparse routines

In the **postTran** function there is the code:

```

((and (atom op) (setq f (get1 op '|postTran|)))
  (funcall f x))

```

The functions in this section are called through the symbol-plist of the symbol being parsed. The original list read:

add	postAdd
@	postAtSign
:BF:	postBigFloat
Block	postBlock
CATEGORY	postCategory

COLLECT	postCollect	
:	postColon	
::	postColonColon	
,	postComma	
construct	postConstruct	
==	postDef	
=>	postExit	
if	postIf	
in	postIn	;" the infix operator version of in"
IN	postIn	;" the iterator form of in"
Join	postJoin	
->	postMapping	
==>	postMDef	
pretend	postPretend	
QUOTE	postQUOTE	
Reduce	postReduce	
REPEAT	postRepeat	
Scripts	postScripts	
;	postSemiColon	
Signature	postSignature	
/	postSlash	
@Tuple	postTuple	
TupleCollect	postTupleCollect	
where	postWhere	
with	postWith	

6.2.1 defplist postAdd plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|add| '|postTran|) '|postAdd|))
```

—————

6.2.2 defun postAdd

[postTran p352]
[postCapsule p359]

— defun postAdd —

```
(defun |postAdd| (arg)
  (if (null (cddr arg))
      (|postCapsule| (second arg))
```

```
(list '|add| (|postTran| (second arg)) (|postCapsule| (third arg))))
```

6.2.3 defun postCapsule

```
[checkWarning p461]
[postBlockItem p360]
[postBlockItemList p359]
[postFlatten p368]
```

— defun postCapsule —

```
(defun |postCapsule| (x)
  (let (op)
    (cond
      ((null (and (consp x) (progn (setq op (qfirst x)) t)))
        (|checkWarning| (list "Apparent indentation error following add")))
      ((or (integerp op) (eq op '==))
        (list 'capsule (|postBlockItem| x)))
      ((eq op '|;|)
        (cons 'capsule (|postBlockItemList| (|postFlatten| x '|;|))))
      ((eq op '|if|)
        (list 'capsule (|postBlockItem| x)))
      (t (|checkWarning| (list "Apparent indentation error following add")))))
```

6.2.4 defun postBlockItemList

```
[postBlockItem p360]
```

— defun postBlockItemList —

```
(defun |postBlockItemList| (args)
  (let (result)
    (dolist (item args (nreverse result))
      (push (|postBlockItem| item) result))))
```

6.2.5 defun postBlockItem

[postTran p352]

— defun postBlockItem —

```
(defun |postBlockItem| (x)
  (let ((tmp1 t) tmp2 y tt z)
    (setq x (|postTran| x))
    (if
      (and (consp x) (eq (qfirst x) '|@Tuple|))
      (progn
        (and (consp (qrest x))
          (progn (setq tmp2 (reverse (qrest x))) t)
          (consp tmp2)
          (progn
            (and (consp (qfirst tmp2)) (eq (qcaar tmp2) '|:|))
            (progn
              (and (consp (qcddar tmp2))
                (progn
                  (setq y (qcadar tmp2))
                  (and (consp (qcddar tmp2))
                    (eq (qcdddar tmp2) nil)
                    (progn (setq tt (qcaddar tmp2)) t))))))
              (progn (setq z (qrest tmp2)) t)
              (progn (setq z (nreverse z)) T)))
            (do ((tmp6 nil (null tmp1)) (tmp7 z (cdr tmp7)) (x nil))
              ((or tmp6 (atom tmp7)) tmp1)
              (setq x (car tmp7))
              (setq tmp1 (and tmp1 (identp x))))
            (cons '|:| (cons (cons 'listof (append z (list y))) (list tt)))
            x)))
      (progn
        (progn (setq z (qrest tmp2)) t)
        (progn (setq z (nreverse z)) T)))
      (do ((tmp6 nil (null tmp1)) (tmp7 z (cdr tmp7)) (x nil))
        ((or tmp6 (atom tmp7)) tmp1)
        (setq x (car tmp7))
        (setq tmp1 (and tmp1 (identp x))))
      (cons '|:| (cons (cons 'listof (append z (list y))) (list tt)))
      x)))
```

— —

6.2.6 defplist postAtSign plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|@'|postTran|) '|postAtSign|))
```

— —

6.2.7 defun postAtSign

[postTran p352]
[postType p361]

— defun postAtSign —

```
(defun |postAtSign| (arg)
  (cons '@ (cons (|postTran| (second arg)) (|postType| (third arg)))))
```

—————

6.2.8 defun postType

[postTran p352]
[unTuple p395]

— defun postType —

```
(defun |postType| (typ)
  (let (source target)
    (cond
      ((and (consp typ) (eq (qfirst typ) '->) (consp (qrest typ))
        (consp (qcddr typ)) (eq (qcddr typ) nil))
       (setq source (qsecond typ))
       (setq target (qthird typ))
       (cond
         ((eq source '|constant|)
          (list (list (|postTran| target)) '|constant|))
         (t
          (list (cons '|Mapping|
                     (cons (|postTran| target)
                           (|unTuple| (|postTran| source)))))))
      ((and (consp typ) (eq (qfirst typ) '->)
        (consp (qrest typ)) (eq (qcddr typ) nil))
       (list (list '|Mapping| (|postTran| (qsecond typ)))))
      (t (list (|postTran| typ)))))
```

—————

6.2.9 defplist postBigFloat plist

— postvars —


```
(eval-when (eval load)
  (setf (get '|:BF:| '|postTran|) '|postBigFloat|))
```

6.2.10 defun postBigFloat

```
[postTran p352]
[$boot p??]
[$InteractiveMode p??]
```

— defun postBigFloat —

```
(defun |postBigFloat| (arg)
  (let (mant expon eltword)
    (declare (special $boot |$InteractiveMode|))
    (setq mant (second arg))
    (setq expon (cddr arg))
    (if $boot
        (times (float mant) (expt (float 10) expon))
        (progn
          (setq eltword (if |$InteractiveMode| '|$elt| '|elt|))
          (|postTran|
            (list (list eltword '(|Float|) '|float|)
                  (list '|,| (list '|,| mant expon) 10)))))))
```

6.2.11 defplist postBlock plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Block| '|postTran|) '|postBlock|))
```

6.2.12 defun postBlock

```
[postBlockItemList p359]
[postTran p352]
```

— defun postBlock —

```
(defun |postBlock| (arg)
  (let (tmp1 x y)
    (setq tmp1 (reverse (cdr arg)))
    (setq x (car tmp1))
    (setq y (nreverse (cdr tmp1)))
    (cons 'seq
      (append (|postBlockItemList| y) (list (list 'exit| (|postTran| x)))))))
```

6.2.13 defplist postCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'category '|postTran|) '|postCategory|))
```

6.2.14 defun postCategory

```
[postTran p352]
[nreverse0 p??]
[$insidePostCategoryIfTrue p??]
```

— defun postCategory —

```
(defun |postCategory| (u)
  (declare (special |$insidePostCategoryIfTrue|))
  (labels (
    (fn (arg)
      (let (|$insidePostCategoryIfTrue|)
        (declare (special |$insidePostCategoryIfTrue|))
        (setq |$insidePostCategoryIfTrue| t)
        (|postTran| arg))) )
    (let ((z (cdr u)) op tmp1)
      (if (null z)
        u
        (progn
          (setq op (if |$insidePostCategoryIfTrue| 'progn 'category))
          (cons op (dolist (x z (nreverse0 tmp1)) (push (fn x) tmp1))))))))
```

6.2.15 defun postCollect,finish

```
[qcar p??]
[qcdr p??]
[postMakeCons p364]
[tuple2List p461]
[postTranList p354]
```

— defun postCollect,finish —

```
(defun |postCollect,finish| (op itl y)
  (let (tmp2 tmp5 newBody)
    (cond
      ((and (consp y) (eq (qfirst y) '|:|)
            (consp (qrest y)) (eq (qcddr y) nil))
        (list 'reduce '|append| 0 (cons op (append itl (list (qsecond y))))))
      ((and (consp y) (eq (qfirst y) '|Tuple|))
        (setq newBody
          (cond
            ((dolist (x (qrest y) tmp2)
              (setq tmp2
                (or tmp2 (and (consp x) (eq (qfirst x) '|:|)
                              (consp (qrest x)) (eq (qcddr x) nil))))))
            (|postMakeCons| (qrest y)))
          ((dolist (x (qrest y) tmp5)
            (setq tmp5 (or tmp5 (and (consp x) (eq (qfirst x) 'segment))))))
            (|tuple2List| (qrest y)))
          (t (cons '|construct| (|postTranList| (qrest y))))))
        (list 'reduce '|append| 0 (cons op (append itl (list newBody))))))
      (t (cons op (append itl (list y))))))
```

—

6.2.16 defun postMakeCons

```
[postMakeCons p364]
[postTran p352]
```

— defun postMakeCons —

```
(defun |postMakeCons| (args)
  (let (a b)
    (cond
      ((null args) '|nil|)
      ((and (consp args) (consp (qfirst args)) (eq (qcaar args) '|:|)
            (consp (qcдар args)) (eq (qcddar args) nil))
        (setq a (qcadar args))
```

```

(setq b (qrest args))
(if b
  (list 'append| (|postTran| a) (|postMakeCons| b))
  (|postTran| a)))
(t (list 'cons| (|postTran| (car args)) (|postMakeCons| (cdr args))))))

```

6.2.17 defplist postCollect plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'collect '|postTran|) '|postCollect|))

```

6.2.18 defun postCollect

```

[postCollect,finish p364]
[postCollect p365]
[postIteratorList p366]
[postTran p352]

```

— defun postCollect —

```

(defun |postCollect| (arg)
  (let (constructOp tmp3 m itl x)
    (setq constructOp (car arg))
    (setq tmp3 (reverse (cdr arg)))
    (setq x (car tmp3))
    (setq m (nreverse (cdr tmp3)))
    (cond
      ((and (consp x) (consp (qfirst x)) (eq (qcaar x) '|elt|)
            (consp (qcдар x)) (consp (qcddar x))
            (eq (qcdddar x) nil)
            (eq (qcaddar x) '|construct|))
        (|postCollect|
         (cons (list '|elt| (qcadar x) 'collect)
               (append m (list (cons '|construct| (qrest x)))))))
      (t
       (setq itl (|postIteratorList| m))
       (setq x
         (if (and (consp x) (eq (qfirst x) '|construct|)

```

```

      (cons (qrest x) (eq (qcddr x) nil))
    (qsecond x)
    x))
  (|postCollect,finish| constructOp itl (|postTran| x))))))

```

6.2.19 defun postIteratorList

[postTran p352]
 [postInSeq p374]
 [postIteratorList p366]

— defun postIteratorList —

```

(defun |postIteratorList| (args)
  (let (z p y u a b)
    (cond
      ((cons (consp args)
        (setq p (|postTran| (qfirst args)))
        (setq z (qrest args))
        (cond
          ((and (consp p) (eq (qfirst p) 'in) (consp (qrest p))
            (consp (qcddr p)) (eq (qcdddr p) nil))
            (setq y (qsecond p))
            (setq u (qthird p))
            (cond
              ((and (consp u) (eq (qfirst u) '|\\|') (consp (qrest u))
                (consp (qcddr u)) (eq (qcdddr u) nil))
                (setq a (qsecond u))
                (setq b (qthird u))
                (cons (list 'in y (|postInSeq| a))
                  (cons (list '|\\|' b)
                    (|postIteratorList| z))))
              (t (cons (list 'in y (|postInSeq| u)) (|postIteratorList| z))))
            (t (cons p (|postIteratorList| z))))
          (t args))))

```

6.2.20 defplist postColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '||| 'postTran|) 'postColon|))
```

6.2.21 defun postColon

```
[postTran p352]
[postType p361]
```

— defun postColon —

```
(defun |postColon| (u)
  (cond
    ((and (consp u) (eq (qfirst u) '|||)
      (consp (qrest u)) (eq (qcddr u) nil))
      (list '||| (|postTran| (qsecond u))))
    ((and (consp u) (eq (qfirst u) '|||) (consp (qrest u))
      (consp (qcddr u)) (eq (qcdddr u) nil))
      (cons '||| (cons (|postTran| (second u)) (|postType| (third u)))))))
```

6.2.22 defplist postColonColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '||:| 'postTran|) 'postColonColon|))
```

6.2.23 defun postColonColon

```
[postForm p356]
[$boot p??]
```

— defun postColonColon —

```
(defun |postColonColon| (u)
  (if (and $boot (consp u) (eq (qfirst u) '||:|) (consp (qrest u))
    (consp (qcddr u)) (eq (qcdddr u) nil))
```

```
(intern (princ-to-string (third u)) (second u))
(|postForm| u)))
```

6.2.24 defplist postComma plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|,| '|postTran|) '|postComma|))
```

6.2.25 defun postComma

```
[postTuple p384]
[comma2Tuple p368]
```

— defun postComma —

```
(defun |postComma| (u)
  (|postTuple| (|comma2Tuple| u)))
```

6.2.26 defun comma2Tuple

```
[postFlatten p368]
```

— defun comma2Tuple —

```
(defun |comma2Tuple| (u)
  (cons '|@Tuple| (|postFlatten| u '|,|)))
```

6.2.27 defun postFlatten

```
[postFlatten p368]
```

— defun postFlatten —

```
(defun |postFlatten| (x op)
  (let (a b)
    (cond
      ((and (consp x) (equal (qfirst x) op) (consp (qrest x))
            (consp (qcddr x)) (eq (qcddr x) nil))
       (setq a (qsecond x))
       (setq b (qthird x))
       (append (|postFlatten| a op) (|postFlatten| b op)))
      (t (list x)))))
```

6.2.28 defplist postConstruct plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|construct| '|postTran|) '|postConstruct|))
```

6.2.29 defun postConstruct

```
[comma2Tuple p368]
[postTranSegment p370]
[postMakeCons p364]
[tuple2List p461]
[postTranList p354]
[postTran p352]
```

— defun postConstruct —

```
(defun |postConstruct| (u)
  (let (b a tmp4 tmp7)
    (cond
      ((and (consp u) (eq (qfirst u) '|construct|)
            (consp (qrest u)) (eq (qcddr u) nil))
       (setq b (qsecond u))
       (setq a
        (if (and (consp b) (eq (qfirst b) '|,|'))
            (|comma2Tuple| b)
            b))
       (cond
         ((and (consp a) (eq (qfirst a) 'segment) (consp (qrest a))
```



```

      (consp (qcddr a)) (eq (qcddr a) nil))
    (list '|construct| (|postTranSegment| (second a) (third a))))
  ((and (consp a) (eq (qfirst a) '|@Tuple|))
   (cond
    ((dolist (x (qrest a) tmp4)
     (setq tmp4
      (or tmp4
       (and (consp x) (eq (qfirst x) '|:|)
        (consp (qrest x)) (eq (qcddr x) nil))))))
     (|postMakeCons| (qrest a)))
    ((dolist (x (qrest a) tmp7)
     (setq tmp7 (or tmp7 (and (consp x) (eq (qfirst x) 'segment))))))
     (|tuple2List| (qrest a)))
    (t (cons '|construct| (|postTranList| (qrest a))))))
  (t (list '|construct| (|postTran| a))))
  (t u)))

```

6.2.30 defun postTranSegment

[postTran p352]

— defun postTranSegment —

```

(defun |postTranSegment| (p q)
  (list 'segment (|postTran| p) (when q (|postTran| q))))

```

6.2.31 defplist postDef plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|==| '|postTran|) '|postDef|))

```

6.2.32 defun postDef

[postMDef p377]

[recordHeaderDocumentation p??]

```
[nequal p??]
[postTran p352]
[postDefArgs p372]
[nreverse0 p??]
[$boot p??]
[$maxSignatureLineNumber p??]
[$headerDocumentation p??]
[$docList p??]
[$InteractiveMode p??]
```

— defun postDef —

```
(defun |postDef| (arg)
  (let (defOp rhs lhs targetType tmp1 op arg1 newLhs
        argTypeList typeList form specialCaseForm tmp4 tmp6 tmp8)
    (declare (special $boot |$maxSignatureLineNumber| |$headerDocumentation|
                      |$docList| |$InteractiveMode|))
    (setq defOp (first arg))
    (setq lhs (second arg))
    (setq rhs (third arg))
    (if (and (consp lhs) (eq (qfirst lhs) '|macro|)
            (consp (qrest lhs)) (eq (qcddr lhs) nil))
        (|postMDef| (list '==> (second lhs) rhs))
        (progn
          (unless $boot (|recordHeaderDocumentation| nil))
          (when (nequal |$maxSignatureLineNumber| 0)
            (setq |$docList|
                  (cons (cons '|constructor| |$headerDocumentation|) |$docList|))
            (setq |$maxSignatureLineNumber| 0))
          (setq lhs (|postTran| lhs))
          (setq tmp1
            (if (and (consp lhs) (eq (qfirst lhs) '|:|)) (cdr lhs) (list lhs nil)))
          (setq form (first tmp1))
          (setq targetType (second tmp1))
          (when (and (null |$InteractiveMode|) (atom form)) (setq form (list form)))
          (setq newLhs
            (if (atom form)
                form
                (progn
                  (setq tmp1
                    (dolist (x form (nreverse0 tmp4))
                      (push
                        (if (and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
                                (consp (qcddr x)) (eq (qcdddr x) nil))
                            (second x)
                            x)
                        tmp4)))
                  (setq op (car tmp1))
                  (setq arg1 (cdr tmp1))
```

```

      (cons op (|postDefArgs| arg1))))))
(setq argTypeList
  (unless (atom form)
    (dolist (x (cdr form) (nreverse0 tmp6))
      (push
        (when (and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
          (third x))
        tmp6))))
(setq typeList (cons targetType argTypeList))
(when (atom form) (setq form (list form)))
(setq specialCaseForm (dolist (x form (nreverse tmp8)) (push nil tmp8)))
(list 'def newLhs typeList specialCaseForm (|postTran| rhs))))))

```

6.2.33 defun postDefArgs

[postError p356]
 [postDefArgs p372]

— defun postDefArgs —

```

(defun |postDefArgs| (args)
  (let (a b)
    (cond
      ((null args) args)
      ((and (consp args) (consp (qfirst args)) (eq (qcaar args) '|:|)
        (consp (qcddr args)) (eq (qcdddr args) nil))
        (setq a (qcadar args))
        (setq b (qrest args))
        (cond
          (b (|postError|
            (list " Argument" a "of indefinite length must be last"))))
          ((or (atom a) (and (consp a) (eq (qfirst a) 'quote)))
            a)
          (t
            (|postError|
              (list " Argument" a "of indefinite length must be a name")))))
        (t (cons (car args) (|postDefArgs| (cdr args)))))))

```

6.2.34 defplist postExit plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|=>' '|postTran|) '|postExit|))
```

—————

6.2.35 defun postExit

[postTran p352]

— defun postExit —

```
(defun |postExit| (arg)
  (list 'if (|postTran| (second arg))
        (list '|exit| (|postTran| (third arg))
              '|noBranch|))
```

—————

6.2.36 defplist postIf plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|if|' '|postTran|) '|postIf|))
```

—————

6.2.37 defun postIf

```
[nreverse0 p??]
[postTran p352]
[$boot p??]
```

— defun postIf —

```
(defun |postIf| (arg)
```

```

(let (tmp1)
  (if (null (and (consp arg) (eq (qfirst arg) '|if|)))
      arg
      (cons 'if
        (dolist (x (qrest arg) (nreverse0 tmp1))
          (push
            (if (and (null (setq x (|postTran| x))) (null $boot)) '|noBranch| x)
            tmp1))))))

```

6.2.38 defplist postin plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|in| '|postTran|) '|postin|))

```

6.2.39 defun postin

```

[systemErrorHere p??]
[postTran p352]
[postInSeq p374]

```

— defun postin —

```

(defun |postin| (arg)
  (if (null (and (consp arg) (eq (qfirst arg) '|in|) (consp (qrest arg))
                (consp (qcddr arg)) (eq (qcdddr arg) nil)))
      (|systemErrorHere| "postin")
      (list '|in| (|postTran| (second arg)) (|postInSeq| (third arg)))))

```

6.2.40 defun postInSeq

```

[postTranSegment p370]
[tuple2List p461]
[postTran p352]

```

— defun postInSeq —

```
(defun |postInSeq| (seq)
  (cond
    ((and (consp seq) (eq (qfirst seq) 'segment) (consp (qrest seq))
      (consp (qcddr seq)) (eq (qcddr seq) nil))
      (|postTranSegment| (second seq) (third seq)))
    ((and (consp seq) (eq (qfirst seq) '@Tuple))
      (|tuple2List| (qrest seq)))
    (t (|postTran| seq))))
```

6.2.41 defplist postIn plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'in '|postTran|) '|postIn|))
```

6.2.42 defun postIn

```
[systemErrorHere p??]
[postTran p352]
[postInSeq p374]
```

— defun postIn —

```
(defun |postIn| (arg)
  (if (null (and (consp arg) (eq (qfirst arg) 'in) (consp (qrest arg))
    (consp (qcddr arg)) (eq (qcddr arg) nil)))
      (|systemErrorHere| "postIn")
      (list 'in (|postTran| (second arg)) (|postInSeq| (third arg)))))
```

6.2.43 defplist postJoin plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Join| '|postTran|) '|postJoin|))
```

6.2.44 defun postJoin

```
[postTran p352]
[postTranList p354]
```

— defun postJoin —

```
(defun |postJoin| (arg)
  (let (a l al)
    (setq a (|postTran| (cadr arg)))
    (setq l (|postTranList| (cddr arg)))
    (when (and (consp l) (eq (qrest l) nil) (consp (qfirst l))
              (member (qcaar l) '(attribute signature))))
      (setq l (list (list 'category (qfirst l)))))
    (setq al (if (and (consp a) (eq (qfirst a) '|@Tuple|)) (qrest a) (list a)))
    (cons '|Join| (append al l))))
```

6.2.45 defplist postMapping plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|->| '|postTran|) '|postMapping|))
```

6.2.46 defun postMapping

```
[postTran p352]
[unTuple p395]
```

— defun postMapping —

```
(defun |postMapping| (u)
  (if (null (and (consp u) (eq (qfirst u) '|->|)) (consp (qrest u))
```

```

      (cons (qcddr u) (eq (qcddr u) nil)))
u
(cons '|Mapping|
  (cons (|postTran| (third u))
    (|unTuple| (|postTran| (second u))))))

```

6.2.47 defplist postMDef plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|=>| '|postTran|) '|postMDef|))

```

6.2.48 defun postMDef

```

[postTran p352]
[throwkeyedmsg p??]
[nreverse0 p??]
[$InteractiveMode p??]
[$boot p??]

```

— defun postMDef —

```

(defun |postMDef| (arg)
  (let (rhs lhs tmp1 targetType form newLhs typeList tmp4 tmp5 tmp8)
    (declare (special |$InteractiveMode| $boot))
    (setq lhs (second arg))
    (setq rhs (third arg))
    (cond
      ((and |$InteractiveMode| (null $boot))
       (setq lhs (|postTran| lhs))
       (if (null (identp lhs))
         (|throwkeyedmsg| 's2ip0001 nil)
         (list 'mdef lhs nil nil (|postTran| rhs)))))
      (t
       (setq lhs (|postTran| lhs))
       (setq tmp1
         (if (and (cons (lhs) (eq (qfirst lhs) '|:|)) (cdr lhs) (list lhs nil)))
         (setq form (first tmp1))
         (setq targetType (second tmp1))

```



```

(setq form (if (atom form) (list form) form))
(setq newLhs
  (dolist (x form (nreverse0 tmp4))
    (push
      (if (and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))) (second x) x)
      tmp4)))
(setq typeList
  (cons targetType
    (dolist (x (qrest form) (nreverse0 tmp5))
      (push
        (when (and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
          (third x))
        tmp5))))
(list 'mdef newLhs typeList
  (dolist (x form (nreverse0 tmp8)) (push nil tmp8))
  (|postTran| rhs))))))

```

6.2.49 defplist postPretend plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|pretend| '|postTran|) '|postPretend|))

```

6.2.50 defun postPretend

```

[postTran p352]
[postType p361]

```

— defun postPretend —

```

(defun |postPretend| (arg)
  (cons '|pretend| (cons (|postTran| (second arg)) (|postType| (third arg)))))

```

6.2.51 defplist postQUOTE plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'quote '|postTran|) '|postQUOTE|))
```

6.2.52 defun postQUOTE

— defun postQUOTE —

```
(defun |postQUOTE| (arg) arg)
```

6.2.53 defplist postReduce plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Reduce| '|postTran|) '|postReduce|))
```

6.2.54 defun postReduce

```
[postTran p352]
[postReduce p379]
[$InteractiveMode p??]
```

— defun postReduce —

```
(defun |postReduce| (arg)
  (let (op expr g)
    (setq op (second arg))
    (setq expr (third arg))
    (if (or |$InteractiveMode| (and (consp expr) (eq (qfirst expr) 'collect)))
```

```
(list 'reduce op 0 (|postTran| expr))
(|postReduce|
 (list '|Reduce| op
  (list 'collect
   (list 'in (setq g (gensym)) expr)
   (list '|construct| g))))))
```

6.2.55 defplist postRepeat plist

— postvars —

```
(eval-when (eval load)
 (setf (get 'repeat '|postTran|) '|postRepeat|))
```

6.2.56 defun postRepeat

```
[postIteratorList p366]
[postTran p352]
```

— defun postRepeat —

```
(defun |postRepeat| (arg)
 (let (tmp1 x m)
  (setq tmp1 (reverse (cdr arg)))
  (setq x (car tmp1))
  (setq m (nreverse (cdr tmp1)))
  (cons 'repeat (append (|postIteratorList| m) (list (|postTran| x))))))
```

6.2.57 defplist postScripts plist

— postvars —

```
(eval-when (eval load)
 (setf (get '|Scripts| '|postTran|) '|postScripts|))
```

6.2.58 defun postScripts

[getScriptName p391]
 [postTranScripts p354]

— defun postScripts —

```
(defun |postScripts| (arg)
  (cons (|getScriptName| (second arg) (third arg) 0)
        (|postTranScripts| (third arg))))
```

—————

6.2.59 defplist postSemiColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|;| ' |postTran|) ' |postSemiColon|))
```

—————

6.2.60 defun postSemiColon

[postBlock p362]
 [postFlattenLeft p381]

— defun postSemiColon —

```
(defun |postSemiColon| (u)
  (|postBlock| (cons '|Block| (|postFlattenLeft| u '|;|))))
```

—————

6.2.61 defun postFlattenLeft

[postFlattenLeft p381]

— defun postFlattenLeft —

```
(defun |postFlattenLeft| (x op)
```

```
(let (a b)
  (cond
    ((and (consp x) (equal (qfirst x) op) (consp (qrest x))
          (consp (qcddr x)) (eq (qcddr x) nil))
     (setq a (qsecond x))
     (setq b (qthird x))
     (append (|postFlattenLeft| a op) (list b)))
    (t (list x)))))
```

6.2.62 defplist postSignature plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Signature| '|postTran|) '|postSignature|))
```

6.2.63 defun postSignature

```
[postType p361]
[removeSuperfluousMapping p383]
[killColons p383]
```

— defun postSignature —

```
(defun |postSignature| (arg)
  (let (sig sig1 op)
    (setq op (second arg))
    (setq sig (third arg))
    (when (and (consp sig) (eq (qfirst sig) '->))
      (setq sig1 (|postType| sig))
      (setq op (|postAtom| (if (stringp op) (setq op (intern op)) op)))
      (cons 'signature
            (cons op (|removeSuperfluousMapping| (|killColons| sig1)))))))
```

6.2.64 defun removeSuperfluousMapping

— defun removeSuperfluousMapping —

```
(defun |removeSuperfluousMapping| (sig1)
  (if (and (consp sig1) (consp (qfirst sig1)) (eq (qcaar sig1) '|Mapping|))
      (cons (cdr (qfirst sig1)) (qrest sig1))
      sig1))
```

6.2.65 defun killColons

[killColons p383]

— defun killColons —

```
(defun |killColons| (x)
  (cond
    ((atom x) x)
    ((and (consp x) (eq (qfirst x) '|Record|)) x)
    ((and (consp x) (eq (qfirst x) '|Union|)) x)
    ((and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
     (|killColons| (third x)))
    (t (cons (|killColons| (car x)) (|killColons| (cdr x))))))
```

6.2.66 defplist postSlash plist

— postvars —

```
(eval-when (eval load)
  (setf (get '/' '|postTran|) '|postSlash|))
```

6.2.67 defun postSlash

[postTran p352]

— defun postSlash —

```
(defun |postSlash| (arg)
  (if (stringp (second arg))
      (|postTran| (list '|Reduce| (intern (second arg)) (third arg) ))
      (list '|/| (|postTran| (second arg)) (|postTran| (third arg)))))
```

6.2.68 defplist postTuple plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|@Tuple| '|postTran|) '|postTuple|))
```

6.2.69 defun postTuple

[postTranList p354]

— defun postTuple —

```
(defun |postTuple| (arg)
  (cond
    ((and (consp arg) (eq (qrest arg) nil) (eq (qfirst arg) '|@Tuple|))
     arg)
    ((and (consp arg) (eq (qfirst arg) '|@Tuple|) (consp (qrest arg)))
     (cons '|@Tuple| (|postTranList| (cdr arg)))))
```

6.2.70 defplist postTupleCollect plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|TupleCollect| '|postTran|) '|postTupleCollect|))
```

6.2.71 defun postTupleCollect

[postCollect p365]

— defun postTupleCollect —

```
(defun |postTupleCollect| (arg)
  (let (constructOp tmp1 x m)
    (setq constructOp (car arg))
    (setq tmp1 (reverse (cdr arg)))
    (setq x (car tmp1))
    (setq m (nreverse (cdr tmp1)))
    (|postCollect| (cons constructOp (append m (list (list '|construct| x)))))))
```

6.2.72 defplist postWhere plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|where| '|postTran|) '|postWhere|))
```

6.2.73 defun postWhere

[postTran p352]

[postTranList p354]

— defun postWhere —

```
(defun |postWhere| (arg)
  (let (b x)
    (setq b (third arg))
    (setq x (if (and (consp b) (eq (qfirst b) '|Block|)) (qrest b) (list b)))
    (cons '|where| (cons (|postTran| (second arg)) (|postTranList| x)))))
```

6.2.74 defplist postWith plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|with| '|postTran|) '|postWith|))
```

—————

6.2.75 defun postWith

```
[postTran p352]
[$insidePostCategoryIfTrue p??]
```

— defun postWith —

```
(defun |postWith| (arg)
  (let (|$insidePostCategoryIfTrue| a)
    (declare (special |$insidePostCategoryIfTrue|))
    (setq |$insidePostCategoryIfTrue| t)
    (setq a (|postTran| (second arg)))
    (cond
      ((and (consp a) (member (qfirst a) '(signature attribute if)))
       (list 'category a))
      ((and (consp a) (eq (qfirst a) 'progn))
       (cons 'category (qrest a)))
      (t a))))
```

—————

6.3 Support routines

6.3.1 defun setDefOp

```
[$defOp p??]
[$topOp p??]
```

— defun setDefOp —

```
(defun |setDefOp| (f)
  (let (tmp1)
    (declare (special |$defOp| |$topOp|))
    (when (and (consp f) (eq (qfirst f) '|:|))
```

```

      (consp (setq tmp1 (qrest f))))
    (setq f (qfirst tmp1)))
  (unless (atom f) (setq f (car f)))
  (if |$topOp|
    (setq |$defOp| f)
    (setq |$topOp| f)))

```

6.3.2 defun aplTran

```

[aplTran1 p387]
[containsBang p390]
[$genno p??]
[$boot p??]

```

— defun aplTran —

```

(defun |aplTran| (x)
  (let ($genno u)
    (declare (special $genno $boot))
    (cond
      ($boot x)
      (t
       (setq $genno 0)
       (setq u (|aplTran1| x))
       (cond
         ((|containsBang| u) (|throwKeyedMsg| 's2ip0002 nil))
         (t u))))))

```

6.3.3 defun aplTran1

```

[aplTranList p389]
[aplTran1 p387]
[hasAplExtension p389]
[nreverse0 p??]
[ p??]
[$boot p??]

```

— defun aplTran1 —

```

(defun |aplTran1| (x)

```

```

(let (op argl1 argl f y opprime yprime tmp1 arglAssoc futureArgl g)
(declare (special $boot))
(if (atom x)
  x
  (progn
    (setq op (car x))
    (setq argl1 (cdr x))
    (setq argl (|aplTranList| argl1))
    (cond
      ((eq op '!))
      (cond
        ((and (consp argl)
              (progn
                (setq f (qfirst argl))
                (setq tmp1 (qrest argl))
                (and (consp tmp1)
                     (eq (qrest tmp1) nil)
                     (progn
                      (setq y (qfirst tmp1))
                      t))))))
        (cond
          ((and (consp y)
                (progn
                  (setq opprime (qfirst y))
                  (setq yprime (qrest y))
                  t)
                (eq opprime '!)))
          (|aplTran1| (cons op (cons op (cons f yprime))))))
      ($boot
       (cons 'collect
             (cons
              (list 'in (setq g (genvar)) (|aplTran1| y))
              (list (list f g) ))))
      (t
       (list 'map f (|aplTran1| y) ))))
    (t x)))
((progn
  (setq tmp1 (|hasAplExtension| argl))
  (and (consp tmp1)
       (progn
        (setq arglAssoc (qfirst tmp1))
        (setq futureArgl (qrest tmp1))
        t)))
 (cons '|reshape|
       (cons
        (cons 'collect
              (append
               (do ((tmp3 arglAssoc (cdr tmp3)) (tmp4 nil))
                   ((or (atom tmp3)
                        (progn (setq tmp4 (car tmp3)) nil)
                        nil)

```

```

      (progn
        (setq g (car tmp4))
        (setq a (cdr tmp4))
        nil))
      (nreverse0 tmp2))
    (push (list 'in g (list '|ravel| a))) tmp2))
  (list (|aplTran1| (cons op futureArg1))))
  (list (cdar arglAssoc)))
  (t (cons op argl))))))

```

6.3.4 defun aplTranList

[aplTran1 p387]
[aplTranList p389]

— defun aplTranList —

```

(defun |aplTranList| (x)
  (if (atom x)
      x
      (cons (|aplTran1| (car x)) (|aplTranList| (cdr x)))))

```

6.3.5 defun hasAplExtension

[nreverse0 p??]
[deepestExpression p390]
[genvar p??]
[aplTran1 p387]
[msubst p??]

— defun hasAplExtension —

```

(defun |hasAplExtension| (arg1)
  (let (tmp2 tmp3 y z g arglAssoc u)
    (when
      (dolist (x arg1 tmp2)
        (setq tmp2 (or tmp2 (and (consp x) (eq (qfirst x) '|')))))
      (setq u
        (dolist (x arg1 (nreverse0 tmp3))
          (push
            (if (and (consp x) (eq (qfirst x) '|')

```

```

      (consp (qrest x)) (eq (qcddr x) nil))
    (progn
      (setq y (qsecond x))
      (setq z (|deepestExpression| y))
      (setq arglAssoc
        (cons (cons (setq g (genvar)) (|aplTran1| z)) arglAssoc))
      (msubst g z y))
    x)
  tmp3)))
(cons arglAssoc u))))

```

6.3.6 defun deepestExpression

[deepestExpression p390]

— defun deepestExpression —

```

(defun |deepestExpression| (x)
  (if (and (consp x) (eq (qfirst x) '!))
      (consp (qrest x)) (eq (qcddr x) nil))
      (|deepestExpression| (qsecond x))
      x))

```

6.3.7 defun containsBang

[containsBang p390]

— defun containsBang —

```

(defun |containsBang| (u)
  (let (tmp2)
    (cond
      ((atom u) (eq u '!))
      ((and (consp u) (equal (qfirst u) 'quote)
        (consp (qrest u)) (eq (qcddr u) nil))
        nil)
      (t
        (dolist (x u tmp2)
          (setq tmp2 (or tmp2 (|containsBang| x)))))))

```

6.3.8 defun getScriptName

```
[getScriptName identp (vol5)]
[postError p356]
[internal p??]
[decodeScripts p391]
[getScriptName pname (vol5)]
```

— **defun getScriptName** —

```
(defun |getScriptName| (op a numberOfFunctionalArgs)
  (when (null (identp op))
    (|postError| (list " " op " cannot have scripts" )))
  (internal '* (princ-to-string numberOfFunctionalArgs)
    (|decodeScripts| a) (pname op)))
```

6.3.9 defun decodeScripts

```
[qcar p??]
[qcdr p??]
[strconc p??]
[decodeScripts p391]
```

— **defun decodeScripts** —

```
(defun |decodeScripts| (a)
  (labels (
    (fn (a)
      (let ((tmp1 0))
        (if (and (consp a) (eq (qfirst a) '|,|))
          (dolist (x (qrest a) tmp1) (setq tmp1 (+ tmp1 (fn x))))
          1))))
    (cond
      ((and (consp a) (eq (qfirst a) '|PrefixSC|)
        (consp (qrest a)) (eq (qcddr a) nil))
       (strconc (princ-to-string 0) (|decodeScripts| (qsecond a))))
      ((and (consp a) (eq (qfirst a) '|;|))
       (apply 'strconc (loop for x in (qrest a) collect (|decodeScripts| x))))
      ((and (consp a) (eq (qfirst a) '|,|))
       (princ-to-string (fn a)))
      (t
       (princ-to-string 1)))))
```

Chapter 7

DEF forms

7.0.10 defvar \$defstack

— initvars —

```
(defvar $defstack nil)
```

—————

7.0.11 defvar \$is-spill

— initvars —

```
(defvar $is-spill nil)
```

—————

7.0.12 defvar \$is-spill-list

— initvars —

```
(defvar $is-spill-list nil)
```

—————

7.0.13 defvar \$vl

— initvars —

```
(defvar $vl nil)
```

—————

7.0.14 defvar \$is-gensymlist

— initvars —

```
(defvar $is-gensymlist nil)
```

—————

7.0.15 defvar \$initial-gensym

— initvars —

```
(defvar initial-gensym (list (gensym)))
```

—————

7.0.16 defvar \$is-eqlist

— initvars —

```
(defvar $is-eqlist nil)
```

—————

7.0.17 defun hackforis

[hackforis1 p395]

— defun hackforis —

```
(defun hackforis (l) (mapcar #'hackforis1 L))
```

7.0.18 defun hackforis1

```
[kar p??]  
[eqcar p??]
```

— defun hackforis1 —

```
(defun hackforis1 (x)  
  (if (and (member (kar x) '(in on)) (eqcar (second x) 'is))  
      (cons (first x) (cons (cons 'spadlet (cdadr x)) (cddr x)))  
      x))
```

7.0.19 defun unTuple

— defun unTuple —

```
(defun |unTuple| (x)  
  (if (and (consp x) (eq (qfirst x) '|@Tuple|))  
      (qrest x)  
      (list x)))
```

7.0.20 defun errhuh

```
[systemError p??]
```

— defun errhuh —

```
(defun errhuh ()  
  (|systemError| "problem with BOOT to LISP translation"))
```

Chapter 8

PARSE forms

8.1 The original meta specification

This package provides routines to support the Metalanguage translator writing system. Metalanguage is described in META/LISP, R.D. Jenks, Tech Report, IBM T.J. Watson Research Center, 1969. Familiarity with this document is assumed.

Note that META/LISP and the meta parser/generator were removed from Axiom. This information is only for documentation purposes.

```
%      Scratchpad II Boot Language Grammar, Common Lisp Version
%      IBM Thomas J. Watson Research Center
%      Summer, 1986
%
%      NOTE: Substantially different from VM/LISP version, due to
%            different parser and attempt to render more within META proper.

.META(New NewExpr Process)
.PACKAGE 'BOOT'
.DECLARE(tmpTok TOK ParseMode DEFINITION-NAME LABLASOC)
.PREFIX 'PARSE-'

NewExpr:      '=')' .(processSynonyms) Command
              / .(SETQ DEFINITION-NAME (CURRENT-SYMBOL)) Statement ;

Command:      ')' SpecialKeyWord SpecialCommand +() ;

SpecialKeyWord: =(MATCH-CURRENT-TOKEN "IDENTIFIER)
                .(SETF (TOKEN-SYMBOL (CURRENT-TOKEN)) (unAbbreviateKeyword (CURRENT-SYMBOL))) ;

SpecialCommand: 'show' '<??' / Expression>! +(show #1) CommandTail
                / ?(MEMBER (CURRENT-SYMBOL) \noParseCommands)
                .(FUNCALL (CURRENT-SYMBOL))
```

```

/ ?(MEMBER (CURRENT-SYMBOL) \ $tokenCommands) TokenList
TokenCommandTail
/ PrimaryOrQM* CommandTail ;

TokenList:      (^?(isTokenDelimiter) +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN))* ;

TokenCommandTail:
    <TokenOption*>! ?(atEndOfLine) +(#2 -#1) .(systemCommand #1) ;

TokenOption:    '))' TokenList ;

CommandTail:    <Option*>! ?(atEndOfLine) +(#2 -#1) .(systemCommand #1) ;

PrimaryOrQM:    '?' +\? / Primary ;

Option:         '))' PrimaryOrQM* ;

Statement:      Expr{0} <(',' Expr{0})* +(Series #2 -#1)>;

InfixWith:      With +(Join #2 #1) ;

With:          'with' Category +(with #1) ;

Category:       'if' Expression 'then' Category <'else' Category>! +(if #3 #2 #1)
/ '(' Category <('; ' Category)*>! '))' +(CATEGORY #2 -#1)
/ .(SETQ $1 (LINE-NUMBER CURRENT-LINE)) Application
    ( ':' Expression +(Signature #2 #1)
      .(recordSignatureDocumentation ##1 $1)
      / +(Attribute #1)
      .(recordAttributeDocumentation ##1 $1));

Expression:     Expr{(PARSE-rightBindingPowerOf (MAKE-SYMBOL-OF PRIOR-TOKEN) ParseMode)}
    +#1 ;

Import:         'import' Expr{1000} <(',' Expr{1000})*>! +(import #2 -#1) ;

Infix:         =TRUE +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) <TokTail>
Expression +(#2 #2 #1) ;

Prefix:        =TRUE +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) <TokTail>
Expression +(#2 #1) ;

Suffix:        +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) <TokTail> +( #1 #1) ;

TokTail:       ?(AND (NULL \ $BOOT) (EQ (CURRENT-SYMBOL) "\$")
    (OR (ALPHA-CHAR-P (CURRENT-CHAR))
        (CHAR-EQ (CURRENT-CHAR) '$')
        (CHAR-EQ (CURRENT-CHAR) '%')
        (CHAR-EQ (CURRENT-CHAR) '(')))
    .(SETQ $1 (COPY-TOKEN PRIOR-TOKEN)) Qualification

```

```

.(SETQ PRIOR-TOKEN $1) ;

Qualification: '$' Primary1 +=(dollarTran #1 #1) ;

SemiColon: ';' (Expr{82} / + \ /throwAway) +(\; #2 #1) ;

Return: 'return' Expression +(return #1) ;

Exit: 'exit' (Expression / +\ $NoValue) +(exit #1) ;

Leave: 'leave' ( Expression / +\ $NoValue )
      ('from' Label +(leaveFrom #1 #1) / +(leave #1)) ;

Seg: GlyphTok{"\.\.} <Expression>! +(SEGMENT #2 #1) ;

Conditional: 'if' Expression 'then' Expression <'else' ElseClause>!
              +(if #3 #2 #1) ;

ElseClause: ?(EQ (CURRENT-SYMBOL) "if) Conditional / Expression ;

Loop: Iterator* 'repeat' Expr{110} +(REPEAT -#2 #1)
      / 'repeat' Expr{110} +(REPEAT #1) ;

Iterator: 'for' Primary 'in' Expression
          ( 'by' Expr{200} +(INBY #3 #2 #1) / +(IN #2 #1) )
          < '\|' Expr{111} +(\| #1) >
          / 'while' Expr{190} +(WHILE #1)
          / 'until' Expr{190} +(UNTIL #1) ;

Expr{RBP}: NudPart{RBP} <LedPart{RBP}>* +#1;

LabelExpr: Label Expr{120} +(LABEL #2 #1) ;

Label: '@<<' Name '>>' ;

LedPart{RBP}: Operation{"Led RBP} +#1;

NudPart{RBP}: (Operation{"Nud RBP} / Reduction / Form) +#1 ;

Operation{ParseMode RBP}:
  ^?(MATCH-CURRENT-TOKEN "IDENTIFIER)
  ?(GETL (SETQ tmptok (CURRENT-SYMBOL)) ParseMode)
  ?(LT RBP (PARSE-leftBindingPowerOf tmptok ParseMode))
  .(SETQ RBP (PARSE-rightBindingPowerOf tmptok ParseMode))
  getSemanticForm{tmptok ParseMode (ELEMN (GETL tmptok ParseMode) 5 NIL)} ;

% Binding powers stored under the Led and Red properties of an operator
% are set up by the file BOTTOMUP.LISP. The format for a Led property
% is <Operator Left-Power Right-Power>, and the same for a Nud, except that
% it may also have a fourth component <Special-Handler>. ELEMN attempts to

```

```

% get the Nth indicator, counting from 1.

leftBindingPowerOf{X IND}: =(LET ((Y (GETL X IND))) (IF Y (ELEMN Y 3 0) 0)) ;

rightBindingPowerOf{X IND}: =(LET ((Y (GETL X IND))) (IF Y (ELEMN Y 4 105) 105)) ;

getSemanticForm{X IND Y}:
    ?(AND Y (EVAL Y)) / ?(EQ IND "Nud) Prefix / ?(EQ IND "Led) Infix ;

Reduction:      ReductionOp Expr{1000} +(Reduce #2 #1) ;

ReductionOp:    ?(AND (GETL (CURRENT-SYMBOL) "Led)
                    (MATCH-NEXT-TOKEN "SPECIAL-CHAR (CODE-CHAR 47))) % Forgive me!
    +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) .(ADVANCE-TOKEN) ;

Form:           'iterate' < 'from' Label +(#1) >! +(iterate -#1)
                / 'yield' Application +(yield #1)
                / Application ;

Application:    Primary <Selector>* <Application +(#2 #1)>;

Selector:      ?NONBLANK ?(EQ (CURRENT-SYMBOL) "\.") ?(CHAR-NE (CURRENT-CHAR) "\ ")
                ' .' PrimaryNoFloat (=$BOOT +(ELT #2 #1)/ +( #2 #1))
                / (Float /' .' Primary) (=$BOOT +(ELT #2 #1)/ +( #2 #1));

PrimaryNoFloat: Primary1 <TokTail> ;

Primary:       Float /PrimaryNoFloat ;

Primary1:      VarForm <=(AND NONBLANK (EQ (CURRENT-SYMBOL) "\()") Primary1 +(#2 #1)>
                /Quad
                /String
                /IntegerTok
                /FormalParameter
                /='\' ( ?$BOOT Data / '\'' Expr{999} +(QUOTE #1))
                /Sequence
                /Enclosure ;

Float:         FloatBase (?NONBLANK FloatExponent / +0) +=(MAKE-FLOAT #4 #2 #2 #1) ;

FloatBase:     ?(FIXP (CURRENT-SYMBOL)) ?(CHAR-EQ (CURRENT-CHAR) ' .')
                ?(CHAR-NE (NEXT-CHAR) ' .')
                IntegerTok FloatBasePart
                /?(FIXP (CURRENT-SYMBOL)) ?(CHAR-EQ (CHAR-UPCASE (CURRENT-CHAR)) "E")
                IntegerTok +0 +0
                /?(DIGITP (CURRENT-CHAR)) ?(EQ (CURRENT-SYMBOL) "\.")
                +0 FloatBasePart ;

FloatBasePart: ' .'

```

```

(? (DIGITP (CURRENT-CHAR)) += (TOKEN-NONBLANK (CURRENT-TOKEN)) IntegerTok
/ +0 +0);

FloatExponent: =(AND (MEMBER (CURRENT-SYMBOL) "(E e))
(FIND (CURRENT-CHAR) '+-'))
.(ADVANCE-TOKEN)
(IntegerTok/'+' IntegerTok/'-' IntegerTok +=(MINUS #1)/+0)
/?(IDENTP (CURRENT-SYMBOL)) =(SETQ $1 (FLOATEXPID (CURRENT-SYMBOL)))
.(ADVANCE-TOKEN) +=$1 ;

Enclosure:      '(' ( Expr{6} ')' / ')' + (Tuple) )
/ '{' ( Expr{6} '}' + (brace (construct #1)) / '}' + (brace)) ;

IntegerTok:      NUMBER ;

FloatTok:        NUMBER +=(IF \ $BOOT #1 (BFP- #1)) ;

FormalParameter: FormalParameterTok ;

FormalParameterTok: ARGUMENT-DESIGNATOR ;

Quad:           '$' +\$ / ?\$BOOT GlyphTok{"\."} +\. ;

String:          SPADSTRING ;

VarForm:        Name <Scripts +(Scripts #2 #1) > + #1 ;

Scripts:         ?NONBLANK '[' ScriptItem ']' ;

ScriptItem:      Expr{90} <(';' ScriptItem)* +(\; #2 -#1)>
/ ';' ScriptItem +(PrefixSC #1) ;

Name:           IDENTIFIER + #1 ;

Data:           .(SETQ LABLASOC NIL) Sexpr +(QUOTE =(TRANSLABEL #1 LABLASOC)) ;

Sexpr:          .(ADVANCE-TOKEN) Sexpr1 ;

Sexpr1:         AnyId
< NBGlyphTok{"\="} Sexpr1
.(SETQ LABLASOC (CONS (CONS #2 ##1) LABLASOC))>
/ '\'' Sexpr1 +(QUOTE #1)
/ IntegerTok
/ '-' IntegerTok +=(MINUS #1)
/ String
/ '<' <Sexpr1*>!'>' +=(LIST2VEC #1)
/ '(' <Sexpr1* <GlyphTok{"\."} Sexpr1 +=(NCONC #2 #1)>>!'>' ;

NBGlyphTok{tok}: ?(AND (MATCH-CURRENT-TOKEN "GLIPH tok) NONBLANK)

```



```

.(ADVANCE-TOKEN) ;

GliphTok{tok}:    ?(MATCH-CURRENT-TOKEN "GLIPH tok) .(ADVANCE-TOKEN) ;

AnyId:           IDENTIFIER
                 / (='$$' +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) / KEYWORD) ;

Sequence:        OpenBracket Sequence1 ']'
                 / OpenBrace Sequence1 '}' +(brace #1) ;

Sequence1:       (Expression +(##2 #1) / +(##1)) <IteratorTail +(COLLECT -#1 #1)> ;

OpenBracket:     =(EQ (getToken (SETQ $1 (CURRENT-SYMBOL))) "\[ )
                 (= (EQCAR $1 "elt) +(elt =(CADR $1) construct)
                 / +construct) .(ADVANCE-TOKEN) ;

OpenBrace:       =(EQ (getToken (SETQ $1 (CURRENT-SYMBOL))) "\{ )
                 (= (EQCAR $1 "elt) +(elt =(CADR $1) brace)
                 / +construct) .(ADVANCE-TOKEN) ;

IteratorTail:    ('repeat' <Iterator*>! / Iterator*) ;

.FIN ;

```

8.2 The PARSE code

8.2.1 defvar \$tmptok

```

— initvars —

(defvar |tmptok| nil)

```

8.2.2 defvar \$tok

```

— initvars —

(defvar tok nil)

```

8.2.3 defvar \$ParseMode— **initvars** —

```
(defvar |ParseMode| nil)
```

—————

8.2.4 defvar \$definition-name— **initvars** —

```
(defvar definition-name nil)
```

—————

8.2.5 defvar \$lablasoc— **initvars** —

```
(defvar lablasoc nil)
```

—————

8.2.6 defun PARSE-NewExpr

```
[match-string p440]
[action p453]
[PARSE-NewExpr processSynonyms (vol5)]
[must p452]
[current-symbol p446]
[PARSE-Statement p408]
[definition-name p403]
```

— **defun PARSE-NewExpr** —

```
(defun |PARSE-NewExpr| ()
  (or (and (match-string "") (action (|processSynonyms|))
          (must (|PARSE-Command|))))
```

```
(and (action (setq definition-name (current-symbol)))
      (|PARSE-Statement|)))
```

8.2.7 defun PARSE-Command

```
[match-advance-string p441]
[must p452]
[PARSE-SpecialKeyWord p404]
[PARSE-SpecialCommand p405]
[push-reduction p454]
```

— defun PARSE-Command —

```
(defun |PARSE-Command| ()
  (and (match-advance-string "") (must (|PARSE-SpecialKeyWord|))
        (must (|PARSE-SpecialCommand|))
        (push-reduction '|PARSE-Command| nil)))
```

8.2.8 defun PARSE-SpecialKeyWord

```
[match-current-token p445]
[action p453]
[token-symbol p??]
[current-token p447]
[PARSE-SpecialKeyWord unAbbreviateKeyword (vol5)]
[current-symbol p446]
```

— defun PARSE-SpecialKeyWord —

```
(defun |PARSE-SpecialKeyWord| ()
  (and (match-current-token 'identifier)
        (action (setf (token-symbol (current-token))
                      (|unAbbreviateKeyword| (current-symbol))))))
```

8.2.9 defun PARSE-SpecialCommand

[match-advance-string p441]
 [bang p??]
 [optional p453]
 [PARSE-Expression p411]
 [push-reduction p454]
 [PARSE-SpecialCommand p405]
 [pop-stack-1 p462]
 [PARSE-CommandTail p407]
 [must p452]
 [current-symbol p446]
 [action p453]
 [PARSE-TokenList p406]
 [PARSE-TokenCommandTail p405]
 [star p453]
 [PARSE-PrimaryOrQM p407]
 [PARSE-CommandTail p407]
 [\$noParseCommands p??]
 [\$tokenCommands p??]

— defun PARSE-SpecialCommand —

```
(defun |PARSE-SpecialCommand| ()
  (declare (special |$noParseCommands| |$tokenCommands|))
  (or (and (match-advance-string "show")
    (bang fil_test
      (optional
        (or (match-advance-string "?")
          (|PARSE-Expression|))))
    (push-reduction '|PARSE-SpecialCommand|
      (list '|show| (pop-stack-1)))
    (must (|PARSE-CommandTail|)))
    (and (member (current-symbol) |$noParseCommands|)
      (action (funcall (current-symbol))))
    (and (member (current-symbol) |$tokenCommands|)
      (|PARSE-TokenList|) (must (|PARSE-TokenCommandTail|)))
    (and (star repeater (|PARSE-PrimaryOrQM|))
      (must (|PARSE-CommandTail|))))))
```

8.2.10 defun PARSE-TokenCommandTail

[bang p??]
 [optional p453]

```

[star p453]
[PARSE-TokenOption p406]
[atEndOfLine p??]
[push-reduction p454]
[PARSE-TokenCommandTail p405]
[pop-stack-2 p463]
[pop-stack-1 p462]
[action p453]
[PARSE-TokenCommandTail systemCommand (vol5)]

```

— **defun PARSE-TokenCommandTail** —

```

(defun |PARSE-TokenCommandTail| ()
  (and (bang fil_test (optional (star repeater (|PARSE-TokenOption|))))
    (|atEndOfLine|)
    (push-reduction ' |PARSE-TokenCommandTail|
      (cons (pop-stack-2) (append (pop-stack-1) nil)))
    (action (|systemCommand| (pop-stack-1)))))

```

—————

8.2.11 defun PARSE-TokenOption

```

[match-advance-string p441]
[must p452]
[PARSE-TokenList p406]

```

— **defun PARSE-TokenOption** —

```

(defun |PARSE-TokenOption| ()
  (and (match-advance-string "") (must (|PARSE-TokenList|))))

```

—————

8.2.12 defun PARSE-TokenList

```

[star p453]
[isTokenDelimiter p443]
[push-reduction p454]
[current-symbol p446]
[action p453]
[advance-token p448]

```

— **defun PARSE-TokenList** —

```
(defun |PARSE-TokenList| ()
  (star repeater
    (and (not (|isTokenDelimiter|))
      (push-reduction '|PARSE-TokenList| (current-symbol))
      (action (advance-token))))))
```

8.2.13 defun PARSE-CommandTail

```
[bang p??]
[optional p453]
[star p453]
[push-reduction p454]
[PARSE-Option p408]
[PARSE-CommandTail p407]
[pop-stack-2 p463]
[pop-stack-1 p462]
[action p453]
[PARSE-CommandTail systemCommand (vol5)]
```

— defun PARSE-CommandTail —

```
(defun |PARSE-CommandTail| ()
  (and (bang fil_test (optional (star repeater (|PARSE-Option|))))
    (|atEndOfLine|)
    (push-reduction '|PARSE-CommandTail|
      (cons (pop-stack-2) (append (pop-stack-1) nil)))
    (action (|systemCommand| (pop-stack-1)))))
```

8.2.14 defun PARSE-PrimaryOrQM

```
[match-advance-string p441]
[push-reduction p454]
[PARSE-PrimaryOrQM p407]
[PARSE-Primary p420]
```

— defun PARSE-PrimaryOrQM —

```
(defun |PARSE-PrimaryOrQM| ()
  (or (and (match-advance-string "?")
    (push-reduction '|PARSE-PrimaryOrQM| '??))
```

```
(|PARSE-Primary|))
```

8.2.15 defun PARSE-Option

```
[match-advance-string p441]
[must p452]
[star p453]
[PARSE-PrimaryOrQM p407]
```

— defun PARSE-Option —

```
(defun |PARSE-Option| ()
  (and (match-advance-string "")
        (must (star repeater (|PARSE-PrimaryOrQM|))))))
```

8.2.16 defun PARSE-Statement

```
[PARSE-Expr p412]
[optional p453]
[star p453]
[match-advance-string p441]
[must p452]
[push-reduction p454]
[pop-stack-2 p463]
[pop-stack-1 p462]
```

— defun PARSE-Statement —

```
(defun |PARSE-Statement| ()
  (and (|PARSE-Expr| 0)
        (optional
          (and (star repeater
                    (and (match-advance-string ",")
                        (must (|PARSE-Expr| 0))))
                (push-reduction '|PARSE-Statement|
                                (cons '|Series|
                                      (cons (pop-stack-2)
                                            (append (pop-stack-1) nil))))))))))
```

8.2.17 defun PARSE-InfixWith

[PARSE-With p409]
 [push-reduction p454]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— defun PARSE-InfixWith —

```
(defun |PARSE-InfixWith| ()
  (and (|PARSE-With|)
        (push-reduction '|PARSE-InfixWith|
                          (list '|Join| (pop-stack-2) (pop-stack-1))))))
```

—————

8.2.18 defun PARSE-With

[match-advance-string p441]
 [must p452]
 [push-reduction p454]
 [pop-stack-1 p462]

— defun PARSE-With —

```
(defun |PARSE-With| ()
  (and (match-advance-string "with") (must (|PARSE-Category|))
        (push-reduction '|PARSE-With|
                          (cons '|with| (cons (pop-stack-1) nil))))))
```

—————

8.2.19 defun PARSE-Category

[match-advance-string p441]
 [must p452]
 [bang p??]
 [optional p453]
 [push-reduction p454]
 [PARSE-Expression p411]
 [PARSE-Category p409]
 [pop-stack-3 p463]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

8.2.20 defun PARSE-Expression

[PARSE-Expr p412]
 [PARSE-rightBindingPowerOf p414]
 [make-symbol-of p446]
 [push-reduction p454]
 [pop-stack-1 p462]
 [ParseMode p403]
 [prior-token p90]

— **defun PARSE-Expression** —

```
(defun |PARSE-Expression| ()
  (declare (special prior-token))
  (and (|PARSE-Expr|
        (|PARSE-rightBindingPowerOf| (make-symbol-of prior-token)
                                       |ParseMode|))
        (push-reduction '|PARSE-Expression| (pop-stack-1))))
```

8.2.21 defun PARSE-Import

[match-advance-string p441]
 [must p452]
 [PARSE-Expr p412]
 [bang p??]
 [optional p453]
 [star p453]
 [push-reduction p454]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— **defun PARSE-Import** —

```
(defun |PARSE-Import| ()
  (and (match-advance-string "import") (must (|PARSE-Expr| 1000))
        (bang fil_test
          (optional
            (star repeater
              (and (match-advance-string ",")
                    (must (|PARSE-Expr| 1000)))))))
        (push-reduction '|PARSE-Import|
          (cons '|import|
            (cons (pop-stack-2) (append (pop-stack-1) nil))))))
```

8.2.22 defun PARSE-Expr

[PARSE-NudPart p412]
 [PARSE-LedPart p412]
 [optional p453]
 [star p453]
 [push-reduction p454]
 [pop-stack-1 p462]

— defun PARSE-Expr —

```
(defun |PARSE-Expr| (rbp)
  (declare (special rbp))
  (and (|PARSE-NudPart| rbp)
    (optional (star opt_expr (|PARSE-LedPart| rbp)))
    (push-reduction '|PARSE-Expr| (pop-stack-1))))
```

8.2.23 defun PARSE-LedPart

[PARSE-Operation p413]
 [push-reduction p454]
 [pop-stack-1 p462]

— defun PARSE-LedPart —

```
(defun |PARSE-LedPart| (rbp)
  (declare (special rbp))
  (and (|PARSE-Operation| '|Led| rbp)
    (push-reduction '|PARSE-LedPart| (pop-stack-1))))
```

8.2.24 defun PARSE-NudPart

[PARSE-Operation p413]
 [PARSE-Reduction p417]
 [PARSE-Form p417]
 [push-reduction p454]
 [pop-stack-1 p462]

[rbp p??]

— defun PARSE-NudPart —

```
(defun |PARSE-NudPart| (rbp)
  (declare (special rbp))
  (and (or (|PARSE-Operation| ' |Nud| rbp) (|PARSE-Reduction|)
    (|PARSE-Form|))
    (push-reduction ' |PARSE-NudPart| (pop-stack-1))))
```

—————

8.2.25 defun PARSE-Operation

[match-current-token p445]
 [current-symbol p446]
 [PARSE-leftBindingPowerOf p413]
 [lt p??]
 [getl p??]
 [action p453]
 [PARSE-rightBindingPowerOf p414]
 [PARSE-getSemanticForm p414]
 [elemn p??]
 [ParseMode p403]
 [rbp p??]
 [tmptok p402]

— defun PARSE-Operation —

```
(defun |PARSE-Operation| (|ParseMode| rbp)
  (declare (special |ParseMode| rbp |tmptok|))
  (and (not (match-current-token 'identifier))
    (getl (setq |tmptok| (current-symbol)) |ParseMode|)
    (lt rbp (|PARSE-leftBindingPowerOf| |tmptok| |ParseMode|))
    (action (setq rbp (|PARSE-rightBindingPowerOf| |tmptok| |ParseMode|))
      (|PARSE-getSemanticForm| |tmptok| |ParseMode|
        (elemn (getl |tmptok| |ParseMode|) 5 nil))))
```

—————

8.2.26 defun PARSE-leftBindingPowerOf

[getl p??]
 [elemn p??]

— **defun PARSE-leftBindingPowerOf** —

```
(defun |PARSE-leftBindingPowerOf| (x ind)
  (declare (special x ind))
  (let ((y (get1 x ind))) (if y (elemn y 3 0) 0)))
```

8.2.27 **defun PARSE-rightBindingPowerOf**

[get1 p??]
[elemn p??]

— **defun PARSE-rightBindingPowerOf** —

```
(defun |PARSE-rightBindingPowerOf| (x ind)
  (declare (special x ind))
  (let ((y (get1 x ind))) (if y (elemn y 4 105) 105)))
```

8.2.28 **defun PARSE-getSemanticForm**

[PARSE-Prefix p414]
[PARSE-Infix p415]

— **defun PARSE-getSemanticForm** —

```
(defun |PARSE-getSemanticForm| (x ind y)
  (declare (special x ind y))
  (or (and y (eval y)) (and (eq ind '|Nud|) (|PARSE-Prefix|))
      (and (eq ind '|Led|) (|PARSE-Infix|))))
```

8.2.29 **defun PARSE-Prefix**

[push-reduction p454]
[current-symbol p446]
[action p453]
[advance-token p448]

[optional p453]
 [PARSE-TokTail p416]
 [must p452]
 [PARSE-Expression p411]
 [push-reduction p454]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— **defun PARSE-Prefix** —

```
(defun |PARSE-Prefix| ()
  (and (push-reduction '|PARSE-Prefix| (current-symbol))
        (action (advance-token)) (optional (|PARSE-TokTail|))
        (must (|PARSE-Expression|))
        (push-reduction '|PARSE-Prefix|
                          (list (pop-stack-2) (pop-stack-1))))))
```

8.2.30 defun PARSE-Infix

[push-reduction p454]
 [current-symbol p446]
 [action p453]
 [advance-token p448]
 [optional p453]
 [PARSE-TokTail p416]
 [must p452]
 [PARSE-Expression p411]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— **defun PARSE-Infix** —

```
(defun |PARSE-Infix| ()
  (and (push-reduction '|PARSE-Infix| (current-symbol))
        (action (advance-token)) (optional (|PARSE-TokTail|))
        (must (|PARSE-Expression|))
        (push-reduction '|PARSE-Infix|
                          (list (pop-stack-2) (pop-stack-2) (pop-stack-1) )))))
```

8.2.31 defun PARSE-TokTail

[current-symbol p446]
 [current-char p449]
 [char-eq p450]
 [copy-token p??]
 [action p453]
 [PARSE-Qualification p416]
 [\$boot p??]

— defun PARSE-TokTail —

```
(defun |PARSE-TokTail| ()
  (let (g1)
    (and (null $boot) (eq (current-symbol) '$)
      (or (alpha-char-p (current-char))
        (char-eq (current-char) "$")
        (char-eq (current-char) "%")
        (char-eq (current-char) "("))
      (action (setq g1 (copy-token prior-token)))
      (|PARSE-Qualification|) (action (setq prior-token g1)))))
```

—————

8.2.32 defun PARSE-Qualification

[match-advance-string p441]
 [must p452]
 [PARSE-Primary1 p420]
 [push-reduction p454]
 [dollarTran p451]
 [pop-stack-1 p462]

— defun PARSE-Qualification —

```
(defun |PARSE-Qualification| ()
  (and (match-advance-string "$") (must (|PARSE-Primary1|))
    (push-reduction '|PARSE-Qualification|
      (|dollarTran| (pop-stack-1) (pop-stack-1)))))
```

—————

8.2.33 defun PARSE-Reduction

[PARSE-ReductionOp p417]
 [must p452]
 [PARSE-Expr p412]
 [push-reduction p454]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— **defun PARSE-Reduction** —

```
(defun |PARSE-Reduction| ()
  (and (|PARSE-ReductionOp|) (must (|PARSE-Expr| 1000))
    (push-reduction '|PARSE-Reduction|
      (list '|Reduce| (pop-stack-2) (pop-stack-1) ))))
```

—————

8.2.34 defun PARSE-ReductionOp

[getl p??]
 [current-symbol p446]
 [match-next-token p446]
 [action p453]
 [advance-token p448]

— **defun PARSE-ReductionOp** —

```
(defun |PARSE-ReductionOp| ()
  (and (getl (current-symbol) '|Led|)
    (match-next-token 'special-char (code-char 47))
    (push-reduction '|PARSE-ReductionOp| (current-symbol))
    (action (advance-token)) (action (advance-token))))
```

—————

8.2.35 defun PARSE-Form

[match-advance-string p441]
 [bang p??]
 [optional p453]
 [must p452]
 [push-reduction p454]
 [pop-stack-1 p462]

[PARSE-Application p418]

— **defun PARSE-Form** —

```
(defun |PARSE-Form| ()
  (or (and (match-advance-string "iterate")
    (bang fil_test
      (optional
        (and (match-advance-string "from")
          (must (|PARSE-Label|))
          (push-reduction '|PARSE-Form|
            (list (pop-stack-1))))))
      (push-reduction '|PARSE-Form|
        (cons '|iterate| (append (pop-stack-1) nil))))
    (and (match-advance-string "yield") (must (|PARSE-Application|))
      (push-reduction '|PARSE-Form|
        (list '|yield| (pop-stack-1))))
    (|PARSE-Application|)))
```

8.2.36 defun PARSE-Application

[PARSE-Primary p420]

[optional p453]

[star p453]

[PARSE-Selector p419]

[PARSE-Application p418]

[push-reduction p454]

[pop-stack-2 p463]

[pop-stack-1 p462]

— **defun PARSE-Application** —

```
(defun |PARSE-Application| ()
  (and (|PARSE-Primary|) (optional (star opt_expr (|PARSE-Selector|)))
    (optional
      (and (|PARSE-Application|)
        (push-reduction '|PARSE-Application|
          (list (pop-stack-2) (pop-stack-1)))))))
```

8.2.37 defun PARSE-Label

```
[match-advance-string p441]
[must p452]
[PARSE-Name p427]
```

— **defun PARSE-Label** —

```
(defun |PARSE-Label| ()
  (and (match-advance-string "<<") (must (|PARSE-Name|))
        (must (match-advance-string ">>"))))
```

8.2.38 defun PARSE-Selector

```
[current-symbol p446]
[char-ne p450]
[current-char p449]
[match-advance-string p441]
[must p452]
[PARSE-PrimaryNoFloat p420]
[push-reduction p454]
[pop-stack-2 p463]
[pop-stack-1 p462]
[PARSE-Float p421]
[PARSE-Primary p420]
[$boot p??]
```

— **defun PARSE-Selector** —

```
(defun |PARSE-Selector| ()
  (declare (special $boot))
  (or (and nonblank (eq (current-symbol) '|.|)
        (char-ne (current-char) '| |) (match-advance-string ".")
        (must (|PARSE-PrimaryNoFloat|))
        (must (or (and $boot
            (push-reduction '|PARSE-Selector|
              (list 'elt (pop-stack-2) (pop-stack-1))))
          (push-reduction '|PARSE-Selector|
            (list (pop-stack-2) (pop-stack-1)))))))
    (and (or (|PARSE-Float|)
      (and (match-advance-string ".")
        (must (|PARSE-Primary|))))
      (must (or (and $boot
        (push-reduction '|PARSE-Selector|
```

```

      (list 'elt (pop-stack-2) (pop-stack-1))))
(push-reduction '|PARSE-Selector|
  (list (pop-stack-2) (pop-stack-1))))))

```

8.2.39 defun PARSE-PrimaryNoFloat

[PARSE-Primary1 p420]
 [optional p453]
 [PARSE-TokTail p416]

— defun PARSE-PrimaryNoFloat —

```

(defun |PARSE-PrimaryNoFloat| ()
  (and (|PARSE-Primary1|) (optional (|PARSE-TokTail|))))

```

8.2.40 defun PARSE-Primary

[PARSE-Float p421]
 [PARSE-PrimaryNoFloat p420]

— defun PARSE-Primary —

```

(defun |PARSE-Primary| ()
  (or (|PARSE-Float|) (|PARSE-PrimaryNoFloat|)))

```

8.2.41 defun PARSE-Primary1

[PARSE-VarForm p426]
 [optional p453]
 [current-symbol p446]
 [PARSE-Primary1 p420]
 [must p452]
 [pop-stack-2 p463]
 [pop-stack-1 p462]
 [push-reduction p454]
 [PARSE-Quad p425]

```
[PARSE-String p425]
[PARSE-IntegerTok p424]
[PARSE-FormalParameter p425]
[match-string p440]
[PARSE-Data p428]
[match-advance-string p441]
[PARSE-Expr p412]
[PARSE-Sequence p431]
[PARSE-Enclosure p424]
[$boot p??]
```

— **defun PARSE-Primary1** —

```
(defun |PARSE-Primary1| ()
  (declare (special $boot))
  (or (and (|PARSE-VarForm|)
    (optional
      (and nonblank (eq (current-symbol) '|(|)
        (must (|PARSE-Primary1|))
        (push-reduction '|PARSE-Primary1|
          (list (pop-stack-2) (pop-stack-1))))))
    (|PARSE-Quad|) (|PARSE-String|) (|PARSE-IntegerTok|)
    (|PARSE-FormalParameter|)
    (and (match-string "'")
      (must (or (and $boot (|PARSE-Data|))
        (and (match-advance-string "'")
          (must (|PARSE-Expr| 999))
          (push-reduction '|PARSE-Primary1|
            (list 'quote (pop-stack-1)))))))
    (|PARSE-Sequence|) (|PARSE-Enclosure|))))
```

—————

8.2.42 **defun PARSE-Float**

```
[PARSE-FloatBase p422]
[must p452]
[PARSE-FloatExponent p423]
[push-reduction p454]
[make-float p??]
[pop-stack-4 p463]
[pop-stack-3 p463]
[pop-stack-2 p463]
[pop-stack-1 p462]
```

— **defun PARSE-Float** —

```
(defun |PARSE-Float| ()
  (and (|PARSE-FloatBase|)
    (must (or (and nonblank (|PARSE-FloatExponent|))
      (push-reduction '|PARSE-Float| 0)))
    (push-reduction '|PARSE-Float|
      (make-float (pop-stack-4) (pop-stack-2) (pop-stack-2)
        (pop-stack-1))))))
```

8.2.43 defun PARSE-FloatBase

```
[current-symbol p446]
[char-eq p450]
[current-char p449]
[char-ne p450]
[next-char p449]
[PARSE-IntegerTok p424]
[must p452]
[PARSE-FloatBasePart p422]
[PARSE-IntegerTok p424]
[push-reduction p454]
[PARSE-FloatBase digitp (vol5)]
```

— defun PARSE-FloatBase —

```
(defun |PARSE-FloatBase| ()
  (or (and (integerp (current-symbol)) (char-eq (current-char) ".")
    (char-ne (next-char) ".") (|PARSE-IntegerTok|)
    (must (|PARSE-FloatBasePart|))))
    (and (integerp (current-symbol))
      (char-eq (char-upcase (current-char)) 'e)
      (|PARSE-IntegerTok|) (push-reduction '|PARSE-FloatBase| 0)
      (push-reduction '|PARSE-FloatBase| 0))
    (and (digitp (current-char)) (eq (current-symbol) '|.|)
      (push-reduction '|PARSE-FloatBase| 0)
      (|PARSE-FloatBasePart|))))
```

8.2.44 defun PARSE-FloatBasePart

```
[match-advance-string p441]
[must p452]
```

— defun PARSE-FloatBasePart —

— defun PARSE-FloatExponent —

[illegible]

```

(push-reduction '|PARSE-FloatExponent| 0)))
(and (identp (current-symbol))
      (setq g1 (floatexpid (current-symbol)))
      (action (advance-token))
      (push-reduction '|PARSE-FloatExponent| g1))))

```

8.2.46 defun PARSE-Enclosure

```

[match-advance-string p441]
[must p452]
[PARSE-Expr p412]
[push-reduction p454]
[pop-stack-1 p462]

```

— defun PARSE-Enclosure —

```

(defun |PARSE-Enclosure| ()
  (or (and (match-advance-string "(")
            (must (or (and (|PARSE-Expr| 6)
                          (must (match-advance-string ")"))
                          (and (match-advance-string ")")
                              (push-reduction '|PARSE-Enclosure|
                                              (list '|@Tuple|))))))
      (and (match-advance-string "{")
            (must (or (and (|PARSE-Expr| 6)
                          (must (match-advance-string "}"))
                          (push-reduction '|PARSE-Enclosure|
                                              (cons '|brace|
                                                    (list (list '|construct| (pop-stack-1)))))
                          (and (match-advance-string "}")
                              (push-reduction '|PARSE-Enclosure|
                                              (list '|brace|)))))))))

```

8.2.47 defun PARSE-IntegerTok

```

[parse-number p460]

```

— defun PARSE-IntegerTok —

```

(defun |PARSE-IntegerTok| () (parse-number))

```

8.2.48 defun PARSE-FormalParameter

[PARSE-FormalParameterTok p425]

— defun PARSE-FormalParameter —

```
(defun |PARSE-FormalParameter| () (|PARSE-FormalParameterTok|))
```

8.2.49 defun PARSE-FormalParameterTok

[parse-argument-designator p460]

— defun PARSE-FormalParameterTok —

```
(defun |PARSE-FormalParameterTok| () (parse-argument-designator))
```

8.2.50 defun PARSE-Quad

[match-advance-string p441]

[push-reduction p454]

[PARSE-GlyphTok p430]

[\$boot p??]

— defun PARSE-Quad —

```
(defun |PARSE-Quad| ()
  (or (and (match-advance-string "$")
            (push-reduction '|PARSE-Quad| '$))
      (and $boot (|PARSE-GlyphTok| '|.|')
            (push-reduction '|PARSE-Quad| '|.|'))))
```

8.2.51 defun PARSE-String

[parse-spadstring p458]

— **defun PARSE-String** —

```
(defun |PARSE-String| () (parse-spadstring))
```

8.2.52 **defun PARSE-VarForm**

[PARSE-Name p427]
 [optional p453]
 [PARSE-Scripts p426]
 [push-reduction p454]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— **defun PARSE-VarForm** —

```
(defun |PARSE-VarForm| ()
  (and (|PARSE-Name|)
    (optional
      (and (|PARSE-Scripts|)
        (push-reduction ' |PARSE-VarForm|
          (list ' |Scripts| (pop-stack-2) (pop-stack-1))))))
    (push-reduction ' |PARSE-VarForm| (pop-stack-1))))
```

8.2.53 **defun PARSE-Scripts**

[match-advance-string p441]
 [must p452]
 [PARSE-ScriptItem p427]

— **defun PARSE-Scripts** —

```
(defun |PARSE-Scripts| ()
  (and nonblank (match-advance-string "[" (must (|PARSE-ScriptItem|))
    (must (match-advance-string "]"")))))
```

8.2.54 defun PARSE-ScriptItem

[PARSE-Expr p412]
 [optional p453]
 [star p453]
 [match-advance-string p441]
 [must p452]
 [PARSE-ScriptItem p427]
 [push-reduction p454]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— **defun PARSE-ScriptItem** —

```
(defun |PARSE-ScriptItem| ()
  (or (and (|PARSE-Expr| 90)
    (optional
      (and (star repeater
        (and (match-advance-string ";")
          (must (|PARSE-ScriptItem|))))
        (push-reduction '|PARSE-ScriptItem|
          (cons '|;|
            (cons (pop-stack-2)
              (append (pop-stack-1) nil)))))))
      (and (match-advance-string ";") (must (|PARSE-ScriptItem|))
        (push-reduction '|PARSE-ScriptItem|
          (list '|PrefixSC| (pop-stack-1)))))))
```

—————

8.2.55 defun PARSE-Name

[parse-identifier p459]
 [push-reduction p454]
 [pop-stack-1 p462]

— **defun PARSE-Name** —

```
(defun |PARSE-Name| ()
  (and (parse-identifier) (push-reduction '|PARSE-Name| (pop-stack-1))))
```

—————

8.2.56 defun PARSE-Data

[action p453]
 [PARSE-Sexpr p428]
 [push-reduction p454]
 [translabel p455]
 [pop-stack-1 p462]
 [labasoc p??]

— **defun PARSE-Data** —

```
(defun |PARSE-Data| ()
  (declare (special lablasoc))
  (and (action (setq lablasoc nil)) (|PARSE-Sexpr|)
    (push-reduction '|PARSE-Data|
      (list 'quote (translabel (pop-stack-1) lablasoc)))))
```

—————

8.2.57 defun PARSE-Sexpr

[PARSE-Sexpr1 p428]

— **defun PARSE-Sexpr** —

```
(defun |PARSE-Sexpr| ()
  (and (action (advance-token)) (|PARSE-Sexpr1|)))
```

—————

8.2.58 defun PARSE-Sexpr1

[PARSE-AnyId p430]
 [optional p453]
 [PARSE-NBGlyphTok p429]
 [must p452]
 [PARSE-Sexpr1 p428]
 [action p453]
 [pop-stack-2 p463]
 [nth-stack p464]
 [match-advance-string p441]
 [push-reduction p454]
 [PARSE-IntegerTok p424]
 [pop-stack-1 p462]

[PARSE-String p425]
 [bang p??]
 [star p453]
 [PARSE-GlyphTok p430]

— defun PARSE-Sexpr1 —

```
(defun |PARSE-Sexpr1| ()
  (or (and (|PARSE-AnyId|)
    (optional
      (and (|PARSE-NBGlyphTok| '=) (must (|PARSE-Sexpr1|))
        (action (setq lablasoc
          (cons (cons (pop-stack-2)
            (nth-stack 1))
            lablasoc))))))
    (and (match-advance-string "'") (must (|PARSE-Sexpr1|))
      (push-reduction '|PARSE-Sexpr1|
        (list 'quote (pop-stack-1))))
    (|PARSE-IntegerTok|)
    (and (match-advance-string "-") (must (|PARSE-IntegerTok|))
      (push-reduction '|PARSE-Sexpr1| (- (pop-stack-1))))
    (|PARSE-String|)
    (and (match-advance-string "<")
      (bang fil_test (optional (star repeater (|PARSE-Sexpr1|))))
      (must (match-advance-string ">"))
      (push-reduction '|PARSE-Sexpr1| (list2vec (pop-stack-1))))
    (and (match-advance-string "(")
      (bang fil_test
        (optional
          (and (star repeater (|PARSE-Sexpr1|))
            (optional
              (and (|PARSE-GlyphTok| '|.|)
                (must (|PARSE-Sexpr1|))
                (push-reduction '|PARSE-Sexpr1|
                  (nconc (pop-stack-2) (pop-stack-1))))))))
      (must (match-advance-string ")")))))
```

—————

8.2.59 defun PARSE-NBGlyphTok

[match-current-token p445]
 [action p453]
 [advance-token p448]
 [tok p402]

— defun PARSE-NBGlyphTok —

```
(defun |PARSE-NBGlyphTok| (|tok|)
  (declare (special |tok|))
  (and (match-current-token 'gliph |tok|) nonblank (action (advance-token))))
```

8.2.60 defun PARSE-GlyphTok

```
[match-current-token p445]
[action p453]
[advance-token p448]
[tok p402]
```

— defun PARSE-GlyphTok —

```
(defun |PARSE-GlyphTok| (|tok|)
  (declare (special |tok|))
  (and (match-current-token 'gliph |tok|) (action (advance-token))))
```

8.2.61 defun PARSE-AnyId

```
[parse-identifier p459]
[match-string p440]
[push-reduction p454]
[current-symbol p446]
[action p453]
[advance-token p448]
[parse-keyword p460]
```

— defun PARSE-AnyId —

```
(defun |PARSE-AnyId| ()
  (or (parse-identifier)
      (or (and (match-string "$")
                (push-reduction '|PARSE-AnyId| (current-symbol))
                (action (advance-token)))
          (parse-keyword)))))
```

8.2.64 defun PARSE-OpenBracket

[getToken p444]
 [current-symbol p446]
 [eqcar p??]
 [push-reduction p454]
 [action p453]
 [advance-token p448]

— defun PARSE-OpenBracket —

```
(defun |PARSE-OpenBracket| ()
  (let (g1)
    (and (eq (|getToken| (setq g1 (current-symbol)))) '[])
    (must (or (and (eqcar g1 '|elt|)
                  (push-reduction '|PARSE-OpenBracket|
                                (list '|elt| (second g1) '|construct|)))
              (push-reduction '|PARSE-OpenBracket| '|construct|)))
    (action (advance-token))))))
```

—————

8.2.65 defun PARSE-OpenBrace

[getToken p444]
 [current-symbol p446]
 [eqcar p??]
 [push-reduction p454]
 [action p453]
 [advance-token p448]

— defun PARSE-OpenBrace —

```
(defun |PARSE-OpenBrace| ()
  (let (g1)
    (and (eq (|getToken| (setq g1 (current-symbol)))) '{})
    (must (or (and (eqcar g1 '|elt|)
                  (push-reduction '|PARSE-OpenBrace|
                                (list '|elt| (second g1) '|brace|)))
              (push-reduction '|PARSE-OpenBrace| '|construct|)))
    (action (advance-token))))))
```

—————

8.2.66 defun PARSE-IteratorTail

[match-advance-string p441]
 [bang p??]
 [optional p453]
 [star p453]
 [PARSE-Iterator p433]

— **defun PARSE-IteratorTail** —

```
(defun |PARSE-IteratorTail| ()
  (or (and (match-advance-string "repeat")
    (bang fil_test (optional (star repeater (|PARSE-Iterator|))))))
    (star repeater (|PARSE-Iterator|))))
```

—————

8.2.67 defun PARSE-Iterator

[match-advance-string p441]
 [must p452]
 [PARSE-Primary p420]
 [PARSE-Expression p411]
 [PARSE-Expr p412]
 [pop-stack-3 p463]
 [pop-stack-2 p463]
 [pop-stack-1 p462]
 [optional p453]

— **defun PARSE-Iterator** —

```
(defun |PARSE-Iterator| ()
  (or (and (match-advance-string "for") (must (|PARSE-Primary|))
    (must (match-advance-string "in"))
    (must (|PARSE-Expression|))
    (must (or (and (match-advance-string "by")
      (must (|PARSE-Expr| 200))
      (push-reduction '|PARSE-Iterator|
        (list 'inby (pop-stack-3)
          (pop-stack-2) (pop-stack-1))))
      (push-reduction '|PARSE-Iterator|
        (list 'in (pop-stack-2) (pop-stack-1))))))
    (optional
      (and (match-advance-string "|")
        (must (|PARSE-Expr| 111))
        (push-reduction '|PARSE-Iterator|
```



```

                (list '|\\| (pop-stack-1))))))
    (and (match-advance-string "while") (must (|PARSE-Expr| 190))
      (push-reduction '|PARSE-Iterator|
        (list 'while (pop-stack-1))))
    (and (match-advance-string "until") (must (|PARSE-Expr| 190))
      (push-reduction '|PARSE-Iterator|
        (list 'until (pop-stack-1))))))

```

8.2.68 The PARSE implicit routines

These symbols are not explicitly referenced in the source. Nevertheless, they are called during runtime. For example, PARSE-SemiColon is called in the chain:

```

PARSE-Enclosure {loc0=nil,loc1="(V ==> Vector; " } [ihs=35]
PARSE-Expr
  PARSE-LedPart
    PARSE-Operation
      PARSE-getSemanticForm
        PARSE-SemiColon

```

so there is a bit of indirection involved in the call.

8.2.69 defun PARSE-Suffix

```

[push-reduction p454]
[current-symbol p446]
[action p453]
[advance-token p448]
[optional p453]
[PARSE-TokTail p416]
[pop-stack-1 p462]

```

— defun PARSE-Suffix —

```

(defun |PARSE-Suffix| ()
  (and (push-reduction '|PARSE-Suffix| (current-symbol))
    (action (advance-token)) (optional (|PARSE-TokTail|))
    (push-reduction '|PARSE-Suffix|
      (list (pop-stack-1) (pop-stack-1))))))

```

8.2.70 defun PARSE-SemiColon

```
[match-advance-string p441]
[must p452]
[PARSE-Expr p412]
[push-reduction p454]
[pop-stack-2 p463]
[pop-stack-1 p462]
```

— **defun PARSE-SemiColon** —

```
(defun |PARSE-SemiColon| ()
  (and (match-advance-string ";")
        (must (or (|PARSE-Expr| 82)
                   (push-reduction '|PARSE-SemiColon| '|/throwAway|)))
        (push-reduction '|PARSE-SemiColon|
                          (list '|;| (pop-stack-2) (pop-stack-1)))))
```

—————

8.2.71 defun PARSE-Return

```
[match-advance-string p441]
[must p452]
[PARSE-Expression p411]
[push-reduction p454]
[pop-stack-1 p462]
```

— **defun PARSE-Return** —

```
(defun |PARSE-Return| ()
  (and (match-advance-string "return") (must (|PARSE-Expression|))
        (push-reduction '|PARSE-Return|
                          (list '|return| (pop-stack-1)))))
```

—————

8.2.72 defun PARSE-Exit

```
[match-advance-string p441]
[must p452]
[PARSE-Expression p411]
[push-reduction p454]
[pop-stack-1 p462]
```

— defun PARSE-Exit —

```
(defun |PARSE-Exit| ()
  (and (match-advance-string "exit")
        (must (or (|PARSE-Expression|)
                   (push-reduction '|PARSE-Exit| '$NoValue|))))
  (push-reduction '|PARSE-Exit|
    (list '|exit| (pop-stack-1)))))
```

— —

8.2.73 defun PARSE-Leave

```
[match-advance-string p441]
[PARSE-Expression p411]
[must p452]
[push-reduction p454]
[PARSE-Label p419]
[pop-stack-1 p462]
```

— defun PARSE-Leave —

```
(defun |PARSE-Leave| ()
  (and (match-advance-string "leave")
        (must (or (|PARSE-Expression|)
                   (push-reduction '|PARSE-Leave| '$NoValue|))))
  (must (or (and (match-advance-string "from")
                 (must (|PARSE-Label|))
                 (push-reduction '|PARSE-Leave|
                   (list '|leaveFrom| (pop-stack-1) (pop-stack-1)))))
        (push-reduction '|PARSE-Leave|
          (list '|leave| (pop-stack-1)))))
```

— —

8.2.74 defun PARSE-Seg

```
[PARSE-GlyphTok p430]
[bang p??]
[optional p453]
[PARSE-Expression p411]
[push-reduction p454]
[pop-stack-2 p463]
```

[pop-stack-1 p462]

— **defun PARSE-Seg** —

```
(defun |PARSE-Seg| ()
  (and (|PARSE-GlyphTok| '|\...|)
        (bang fil_test (optional (|PARSE-Expression|)))
        (push-reduction '|PARSE-Seg|
          (list 'segment (pop-stack-2) (pop-stack-1))))))
```

—————

8.2.75 defun PARSE-Conditional

[match-advance-string p441]
 [must p452]
 [PARSE-Expression p411]
 [bang p??]
 [optional p453]
 [PARSE-ElseClause p437]
 [push-reduction p454]
 [pop-stack-3 p463]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— **defun PARSE-Conditional** —

```
(defun |PARSE-Conditional| ()
  (and (match-advance-string "if") (must (|PARSE-Expression|))
        (must (match-advance-string "then")) (must (|PARSE-Expression|))
        (bang fil_test
          (optional
            (and (match-advance-string "else")
                  (must (|PARSE-ElseClause|))))))
        (push-reduction '|PARSE-Conditional|
          (list '|if| (pop-stack-3) (pop-stack-2) (pop-stack-1))))))
```

—————

8.2.76 defun PARSE-ElseClause

[current-symbol p446]
 [PARSE-Conditional p437]
 [PARSE-Expression p411]

— defun PARSE-ElseClause —

```
(defun |PARSE-ElseClause| ()
  (or (and (eq (current-symbol) '|if|) (|PARSE-Conditional|))
      (|PARSE-Expression|)))
```

8.2.77 defun PARSE-Loop

[star p453]
 [PARSE-Iterator p433]
 [must p452]
 [match-advance-string p441]
 [PARSE-Expr p412]
 [push-reduction p454]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— defun PARSE-Loop —

```
(defun |PARSE-Loop| ()
  (or (and (star repeater (|PARSE-Iterator|))
          (must (match-advance-string "repeat"))
          (must (|PARSE-Expr| 110))
          (push-reduction '|PARSE-Loop|
                          (cons 'repeat
                                (append (pop-stack-2) (list (pop-stack-1))))))
      (and (match-advance-string "repeat") (must (|PARSE-Expr| 110))
          (push-reduction '|PARSE-Loop|
                          (list 'repeat (pop-stack-1))))))
```

8.2.78 defun PARSE-LabelExpr

[PARSE-Label p419]
 [must p452]
 [PARSE-Expr p412]
 [push-reduction p454]
 [pop-stack-2 p463]
 [pop-stack-1 p462]

— defun PARSE-LabelExpr —

```
(defun |PARSE-LabelExpr| ()
  (and (|PARSE-Label|) (must (|PARSE-Expr| 120))
    (push-reduction '|PARSE-LabelExpr|
      (list 'label (pop-stack-2) (pop-stack-1))))))
```

8.2.79 defun PARSE-FloatTok

```
[parse-number p460]
[push-reduction p454]
[pop-stack-1 p462]
[bfp- p??]
[$boot p??]
```

— defun PARSE-FloatTok —

```
(defun |PARSE-FloatTok| ()
  (declare (special $boot))
  (and (parse-number)
    (push-reduction '|PARSE-FloatTok|
      (if $boot (pop-stack-1) (bfp- (pop-stack-1))))))
```

8.3 The PARSE support routines

This section is broken up into 3 levels:

- String grabbing: Match String, Match Advance String
- Token handling: Current Token, Next Token, Advance Token
- Character handling: Current Char, Next Char, Advance Char
- Line handling: Next Line, Print Next Line
- Error Handling
- Floating Point Support
- Dollar Translation

8.3.1 String grabbing

String grabbing is the art of matching initial segments of the current line, and removing them from the line before the get tokenized if they match (or removing the corresponding current tokens).

8.3.2 defun match-string

The match-string function returns length of X if X matches initial segment of inputstream.

[unget-tokens p444]
 [skip-blanks p440]
 [line-past-end-p p537]
 [current-char p449]
 [initial-substring-p p442]
 [subseq p??]
 [\$line p535]
 [line p535]

— defun match-string —

```
(defun match-string (x)
  (unget-tokens) ; So we don't get out of synch with token stream
  (skip-blanks)
  (if (and (not (line-past-end-p current-line)) (current-char) )
      (initial-substring-p x
        (subseq (line-buffer current-line) (line-current-index current-line))))))
```

—————

8.3.3 defun skip-blanks

[current-char p449]
 [token-lookahead-type p441]
 [advance-char p??]

— defun skip-blanks —

```
(defun skip-blanks ()
  (loop (let ((cc (current-char)))
        (if (not cc) (return nil))
        (if (eq (token-lookahead-type cc) 'white)
            (if (not (advance-char)) (return nil))
            (return t)))))
```

— **initvars** —

```
(defvar Escape-Character #\\ "Superquoting character.")
```

8.3.4 defun token-lookahead-type

[Escape-Character p??]

— **defun token-lookahead-type** —

```
(defun token-lookahead-type (char)
  "Predicts the kind of token to follow, based on the given initial character."
  (declare (special Escape-Character))
  (cond
    ((not char)                                     'eof)
    ((or (char= char Escape-Character) (alpha-char-p char)) 'id)
    ((digitp char)                                     'num)
    ((char= char #\')                                  'string)
    ((char= char #\[)                                  'bstring)
    ((member char '(#\Space #\Tab #\Return) :test #'char=) 'white)
    (t                                                'special-char)))
```

8.3.5 defun match-advance-string

The match-string function returns length of X if X matches initial segment of inputstream. If it is successful, advance inputstream past X. [quote-if-string p442]

[current-token p447]
 [match-string p440]
 [line-current-index p??]
 [line-past-end-p p537]
 [line-current-char p??]
 [\$token p90]
 [\$line p535]

— **defun match-advance-string** —

```
(defun match-advance-string (x)
```



```

(let ((y (if (>= (length (string x))
                (length (string (quote-if-string (current-token))))
              (match-string x)
              nil))) ; must match at least the current token
  (when y
    (incf (line-current-index current-line) y)
    (if (not (line-past-end-p current-line))
        (setf (line-current-char current-line)
              (elt (line-buffer current-line)
                  (line-current-index current-line)))
        (setf (line-current-char current-line) #\space))
    (setq prior-token
          (make-token :symbol (intern (string x))
                     :type 'identifier
                     :nonblank nonblank))
    t)))

```

8.3.6 defun initial-substring-p

[string-not-greaterp p??]

— defun initial-substring-p —

```

(defun initial-substring-p (part whole)
  "Returns length of part if part matches initial segment of whole."
  (let ((x (string-not-greaterp part whole)))
    (and x (= x (length part)) x)))

```

8.3.7 defun quote-if-string

[token-type p??]
 [strconc p??]
 [token-symbol p??]
 [underscore p444]
 [token-nonblank p??]
 [pack p??]
 [escape-keywords p443]
 [\$boot p??]
 [\$spad p489]

— defun quote-if-string —

```

(defun quote-if-string (token)
  (declare (special $boot $spad))
  (when token ;only use token-type on non-null tokens
    (case (token-type token)
      (bstring      (strconc "[" (token-symbol token) "]*"))
      (string       (strconc "'" (token-symbol token) "'"))
      (spadstring   (strconc "\" (underscore (token-symbol token)) "\"))
      (number       (format nil "~v,'OD" (token-nonblank token)
                            (token-symbol token)))
      (special-char (string (token-symbol token)))
      (identifier   (let ((id (symbol-name (token-symbol token)))
                          (pack (package-name (symbol-package
                                                (token-symbol token)))))
                      (if (or $boot $spad)
                          (if (string= pack "BOOT")
                              (escape-keywords (underscore id) (token-symbol token))
                              (concatenate 'string
                                             (underscore pack) "'" (underscore id)))
                          id)))
                    (token-symbol token))))))

```

8.3.8 defun escape-keywords

[\$keywords p??]

— defun escape-keywords —

```

(defun escape-keywords (pname id)
  (declare (special keywords))
  (if (member id keywords)
      (concatenate 'string "_" pname)
      pname))

```

8.3.9 defun isTokenDelimiter

NIL needed below since END_UNIT is not generated by current parser [current-symbol p446]

— defun isTokenDelimiter —

```

(defun |isTokenDelimiter| ()
  (member (current-symbol) '(\ end\_unit nil)))

```

8.3.10 defun underscore

[vector-push p??]

— defun underscore —

```
(defun underscore (string)
  (if (every #'alpha-char-p string)
      string
      (let* ((size (length string))
              (out-string (make-array (* 2 size)
                                      :element-type 'string-char
                                      :fill-pointer 0))
              next-char)
        (dotimes (i size)
          (setq next-char (char string i))
          (unless (alpha-char-p next-char) (vector-push #\_ out-string))
          (vector-push next-char out-string))
        out-string)))
```

8.3.11 Token Handling

8.3.12 defun getToken

[eqcar p??]

— defun getToken —

```
(defun |getToken| (x)
  (if (eqcar x '|elt|) (third x) x))
```

8.3.13 defun unget-tokens

```
[quote-if-string p442]
[line-current-segment p538]
[strconc p??]
[line-number p??]
```

[token-nonblank p??]
 [line-new-line p538]
 [line-number p??]
 [valid-tokens p91]

— **defun unget-tokens** —

```
(defun unget-tokens ()
  (case valid-tokens
    (0 t)
    (1 (let* ((cursym (quote-if-string current-token))
              (curline (line-current-segment current-line))
              (revised-line (strconc cursym curline (copy-seq " "))))
        (line-new-line revised-line current-line (line-number current-line))
        (setq nonblank (token-nonblank current-token))
        (setq valid-tokens 0)))
    (2 (let* ((cursym (quote-if-string current-token))
              (nextsym (quote-if-string next-token))
              (curline (line-current-segment Current-Line))
              (revised-line
               (strconc (if (token-nonblank current-token) " " " ")
                        cursym
                        (if (token-nonblank next-token) " " " ")
                        nextsym curline " ")))
        (setq nonblank (token-nonblank current-token))
        (line-new-line revised-line current-line (line-number current-line))
        (setq valid-tokens 0)))
    (t (error "How many tokens do you think you have?"))))
```

—

8.3.14 **defun match-current-token**

This returns the current token if it has EQ type and (optionally) equal symbol. [current-token p447]
 [match-token p446]

— **defun match-current-token** —

```
(defun match-current-token (type &optional (symbol nil))
  (match-token (current-token) type symbol))
```

—

8.3.15 defun match-token

[token-type p??]
[token-symbol p??]

— **defun match-token** —

```
(defun match-token (token type &optional (symbol nil))
  (when (and token (eq (token-type token) type))
    (if symbol
      (when (equal symbol (token-symbol token)) token)
      token)))
```

—————

8.3.16 defun match-next-token

This returns the next token if it has equal type and (optionally) equal symbol. [next-token p448]
[match-token p446]

— **defun match-next-token** —

```
(defun match-next-token (type &optional (symbol nil))
  (match-token (next-token) type symbol))
```

—————

8.3.17 defun current-symbol

[make-symbol-of p446]
[current-token p447]

— **defun current-symbol** —

```
(defun current-symbol ()
  (make-symbol-of (current-token)))
```

—————

8.3.18 defun make-symbol-of

[\$token p90]

— **defun make-symbol-of** —

```
(defun make-symbol-of (token)
  (let ((u (and token (token-symbol token))))
    (cond
      ((not u) nil)
      ((characterp u) (intern (string u)))
      (u))))
```

8.3.19 **defun current-token**

This returns the current token getting a new one if necessary. [try-get-token p447]
 [valid-tokens p91]
 [current-token p447]

— **defun current-token** —

```
(defun current-token ()
  (declare (special valid-tokens current-token))
  (if (> valid-tokens 0)
      current-token
      (try-get-token current-token)))
```

8.3.20 **defun try-get-token**

[get-token p449]
 [valid-tokens p91]

— **defun try-get-token** —

```
(defun try-get-token (token)
  (declare (special valid-tokens))
  (let ((tok (get-token token)))
    (when tok
      (incf valid-tokens)
      token)))
```

8.3.21 defun next-token

This returns the token after the current token, or NIL if there is none after. [try-get-token p447]

[current-token p447]
[valid-tokens p91]
[next-token p448]

— defun next-token —

```
(defun next-token ()
  (declare (special valid-tokens next-token))
  (current-token)
  (if (> valid-tokens 1)
      next-token
      (try-get-token next-token)))
```

—————

8.3.22 defun advance-token

This makes the next token be the current token. [current-token p447]

[copy-token p??]
[try-get-token p447]
[valid-tokens p91]
[current-token p447]

— defun advance-token —

```
(defun advance-token ()
  (current-token) ;don't know why this is needed
  (case valid-tokens
    (0 (try-get-token (current-token)))
    (1 (decf valid-tokens)
        (setq prior-token (copy-token current-token))
        (try-get-token current-token))
    (2 (setq prior-token (copy-token current-token))
        (setq current-token (copy-token next-token))
        (decf valid-tokens))))
```

—————

8.3.23 defvar \$XTokenReader— **initvars** —

```
(defvar XTokenReader 'get-meta-token "Name of tokenizing function")
```

—————

8.3.24 defun get-token

[XTokenReader p449]
 [XTokenReader p449]

— **defun get-token** —

```
(defun get-token (token)
  (funcall XTokenReader token))
```

—————

8.3.25 Character handling**8.3.26 defun current-char**

This returns the current character of the line, initially blank for an unread line. [line p535]
 [current-line p536]

— **defun current-char** —

```
(defun current-char ()
  (if (line-past-end-p current-line)
      #\return
      (line-current-char current-line)))
```

—————

8.3.27 defun next-char

This returns the character after the current character, blank if at end of line. The blank-at-end-of-line assumption is allowable because we assume that end-of-line is a token separator, which blank is equivalent to. [line-at-end-p p536]

[line-next-char p537]
 [current-line p536]

— **defun next-char** —

```
(defun next-char ()
  (if (line-at-end-p current-line)
      #\return
      (line-next-char current-line)))
```

—————

8.3.28 **defun char-eq**

— **defun char-eq** —

```
(defun char-eq (x y)
  (char= (character x) (character y)))
```

—————

8.3.29 **defun char-ne**

— **defun char-ne** —

```
(defun char-ne (x y)
  (char/= (character x) (character y)))
```

—————

8.3.30 **Error handling**

8.3.31 **defvar \$meta-error-handler**

— **initvars** —

```
(defvar meta-error-handler 'meta-meta-error-handler)
```

—————

8.3.32 defun meta-syntax-error

[meta-error-handler p450]
 [meta-error-handler p450]

— **defun meta-syntax-error** —

```
(defun meta-syntax-error (&optional (wanted nil) (parsing nil))
  (declare (special meta-error-handler))
  (funcall meta-error-handler wanted parsing))
```

—————

8.3.33 Floating Point Support**8.3.34 defun floatexpid**

TPDHERE: The use of and in spadreduce is undefined. rewrite this to loop

[floatexpid identp (vol5)]
 [floatexpid pname (vol5)]
 [spadreduce p??]
 [collect p365]
 [step p??]
 [maxindex p??]
 [floatexpid digitp (vol5)]

— **defun floatexpid** —

```
(defun floatexpid (x &aux s)
  (when (and (identp x) (char= (char-upcase (elt (setq s (pname x)) 0)) #\E)
    (> (length s) 1)
    (spadreduce and 0 (collect (step i 1 1 (maxindex s))
                              (digitp (elt s i)))))
    (read-from-string s t nil :start 1)))
```

—————

8.3.35 Dollar Translation**8.3.36 defun dollarTran**

[\$InteractiveMode p??]

— **defun dollarTran** —

```
(defun |dollarTran| (dom rand)
  (let ((eltWord (if |$InteractiveMode| '|$elt| '|elt|)))
    (declare (special |$InteractiveMode|))
    (if (and (not (atom rand)) (cdr rand))
        (cons (list eltWord dom (car rand)) (cdr rand))
        (list eltWord dom rand))))
```

8.3.37 Applying metagrammatical elements of a production (e.g., Star).

- **must** means that if it is not present in the token stream, it is a syntax error.
- **optional** means that if it is present in the token stream, that is a good thing, otherwise don't worry (like [foo] in BNF notation).
- **action** is something we do as a consequence of successful parsing; it is inserted at the end of the conjunction of requirements for a successful parse, and so should return T.
- **sequence** consists of a head, which if recognized implies that the tail must follow. Following tail are actions, which are performed upon recognizing the head and tail.

8.3.38 defmacro Bang

If the execution of prod does not result in an increase in the size of the stack, then stack a NIL. Return the value of prod.

— defmacro bang —

```
(defmacro bang (lab prod)
  '(progn
    (setf (stack-updated reduce-stack) nil)
    (let* ((prodvalue ,prod) (updated (stack-updated reduce-stack)))
      (unless updated (push-reduction ',lab nil))
      prodvalue)))
```

8.3.39 defmacro must

[meta-syntax-error p451]

— defmacro must —

```
(defmacro must (dothis &optional (this-is nil) (in-rule nil))
  `(or ,dothis (meta-syntax-error ,this-is ,in-rule)))
```

8.3.40 defun action

— defun action —

```
(defun action (dothis) (or dothis t))
```

8.3.41 defun optional

— defun optional —

```
(defun optional (dothis) (or dothis t))
```

8.3.42 defmacro star

Succeeds if there are one or more of PROD, stacking as one unit the sub-reductions of PROD and labelling them with LAB. E.G., (Star IDs (parse-id)) with A B C will stack (3 IDs (A B C)), where (parse-id) would stack (1 ID (A)) when applied once. [stack-size p??]
 [push-reduction p454]
 [pop-stack-1 p462]

— defmacro star —

```
(defmacro star (lab prod)
  `(prog ((oldstacksize (stack-size reduce-stack)))
    (if (not ,prod) (return nil))
  loop
    (if (not ,prod)
      (let* ((newstacksize (stack-size reduce-stack))
              (number-of-new-reductions (- newstacksize oldstacksize)))
        (if (> number-of-new-reductions 0)
          (return (do ((i 0 (1+ i)) (accum nil))
```

```

      ((= i number-of-new-reductions)
       (push-reduction ',lab accum)
       (return t))
      (push (pop-stack-1) accum)))
    (return t)))
  (go loop))))

```

8.3.43 Stacking and retrieving reductions of rules.

8.3.44 `defvar $reduce-stack`

Stack of results of reduced productions. [`$stack` p88]

— `initvars` —

```

(defvar reduce-stack (make-stack) )

```

8.3.45 `defmacro reduce-stack-clear`

— `defmacro reduce-stack-clear` —

```

(defmacro reduce-stack-clear () '(stack-load nil reduce-stack))

```

8.3.46 `defun push-reduction`

[`stack-push` p89]
 [`make-reduction` p??]
 [`reduce-stack` p454]

— `defun push-reduction` —

```

(defun push-reduction (rule redn)
  (stack-push (make-reduction :rule rule :value redn) reduce-stack))

```

Chapter 9

Utility Functions

9.0.47 defun translabel

[translabel1 p455]

— defun translabel —

```
(defun translabel (x al)
  (translabel1 x al) x)
```

—————

9.0.48 defun translabel1

[refvecp p??]
[maxindex p??]
[translabel1 p455]
[lassoc p??]

— defun translabel1 —

```
(defun translabel1 (x al)
  "Transforms X according to AL = ((<label> . Sexpr) ..)."
  (cond
    ((refvecp x)
     (do ((i 0 (1+ i)) (k (maxindex x)))
         ((> i k)
          (if (let ((y (lassoc (elt x i) al))) (setelt x i y))
              (translabel1 (elt x i) al))))
     ((atom x) nil)
    ((let ((y (lassoc (first x) al)))
```

```
(if y (setf (first x) y) (translabel1 (cdr x) al)))
((translabel1 (first x) al) (translabel1 (cdr x) al)))
```

9.0.49 defun displayPreCompilationErrors

```
[length p??]
[remdup p??]
[sayBrightly p??]
[nequal p??]
[sayMath p??]
[$postStack p??]
[$topOp p??]
[$InteractiveMode p??]
```

— defun displayPreCompilationErrors —

```
(defun |displayPreCompilationErrors| ()
  (let (n errors heading)
    (declare (special |$postStack| |$topOp| |$InteractiveMode|))
    (setq n (|#| (setq |$postStack| (remdup (nreverse |$postStack|)))))
    (unless (eql n 0)
      (setq errors (cond ((> n 1) "errors") (t "error")))
      (cond
        (|$InteractiveMode|
         (|sayBrightly| (list " Semantic " errors " detected: ")))
        (t
         (setq heading
           (if (nequal |$topOp| '|$topOp|)
             (list " " |$topOp| " has")
             (list " You have")))
         (|sayBrightly|
          (append heading (list n "precompilation " errors ":" )))))
      (cond
        ((> n 1)
         (let ((i 1))
           (dolist (x |$postStack|)
             (|sayMath| (cons " " (cons i (cons " " x))))))
          (t (|sayMath| (cons " " (car |$postStack|))))
          (terpri))))
```

9.0.50 defun bumperrorcount

[\$InteractiveMode p??]
[\$spad-errors p??]

— defun bumperrorcount —

```
(defun bumperrorcount (kind)
  (declare (special |$InteractiveMode| $spad_errors))
  (unless |$InteractiveMode|
    (let ((index (case kind
                    (|syntax| 0)
                    (|precompilation| 1)
                    (|semantic| 2)
                    (t (error (break "BUMPERERRORCOUNT: kind=~s~%" kind))))))
      (setelt $spad_errors index (1+ (elt $spad_errors index))))))
```

9.0.51 defun parseTranCheckForRecord

[qcar p??]
[qcdr p??]
[postError p356]
[parseTran p93]

— defun parseTranCheckForRecord —

```
(defun |parseTranCheckForRecord| (x op)
  (declare (ignore op))
  (let (tmp3)
    (setq x (|parseTran| x))
    (cond
      ((and (consp x) (eq (qfirst x) '|Record|))
        (cond
          ((do ((z nil tmp3) (tmp4 (qrest x) (cdr tmp4)) (y nil))
                ((or z (atom tmp4)) tmp3)
                (setq y (car tmp4))
                (cond
                  ((null (and (consp y) (eq (qfirst y) '|:|) (consp (qrest y))
                               (consp (qcddr y)) (eq (qcdddr y) nil)))
                    (setq tmp3 (or tmp3 y))))
              (|postError| (list " Constructor" x "has missing label" )))
            (t x))))
      (t x))))
```

9.0.52 defun new2OldLisp

```
[new2OldTran p??]
[postTransform p351]
```

— **defun new2OldLisp** —

```
(defun |new2OldLisp| (x)
  (|new2OldTran| (|postTransform| x)))
```

9.0.53 defun makeSimplePredicateOrNil

```
[isSimple p??]
[isAlmostSimple p??]
[wrapSEQExit p??]
```

— **defun makeSimplePredicateOrNil** —

```
(defun |makeSimplePredicateOrNil| (p)
  (let (u g)
    (cond
      ((|isSimple| p) nil)
      ((setq u (|isAlmostSimple| p)) u)
      (t (|wrapSEQExit| (list (list 'let (setq g (gensym)) p) g))))))
```

9.0.54 defun parse-spadstring

```
[match-current-token p445]
[token-symbol p??]
[push-reduction p454]
[advance-token p448]
```

— **defun parse-spadstring** —

```
(defun parse-spadstring ()
  (let* ((tok (match-current-token 'spadstring))
        (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'spadstring-token (copy-tree symbol))
      (advance-token))))
```

t)))

9.0.55 defun parse-string

[match-current-token p445]
[token-symbol p??]
[push-reduction p454]
[advance-token p448]

— defun parse-string —

```
(defun parse-string ()
  (let* ((tok (match-current-token 'string))
         (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'string-token (copy-tree symbol))
      (advance-token)
      t)))
```

9.0.56 defun parse-identifier

[match-current-token p445]
[token-symbol p??]
[push-reduction p454]
[advance-token p448]

— defun parse-identifier —

```
(defun parse-identifier ()
  (let* ((tok (match-current-token 'identifier))
         (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'identifier-token (copy-tree symbol))
      (advance-token)
      t)))
```

9.0.57 defun parse-number

[match-current-token p445]
 [token-symbol p??]
 [push-reduction p454]
 [advance-token p448]

— **defun parse-number** —

```
(defun parse-number ()
  (let* ((tok (match-current-token 'number))
         (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'number-token (copy-tree symbol))
      (advance-token)
      t)))
```

9.0.58 defun parse-keyword

[match-current-token p445]
 [token-symbol p??]
 [push-reduction p454]
 [advance-token p448]

— **defun parse-keyword** —

```
(defun parse-keyword ()
  (let* ((tok (match-current-token 'keyword))
         (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'keyword-token (copy-tree symbol))
      (advance-token)
      t)))
```

9.0.59 defun parse-argument-designator

[push-reduction p454]
 [match-current-token p445]
 [token-symbol p??]
 [advance-token p448]

— defun parse-argument-designator —

```
(defun parse-argument-designator ()
  (let* ((tok (match-current-token 'argument-designator))
         (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'argument-designator-token (copy-tree symbol))
      (advance-token)
      t)))
```

9.0.60 defun print-package

[out-stream p??]

— defun print-package —

```
(defun print-package (package)
  (declare (special out-stream))
  (format out-stream "~&~%(IN-PACKAGE ~S )~%~%" package))
```

9.0.61 defun checkWarning

[postError p356]
[concat p??]

— defun checkWarning —

```
(defun |checkWarning| (msg)
  (|postError| (|concat| "Parsing error: " msg)))
```

9.0.62 defun tuple2List

[tuple2List p461]
[postTranSegment p370]
[postTran p352]

```
[$boot p??]
[$InteractiveMode p??]
```

— **defun tuple2List** —

```
(defun |tuple2List| (arg)
  (let (u p q)
    (declare (special |$InteractiveMode| $boot))
    (when (consp arg)
      (setq u (|tuple2List| (qrest arg)))
      (cond
        ((and (consp (qfirst arg)) (eq (qcaar arg) 'segment)
              (consp (qcddar arg))
              (consp (qcddar arg))
              (eq (qcdddar arg) nil))
         (setq p (qcadar arg))
         (setq q (qcaddar arg))
         (cond
           ((null u) (list '|construct| (|postTranSegment| p q)))
           ((and |$InteractiveMode| (null $boot))
            (cons '|append|
                  (cons (list '|construct| (|postTranSegment| p q))
                        (list (|tuple2List| (qrest arg))))))
           (t
            (cons '|nconc|
                  (cons (list '|construct| (|postTranSegment| p q))
                        (list (|tuple2List| (qrest arg)))))))
         ((null u) (list '|construct| (|postTran| (qfirst arg))))
         (t (list '|cons| (|postTran| (qfirst arg)) (|tuple2List| (qrest arg)))))))
```

—————

9.0.63 defmacro pop-stack-1

```
[reduction-value p??]
[Pop-Reduction p464]
```

— **defmacro pop-stack-1** —

```
(defmacro pop-stack-1 () '(reduction-value (Pop-Reduction)))
```

—————

9.0.64 defmacro pop-stack-2

[stack-push p89]
 [reduction-value p??]
 [Pop-Reduction p464]

— defmacro pop-stack-2 —

```
(defmacro pop-stack-2 ()
  '(let* ((top (Pop-Reduction)) (next (Pop-Reduction)))
    (stack-push top Reduce-Stack)
    (reduction-value next)))
```

—————

9.0.65 defmacro pop-stack-3

[stack-push p89]
 [reduction-value p??]
 [Pop-Reduction p464]

— defmacro pop-stack-3 —

```
(defmacro pop-stack-3 ()
  '(let* ((top (Pop-Reduction)) (next (Pop-Reduction)) (nnext (Pop-Reduction)))
    (stack-push next Reduce-Stack)
    (stack-push top Reduce-Stack)
    (reduction-value nnext)))
```

—————

9.0.66 defmacro pop-stack-4

[stack-push p89]
 [reduction-value p??]
 [Pop-Reduction p464]

— defmacro pop-stack-4 —

```
(defmacro pop-stack-4 ()
  '(let* ((top (Pop-Reduction))
    (next (Pop-Reduction))
    (nnext (Pop-Reduction))
    (nnnext (Pop-Reduction)))
```

```

(stack-push nnext Reduce-Stack)
(stack-push next Reduce-Stack)
(stack-push top Reduce-Stack)
(reduction-value nnext)))

```

9.0.67 defmacro nth-stack

```

[stack-store p??]
[reduction-value p??]

```

— defmacro nth-stack —

```

(defmacro nth-stack (x)
  '(reduction-value (nth (1- ,x) (stack-store Reduce-Stack))))

```

9.0.68 defun Pop-Reduction

```

[stack-pop p90]

```

— defun Pop-Reduction —

```

(defun Pop-Reduction () (stack-pop Reduce-Stack))

```

9.0.69 defun addclose

```

[suffix p??]

```

— defun addclose —

```

(defun addclose (line char)
  (cond
    ((char= (char line (maxindex line)) #\; )
     (setelt line (maxindex line) char)
     (if (char= char #\;) line (suffix #\; line)))
    ((suffix char line))))

```

9.0.70 defun blankp

— defun blankp —

```
(defun blankp (char)
  (or (eq char #\Space) (eq char #\tab)))
```

—————

9.0.71 defun drop

Return a pointer to the Nth cons of X, counting 0 as the first cons. [drop p465]

[take p??]

[croak p??]

— defun drop —

```
(defun drop (n x &aux m)
  (cond
    ((eq1 n 0) x)
    ((> n 0) (drop (1- n) (cdr x)))
    ((>= (setq m (+ (length x) n)) 0) (take m x))
    ((croak (list "Bad args to DROP" n x)))))
```

—————

9.0.72 defun escaped

— defun escaped —

```
(defun escaped (str n)
  (and (> n 0) (eq (char str (1- n)) #\_)))
```

—————

9.0.73 defvar \$comblocklist

— initvars —


```
(defvar $comblocklist nil "a dynamic lists of comments for this block")
```

9.0.74 defun fincomblock

- NUM is the line number of the current line
- OLDNUMS is the list of line numbers of previous lines
- OLDLOCS is the list of previous indentation locations
- NCBLOCK is the current comment block

```
[preparse-echo p88]
[$comblocklist p465]
[$EchoLineStack p??]
```

— defun fincomblock —

```
(defun fincomblock (num oldnums oldlocs ncblock linelist)
  (declare (special $EchoLineStack $comblocklist))
  (push
   (cond
    ((eql (car ncblock) 0) (cons (1- num) (reverse (cdr ncblock))))
    ;; comment for constructor itself paired with 1st line -1
    (t
     (when $EchoLineStack
      (setq num (pop $EchoLineStack))
      (preparse-echo linelist)
      (setq $EchoLineStack (list num)))
     (cons
      ;; scan backwards for line to left of current
      (do ((onums oldnums (cdr onums))
          (olocs oldlocs (cdr olocs))
          (sloc (car ncblock)))
          ((null onums) nil)
          (when (and (numberp (car olocs)) (<= (car olocs) sloc))
            (return (car onums))))
      (reverse (cdr ncblock))))
    $comblocklist))
```

9.0.75 defun indent-pos

— defun indent-pos —

```
(defun indent-pos (str)
  (do ((i 0 (1+ i)) (pos 0))
      ((>= i (length str)) nil)
    (case (char str i)
      (#\space (incf pos))
      (#\tab (setq pos (next-tab-loc pos)))
      (otherwise (return pos)))))
```

9.0.76 defun infixtok

[string2id-n p??]

— defun infixtok —

```
(defun infixtok (s)
  (member (string2id-n s 1) '(|then| |else|) :test #'eq))
```

9.0.77 defun is-console

[fp-output-stream p??]

[*terminal-io* p??]

— defun is-console —

```
(defun is-console (stream)
  (and (streamp stream) (output-stream-p stream)
    (eq (system:fp-output-stream stream)
        (system:fp-output-stream *terminal-io*))))
```

9.0.78 defun next-tab-loc

— defun next-tab-loc —

```
(defun next-tab-loc (i)
  (* (1+ (truncate i 8)) 8))
```

9.0.79 defun nonblankloc

[blankp p465]

— **defun nonblankloc** —

```
(defun nonblankloc (str)
  (position-if-not #'blankp str))
```

—

9.0.80 defun parseprint— **defun parseprint** —

```
(defun parseprint (l)
  (when l
    (format t "~&~%          ***      PREPARSE          ***~%~%"
      (dolist (x l) (format t "~5d. ~a~%" (car x) (cdr x)))
      (format t "%"))))
```

—

9.0.81 defun skip-to-endif

[initial-substring p539]

[preparseReadLine p85]

[preparseReadLine1 p87]

[skip-to-endif p468]

— **defun skip-to-endif** —

```
(defun skip-to-endif (x)
  (let (line ind tmp1)
    (setq tmp1 (preparseReadLine1))
    (setq ind (car tmp1))
    (setq line (cdr tmp1))
    (cond
      ((not (stringp line)) (cons ind line))
      ((initial-substring line ")endif") (preparseReadLine x))
      ((initial-substring line ")fin") (cons ind nil))
      (t (skip-to-endif x)))))
```

—

Chapter 10

The Compiler

10.1 Compiling EQ.spad

Given the top level command:

```
)co EQ
```

The default call chain looks like:

```
1> (|compiler| ...)
2> (|compileSpad2Cmd| ...)
   Compiling AXIOM source code from file /tmp/A.spad using old system
   compiler.
3> (|compilerDoit| ...)
4> (|/RQ,LIB|)
5> (/RF-1 ...)
6> (SPAD ...)
AXSERV abbreviates package AxiomServer
7> (S-PROCESS ...)
8> (|compTopLevel| ...)
9> (|comp0rCroak| ...)
10> (|comp0rCroak1| ...)
11> (|comp| ...)
12> (|compNoStacking| ...)
13> (|comp2| ...)
14> (|comp3| ...)
15> (|compExpression| ...)
* 16> (|compWhere| ...)
   17> (|comp| ...)
   18> (|compNoStacking| ...)
   19> (|comp2| ...)
   20> (|comp3| ...)
   21> (|compExpression| ...)
```

```

22> (|compSeq| ...)
23> (|compSeq1| ...)
24> (|compSeqItem| ...)
25> (|comp| ...)
26> (|compNoStacking| ...)
27> (|comp2| ...)
28> (|comp3| ...)
29> (|compExpression| ...)
<29 (|compExpression| ...)
<28 (|comp3| ...)
<27 (|comp2| ...)
<26 (|compNoStacking| ...)
<25 (|comp| ...)
<24 (|compSeqItem| ...)
24> (|compSeqItem| ...)
25> (|comp| ...)
26> (|compNoStacking| ...)
27> (|comp2| ...)
28> (|comp3| ...)
29> (|compExpression| ...)
30> (|compExit| ...)
31> (|comp| ...)
32> (|compNoStacking| ...)
33> (|comp2| ...)
34> (|comp3| ...)
35> (|compExpression| ...)
<35 (|compExpression| ...)
<34 (|comp3| ...)
<33 (|comp2| ...)
<32 (|compNoStacking| ...)
<31 (|comp| ...)
31> (|modifyModeStack| ...)
<31 (|modifyModeStack| ...)
<30 (|compExit| ...)
<29 (|compExpression| ...)
<28 (|comp3| ...)
<27 (|comp2| ...)
<26 (|compNoStacking| ...)
<25 (|comp| ...)
<24 (|compSeqItem| ...)
24> (|replaceExitEtc| ...)
25> (|replaceExitEtc,fn| ...)
26> (|replaceExitEtc| ...)
27> (|replaceExitEtc,fn| ...)
28> (|replaceExitEtc| ...)
29> (|replaceExitEtc,fn| ...)
<29 (|replaceExitEtc,fn| ...)
<28 (|replaceExitEtc| ...)
28> (|replaceExitEtc| ...)
29> (|replaceExitEtc,fn| ...)

```

```

    <29 (|replaceExitEtc,fn| ...)
    <28 (|replaceExitEtc| ...)
    <27 (|replaceExitEtc,fn| ...)
    <26 (|replaceExitEtc| ...)
    26> (|replaceExitEtc| ...)
    27> (|replaceExitEtc,fn| ...)
    28> (|replaceExitEtc| ...)
    29> (|replaceExitEtc,fn| ...)
    30> (|replaceExitEtc| ...)
    31> (|replaceExitEtc,fn| ...)
    32> (|replaceExitEtc| ...)
    33> (|replaceExitEtc,fn| ...)
    <33 (|replaceExitEtc,fn| ...)
    <32 (|replaceExitEtc| ...)
    32> (|replaceExitEtc| ...)
    33> (|replaceExitEtc,fn| ...)
    <33 (|replaceExitEtc,fn| ...)
    <32 (|replaceExitEtc| ...)
    <31 (|replaceExitEtc,fn| ...)
    <30 (|replaceExitEtc| ...)
    30> (|convertOrCroak| ...)
    31> (|convert| ...)
    <31 (|convert| ...)
    <30 (|convertOrCroak| ...)
    <29 (|replaceExitEtc,fn| ...)
    <28 (|replaceExitEtc| ...)
    28> (|replaceExitEtc| ...)
    29> (|replaceExitEtc,fn| ...)
    <29 (|replaceExitEtc,fn| ...)
    <28 (|replaceExitEtc| ...)
    <27 (|replaceExitEtc,fn| ...)
    <26 (|replaceExitEtc| ...)
    <25 (|replaceExitEtc,fn| ...)
    <24 (|replaceExitEtc| ...)
    <23 (|compSeq1| ...)
    <22 (|compSeq| ...)
    <21 (|compExpression| ...)
    <20 (|comp3| ...)
    <19 (|comp2| ...)
    <18 (|compNoStacking| ...)
    <17 (|comp| ...)
    17> (|comp| ...)
    18> (|compNoStacking| ...)
    19> (|comp2| ...)
    20> (|comp3| ...)
    21> (|compExpression| ...)
    22> (|comp| ...)
    23> (|compNoStacking| ...)
    24> (|comp2| ...)
    25> (|comp3| ...)

```

```

26> (|compColon| ...)
<26 (|compColon| ...)
<25 (|comp3| ...)
<24 (|comp2| ...)
<23 (|compNoStacking| ...)
<22 (|comp| ...)

```

In order to explain the compiler we will walk through the compilation of EQ.spad, which handles equations as mathematical objects. We start the system. Most of the structure in Axiom are circular so we have to the `*print-cycle*` to true.

```
root@spiff:/tmp# axiom -nox
```

```
(1) -> )lisp (setq *print-circle* t)
```

```
Value = T
```

We trace the function we find interesting:

```
(1) -> )lisp (trace |compiler|)
```

```
Value = (|compiler|)
```

10.1.1 The top level compiler command

We compile the spad file. We can see that the `compiler` function gets a list

```
(1) -> )co EQ
```

```
1> (|compiler| (EQ))
```

In order to find this file, the `pathname` and `pathnameType` functions are used to find the location and pathname to the file. They `pathnameType` function eventually returns the fact that this is a spad source file. Once that is known we call the `compileSpad2Cmd` function with a list containing the full pathname as a string.

```

1> (|compiler| (EQ))
2> (|pathname| (EQ))
<2 (|pathname| #p"EQ")
2> (|pathnameType| #p"EQ")
3> (|pathname| #p"EQ")
<3 (|pathname| #p"EQ")
<2 (|pathnameType| NIL)
2> (|pathnameType| "/tmp/EQ.spad")
3> (|pathname| "/tmp/EQ.spad")
<3 (|pathname| #p"/tmp/EQ.spad")
<2 (|pathnameType| "spad")
2> (|pathnameType| "/tmp/EQ.spad")

```

```

3> (|pathname| "/tmp/EQ.spad")
<3 (|pathname| #p"/tmp/EQ.spad")
<2 (|pathnameType| "spad")
2> (|pathnameType| "/tmp/EQ.spad")
3> (|pathname| "/tmp/EQ.spad")
<3 (|pathname| #p"/tmp/EQ.spad")
<2 (|pathnameType| "spad")
2> (|compileSpad2Cmd| ("/tmp/EQ.spad"))

[compiler helpSpad2Cmd (vol5)]
[compiler selectOptionLC (vol5)]
[compiler pathname (vol5)]
[compiler mergePathnames (vol5)]
[compiler pathnameType (vol5)]
[compiler namestring (vol5)]
[throwKeyedMsg p??]
[findfile p??]
[compileSpad2Cmd p474]
[compileSpadLispCmd p528]
[$newConlist p??]
[$options p??]
[/editfile p??]

— defun compiler —

(defun |compiler| (args)
  "The top level compiler command"
  (let (|$newConlist| optlist optname optargs havenew haveold aft ef af af1)
    (declare (special |$newConlist| |$options| /editfile))
    (setq |$newConlist| nil)
    (cond
      ((and (null args) (null |$options|) (null /editfile))
        (|helpSpad2Cmd| '(|compiler|)))
      (t
        (cond ((null args) (setq args (cons /editfile nil))))
        (setq optlist '(|new| |old| |translate| |constructor|))
        (setq havenew nil)
        (setq haveold nil)
        (do ((t0 |$options| (cdr t0)) (opt nil))
            ((or (atom t0)
                 (progn (setq opt (car t0)) nil)
                 (null (null (and havenew haveold)))))
          nil)
        (setq optname (car opt))
        (setq optargs (cdr opt))
        (case (|selectOptionLC| optname optlist nil)
          (|new| (setq havenew t))
          (|translate| (setq haveold t))
          (|constructor| (setq haveold t))

```



```

      (|old|          (setq haveold t))))
(cond
  ((and havenew haveold) (|throwKeyedMsg| 's2iz0081 nil))
  (t
   (setq af (|pathname| args))
   (setq aft (|pathnameType| af))
   (cond
    ((or haveold (string= aft "spad"))
     (if (null (setq af1 ($findfile af '(|spad|))))
         (|throwKeyedMsg| 's2il0003 (cons (namestring af) nil))
         (|compileSpad2Cmd| (cons af1 nil))))
    ((string= aft "nrllib")
     (if (null (setq af1 ($findfile af '(|nrllib|))))
         (|throwKeyedMsg| 'S2IL0003 (cons (namestring af) nil))
         (|compileSpadLispCmd| (cons af1 nil))))
    (t
     (setq af1 ($findfile af '(|spad|)))
     (cond
      ((and af1 (string= (|pathnameType| af1) "spad"))
       (|compileSpad2Cmd| (cons af1 nil)))
      (t
       (setq ef (|pathname| /editfile))
       (setq ef (|mergePathnames| af ef))
       (cond
        ((boot-equal ef af) (|throwKeyedMsg| 's2iz0039 nil))
        (t
         (setq af ef)
         (cond
          ((string= (|pathnameType| af) "spad")
           (|compileSpad2Cmd| args))
          (t
           (setq af1 ($findfile af '(|spad|)))
           (cond
            ((and af1 (string= (|pathnameType| af1) "spad"))
             (|compileSpad2Cmd| (cons af1 nil)))
            (t (|throwKeyedMsg| 's2iz0039 nil))))))))))))))

```

10.1.2 The Spad compiler top level function

The argument to this function, as noted above, is a list containing the string pathname to the file.

```
2> (|compileSpad2Cmd| ("/tmp/EQ.spad"))
```

There is a fair bit of redundant work to find the full filename and pathname of the file. This needs to be eliminated.

The trace of the functions in this routines is:

```

1> (|selectOptionLC| "compiler" (|abbreviations| |boot| |browse| |cd| |clear| |close| |compiler| |copy
<1 (|selectOptionLC| |compiler|)
1> (|selectOptionLC| |compiler| (|abbreviations| |boot| |browse| |cd| |clear| |close| |compiler| |copy
<1 (|selectOptionLC| |compiler|)
1> (|pathname| (EQ))
<1 (|pathname| #p"EQ")
1> (|pathnameType| #p"EQ")
  2> (|pathname| #p"EQ")
  <2 (|pathname| #p"EQ")
<1 (|pathnameType| NIL)
1> (|pathnameType| "/tmp/EQ.spad")
  2> (|pathname| "/tmp/EQ.spad")
  <2 (|pathname| #p"/tmp/EQ.spad")
<1 (|pathnameType| "spad")
1> (|pathnameType| "/tmp/EQ.spad")
  2> (|pathname| "/tmp/EQ.spad")
  <2 (|pathname| #p"/tmp/EQ.spad")
<1 (|pathnameType| "spad")
1> (|pathnameType| "/tmp/EQ.spad")
  2> (|pathname| "/tmp/EQ.spad")
  <2 (|pathname| #p"/tmp/EQ.spad")
<1 (|pathnameType| "spad")
1> (|compileSpad2Cmd| ("/tmp/EQ.spad"))
  2> (|pathname| ("/tmp/EQ.spad"))
  <2 (|pathname| #p"/tmp/EQ.spad")
  2> (|pathnameType| #p"/tmp/EQ.spad")
    3> (|pathname| #p"/tmp/EQ.spad")
    <3 (|pathname| #p"/tmp/EQ.spad")
  <2 (|pathnameType| "spad")
  2> (|updateSourceFiles| #p"/tmp/EQ.spad")
    3> (|pathname| #p"/tmp/EQ.spad")
    <3 (|pathname| #p"/tmp/EQ.spad")
    3> (|pathname| #p"/tmp/EQ.spad")
    <3 (|pathname| #p"/tmp/EQ.spad")
    3> (|pathnameType| #p"/tmp/EQ.spad")
      4> (|pathname| #p"/tmp/EQ.spad")
      <4 (|pathname| #p"/tmp/EQ.spad")
    <3 (|pathnameType| "spad")
    3> (|pathname| ("EQ" "spad" "*"))
    <3 (|pathname| #p"EQ.spad")
    3> (|pathnameType| #p"EQ.spad")
      4> (|pathname| #p"EQ.spad")
      <4 (|pathname| #p"EQ.spad")
    <3 (|pathnameType| "spad")
  <2 (|updateSourceFiles| #p"EQ.spad")
  2> (|namestring| ("/tmp/EQ.spad"))
    3> (|pathname| ("/tmp/EQ.spad"))
    <3 (|pathname| #p"/tmp/EQ.spad")

```

```

<2 (|namestring| "/tmp/EQ.spad")
  Compiling AXIOM source code from file /tmp/EQ.spad using old system
  compiler.

```

Again we find a lot of redundant work. We finally end up calling **compilerDoit** with a constructed argument list:

```

2> (|compilerDoit| NIL (|rq| |lib|))

[compileSpad2Cmd pathname (vol5)]
[compileSpad2Cmd pathnameType (vol5)]
[compileSpad2Cmd namestring (vol5)]
[compileSpad2Cmd updateSourceFiles (vol5)]
[compileSpad2Cmd selectOptionLC (vol5)]
[compileSpad2Cmd terminateSystemCommand (vol5)]
[nequal p??]
[throwKeyedMsg p??]
[compileSpad2Cmd sayKeyedMsg (vol5)]
[error p??]
[strconc p??]
[object2String p??]
[browserAutoloadOnceTrigger p??]
[spad2AsTranslatorAutoloadOnceTrigger p??]
[compilerDoitWithScreenedLisplib p??]
[compilerDoit p478]
[extendLocalLibdb p??]
[spadPrompt p??]
[$newComp p??]
[$scanIfTrue p??]
[$compileOnlyCertainItems p??]
[$f p??]
[$m p??]
[$QuickLet p??]
[$QuickCode p??]
[$sourceFileTypes p??]
[$InteractiveMode p??]
[$options p??]
[$newConlist p??]
[/editfile p??]

```

— defun compileSpad2Cmd —

```

(defun |compileSpad2Cmd| (args)
  (let (|$newComp| |$scanIfTrue|
        |$compileOnlyCertainItems| |$f| |$m| |$QuickLet| |$QuickCode|
        |$sourceFileTypes| |$InteractiveMode| path optlist fun optname
        optargs fullopt constructor)

```

```

(declare (special |$newComp| |$scanIfTrue|
  |$compileOnlyCertainItems| |$f| |$m| |$QuickLet| |$QuickCode|
  |$sourceFileTypes| |$InteractiveMode| /editfile |$options|
  |$newConlist|))
(setq path (|pathname| args))
(cond
  ((nequal (|pathnameType| path) "spad") (|throwKeyedMsg| 's2iz0082 nil))
  ((null (probe-file path))
    (|throwKeyedMsg| 's2il0003 (cons (|namestring| args) nil)))
  (t
    (setq /editfile path)
    (|updateSourceFiles| path)
    (|sayKeyedMsg| 's2iz0038 (list (|namestring| args)))
    (setq optlist '(|break| |constructor| |functions| |library| |lisp|
      |new| |old| |nbreak| |nolibrary| |noquiet| |vartrace| |quiet|
      |translate|))
    (setq |$QuickLet| t)
    (setq |$QuickCode| t)
    (setq fun '(|rq| |lib|))
    (setq |$sourceFileTypes| '("SPAD"))
    (dolist (opt |$options|)
      (setq optname (car opt))
      (setq optargs (cdr opt))
      (setq fullopt (|selectOptionLC| optname optlist nil))
      (case fullopt
        (|old| nil)
        (|library| (setelt fun 1 '|lib|))
        (|nolibrary| (setelt fun 1 '|nolib|))
        (|quiet| (when (nequal (elt fun 0) '|c|) (setelt fun 0 '|rq|)))
        (|noquiet| (when (nequal (elt fun 0) '|c|) (setelt fun 0 '|rf|)))
        (|nbreak| (setq |$scanIfTrue| t))
        (|break| (setq |$scanIfTrue| nil))
        (|vartrace| (setq |$QuickLet| nil))
        (|lisp| (|throwKeyedMsg| 's2iz0036 (list ")lisp")))
        (|functions|
          (if (null optargs)
            (|throwKeyedMsg| 's2iz0037 (list ")functions"))
            (setq |$compileOnlyCertainItems| optargs)))
        (|constructor|
          (if (null optargs)
            (|throwKeyedMsg| 's2iz0037 (list ")constructor"))
            (progn
              (setelt fun 0 '|c|)
              (setq constructor (mapcar #'|unabbrev| optargs))))))
    (t
      (|throwKeyedMsg| 's2iz0036
        (list (strconc ") " (|object2String| optname))))))
  (setq |$InteractiveMode| nil)
  (cond
    (|$compileOnlyCertainItems|

```

```

(if (null constructor)
  (|sayKeyedMsg| 's2iz0040 nil)
  (|compilerDoitWithScreenedLisplib| constructor fun)))
(t (|compilerDoit| constructor fun)))
(|extendLocalLibdb| |$newConlist|)
(|terminateSystemCommand|)
(|spadPrompt|))))))

```

This trivial function cases on the second argument to decide which combination of operations was requested. For this case we see:

```

(1) -> )co EQ
      Compiling AXIOM source code from file /tmp/EQ.spad using old system
      compiler.
1> (|compilerDoit| NIL (|rq| |lib|))
2> (|/RQ,LIB|)

... [snip]...

<2 (|/RQ,LIB| T)
<1 (|compilerDoit| T)
(1) ->

```

10.1.3 defun compilerDoit

```

[compilerDoit /rq (vol5)]
[compilerDoit /rf (vol5)]
[compilerDoit member (vol5)]
[sayBrightly p??]
[opOf p??]
[/RQ,LIB p479]
[$byConstructors p531]
[$constructorsSeen p531]

```

— defun compilerDoit —

```

(defun |compilerDoit| (constructor fun)
  (let (|$byConstructors| |$constructorsSeen|)
    (declare (special |$byConstructors| |$constructorsSeen|))
    (cond
      ((equal fun '(|rf| |lib|)) (|/RQ,LIB|)) ; Ignore "noquiet"
      ((equal fun '(|rf| |nolib|)) (/rf))
      ((equal fun '(|rq| |lib|)) (|/RQ,LIB|))
      ((equal fun '(|rq| |nolib|)) (/rq))
      ((equal fun '(|c| |lib|))

```

```
(setq |$byConstructors| (loop for x in constructor collect (|opOf| x)))
(|/RQ,LIB|)
(dolist (x |$byConstructors|)
  (unless (|member| x |$constructorsSeen|)
    (|sayBrightly| '(">>> Warning " |%b| ,x |%d| " was not found"))))))
```

This function simply calls `/rf-1`.

```
(2) -> )co EQ
      Compiling AXIOM source code from file /tmp/EQ.spad using old system
      compiler.
1> (|compilerDoit| NIL (|rq| |lib|))
2> (|/RQ,LIB|)
3> (|/RF-1 NIL)
...[snip]...
    <3 (|/RF-1 T)
    <2 (|/RQ,LIB| T)
    <1 (|compilerDoit| T)
```

10.1.4 defun `/RQ,LIB`

```
[/rf-1 p480]
[/RQ,LIB echo-meta (vol5)]
[$lisplib p??]
```

— **defun `/RQ,LIB`** —

```
(defun |/RQ,LIB| (&rest foo &aux (echo-meta nil) ($lisplib t))
  (declare (special echo-meta $lisplib) (ignore foo))
  (/rf-1 nil))
```

Since this function is called with nil we fall directly into the call to the function **spad**:

```
(2) -> )co EQ
      Compiling AXIOM source code from file /tmp/EQ.spad using old system
      compiler.
1> (|compilerDoit| NIL (|rq| |lib|))
2> (|/RQ,LIB|)
3> (|/RF-1 NIL)
4> (SPAD "/tmp/EQ.spad")
...[snip]...
    <4 (SPAD T)
```

```

<3 (/RF-1 T)
<2 (|/RQ,LIB| T)
<1 (|compilerDoit| T)

```

10.1.5 defun /rf-1

```

[/rf-1 makeInputFilename (vol5)]
[ncINTERPFILE p527]
[/rf-1 spad (vol5)]
[/editfile p??]
[echo-meta p??]

```

— defun /rf-1 —

```

(defun /rf-1 (ignore)
  (declare (ignore ignore))
  (let* ((input-file (makeInputFilename /editfile))
        (type (pathname-type input-file)))
    (declare (special echo-meta /editfile))
    (cond
      ((string= type "lisp") (load input-file))
      ((string= type "input") (|ncINTERPFILE| input-file echo-meta))
      (t (spad input-file)))))

```

—————

Here we begin the actual compilation process.

```

1> (SPAD "/tmp/EQ.spad")
2> (|makeInitialModemapFrame|)
<2 (|makeInitialModemapFrame| ((NIL)))
2> (INIT-BOOT/SPAD-READER)
<2 (INIT-BOOT/SPAD-READER NIL)
2> (OPEN "/tmp/EQ.spad" :DIRECTION :INPUT)
<2 (OPEN #<input stream "/tmp/EQ.spad">)
2> (INITIALIZE-PREPARSE #<input stream "/tmp/EQ.spad">)
<2 (INITIALIZE-PREPARSE ")abbrev domain EQ Equation")
2> (PREPARSE #<input stream "/tmp/EQ.spad">)
EQ abbreviates domain Equation
<2 (PREPARSE (# # # # # # # ...))
2> (|PARSE-NewExpr|)
<2 (|PARSE-NewExpr| T)
2> (S-PROCESS (|where| # #))
...[snip]...
3> (OPEN "/tmp/EQ.erlib/info" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.erlib/info">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.lsp")

```

```

<3 (OPEN #<input stream "/tmp/EQ.nrllib/EQ.lsp">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.data" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.data">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.c" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.c">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.h" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.h">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.fn" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.fn">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.o" :DIRECTION :OUTPUT :IF-EXISTS :APPEND)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.o">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.data")
<3 (OPEN #<input stream "/tmp/EQ.nrllib/EQ.data">)
3> (OPEN #p"/tmp/EQ.nrllib/index.kaf")
<3 (OPEN #<input stream "/tmp/EQ.nrllib/index.kaf">)
<2 (S-PROCESS NIL)
<1 (SPAD T)
1> (OPEN "temp.text" :DIRECTION :OUTPUT)
<1 (OPEN #<output stream "temp.text">)
1> (OPEN "libdb.text")
<1 (OPEN #<input stream "libdb.text">)
1> (OPEN "temp.text")
<1 (OPEN #<input stream "temp.text">)
1> (OPEN "libdb.text" :DIRECTION :OUTPUT)
<1 (OPEN #<output stream "libdb.text">)

```

The major steps in this process involve the **preparse** function. (See book volume 5 for more details). The **preparse** function returns a list of pairs of the form: ((linenumber . linestring) (linenumber . linestring)) For instance, for the file `EQ.spad`, we get:

```

<2 (PREPARSE (
(19 . "Equation(S: Type): public == private where")
(20 . " (Ex ==> OutputForm;")
(21 . "   public ==> Type with")
(22 . "   (\\"=\\" : (S, S) -> $;")
...[skip]...
(202 . "      inv eq == [inv lhs eq, inv rhs eq]);")
(203 . "      if S has ExpressionSpace then")
(204 . "          subst(eq1,eq2) ==")
(205 . "              (eq3 := eq2 pretend Equation S;")
(206 . "                  [subst(lhs eq1,eq3),subst(rhs eq1,eq3)])))))

```

And the **s-process** function which returns a parsed version of the input.

```

2> (S-PROCESS
(|where|
(== (|:| (|Equation| (|:| S |Type|)) |public|) |private|)
(|:|
(|:|
(|:|

```



```

(==> |Ex| |OutputForm|)
(==> |public|
  (|Join| |Type|
    (|with|
      (CATEGORY
        (|Signature| "=" (-> (|,| S S) $))
        (|Signature| |equation| (-> (|,| S S) $))
        (|Signature| |swap| (-> $ $))
        (|Signature| |lhs| (-> $ S))
        (|Signature| |rhs| (-> $ S))
        (|Signature| |map| (-> (|,| (-> S S) $) $))
        (|if| (|has| S (|InnerEvalable| (|,| |Symbol| S)))
          (|Attribute| (|InnerEvalable| (|,| |Symbol| S)))
          NIL)
        (|if| (|has| S |SetCategory|)
          (CATEGORY
            (|Attribute| |SetCategory|)
            (|Attribute| (|CoercibleTo| |Boolean|))
            (|if| (|has| S (|Evalable| S))
              (CATEGORY
                (|Signature| |eval| (-> (|,| $ $) $))
                (|Signature| |eval| (-> (|,| $ (|List| $)) $)))
              NIL))
            NIL)
          (|if| (|has| S |AbelianSemiGroup|)
            (CATEGORY
              (|Attribute| |AbelianSemiGroup|)
              (|Signature| "+" (-> (|,| S $) $))
              (|Signature| "+" (-> (|,| $ S) $)))
              NIL)
            (|if| (|has| S |AbelianGroup|)
              (CATEGORY
                (|Attribute| |AbelianGroup|)
                (|Signature| |leftZero| (-> $ $))
                (|Signature| |rightZero| (-> $ $))
                (|Signature| "-" (-> (|,| S $) $))
                (|Signature| "-" (-> (|,| $ S) $)))
                NIL)
              (|if| (|has| S |SemiGroup|)
                (CATEGORY
                  (|Attribute| |SemiGroup|)
                  (|Signature| "*" (-> (|,| S $) $))
                  (|Signature| "*" (-> (|,| $ S) $)))
                  NIL)
                (|if| (|has| S |Monoid|)
                  (CATEGORY
                    (|Attribute| |Monoid|)
                    (|Signature| |leftOne| (-> $ (|Union| (|,| $ "failed"))))
                    (|Signature| |rightOne| (-> $ (|Union| (|,| $ "failed")))))
                    NIL)
                  (|if| (|has| S |Group|)

```



```

(|;|
  (|;|
    (|;|
      (|:=| |Rep|
        (|Record| (|,| (|:| |lhs| S) (|:| |rhs| S))))
      (|,| |eq1| (|:| |eq2| $)))
    (|:| |s| S))
  (|if| (|has| S |IntegralDomain|)
    (==
      (|factorAndSplit| |eq1|)
      (|;|
        (=> (|has| S (|:| |factor| (-> S (|Factored| S))))
        (|;|
          (|:=| |eq0| (|rightZero| |eq1|))
          (COLLECT
            (IN |rcf| (|factors| (|factor| (|lhs| |eq0|))))
            (|construct|
              (|equation| (|,| (|rcf| |factor|) 0))))))
          (|construct| |eq1|))
        NIL))
    (==
      (= (|:| |l| S) (|:| |r| S))
      (|construct| (|,| |l| |r|)))
    (==
      (|equation| (|,| |l| |r|))
      (|construct| (|,| |l| |r|)))
      (== (|lhs| |eqn|) (|eqn| |lhs|))
      (== (|rhs| |eqn|) (|eqn| |rhs|))
      (==
        (|swap| |eqn|)
        (|construct| (|,| (|rhs| |eqn|) (|lhs| |eqn|))))
      (==
        (|map| (|,| |fn| |eqn|))
        (|equation|
          (|,| (|fn| (|eqn| |lhs|)) (|fn| (|eqn| |rhs|)))))
      (|if| (|has| S (|InnerEvalable| (|,| |Symbol| S)))
        (|;|
          (|;|
            (|;|
              (|:| (|:| |s| |Symbol|) (|:| |ls| (|List| |Symbol|)))
              (|:| |x| S))
              (|:| |lx| (|List| S)))
            (==
              (|eval| (|,| (|,| |eqn| |s|) |x|))
              (=
                (|eval| (|,| (|,| (|eqn| |lhs|) |s|) |x|))
                (|eval| (|,| (|,| (|eqn| |rhs|) |s|) |x|))))
              (==
                (|eval| (|,| (|,| |eqn| |ls|) |lx|))

```

```

(=
  (|eval| (|,| (|,| (|eqn| |lhs|) |ls|) |lx|))
  (|eval| (|,| (|,| (|eqn| |rhs|) |ls|) |lx|))))
NIL))
(|if| (|has| S (|Evalable| S))
(|;|
(==
  (|:| (|eval| (|,| (|:| |eqn1| $) (|:| |eqn2| $))) $)
  (=
    (|eval|
      (|,| (|eqn1| |lhs|) (|pretend| |eqn2| (|Equation| S))))
    (|eval|
      (|,| (|eqn1| |rhs|) (|pretend| |eqn2| (|Equation| S))))))
  (==
    (|:|
      (|eval| (|,| (|:| |eqn1| $) (|:| |eqn2| (|List| $)))) $)
    (=
      (|eval|
        (|,|
          (|eqn1| |lhs|)
          (|pretend| |eqn2| (|List| (|Equation| S))))
        (|eval|
          (|,|
            (|eqn1| |rhs|)
            (|pretend| |eqn2| (|List| (|Equation| S)))))))
    NIL))
(|if| (|has| S |SetCategory|)
(|;|
(|;|
(==
  (= |eq1| |eq2|)
  (|and|
    (@ (= (|eq1| |lhs|) (|eq2| |lhs|)) |Boolean|)
    (@ (= (|eq1| |rhs|) (|eq2| |rhs|)) |Boolean|)))
  (==
    (|:| (|coerce| (|:| |eqn| $)) |Ex|)
    (= (|::| (|eqn| |lhs|) |Ex|) (|::| (|eqn| |rhs|) |Ex|))))
  (==
    (|:| (|coerce| (|:| |eqn| $)) |Boolean|)
    (= (|eqn| |lhs|) (|eqn| |rhs|))))
  NIL))
(|if| (|has| S |AbelianSemiGroup|)
(|;|
(|;|
(==
  (+ |eq1| |eq2|)
  (=
    (+ (|eq1| |lhs|) (|eq2| |lhs|))
    (+ (|eq1| |rhs|) (|eq2| |rhs|))))
  (== (+ |s| |eq2|) (+ (|construct| (|,| |s| |s|)) |eq2|)))

```

```

      (== (+ |eq1| |s|) (+ |eq1| (|construct| (|,| |s| |s|))))
    NIL))
  (|if| (|has| S |AbelianGroup|)
    (|;|
      (|;|
        (|;|
          (|;|
            (|;|
              (== (- |eq1|) (= (- (|lhs| |eq1|)) (- (|rhs| |eq1|))))
              (== (- |s| |eq2|) (- (|construct| (|,| |s| |s|)) |eq2|)))
              (== (- |eq1| |s|) (- |eq1| (|construct| (|,| |s| |s|))))
              (== (|leftZero| |eq1|) (= 0 (- (|rhs| |eq1|) (|lhs| |eq1|))))
              (== (|rightZero| |eq1|) (= (- (|lhs| |eq1|) (|rhs| |eq1|)) 0)))
              (== 0 (|equation| (|,| (|elt| S 0) (|elt| S 0))))
            (==
              (- |eq1| |eq2|)
              (=
                (- (|eq1| |lhs|) (|eq2| |lhs|))
                (- (|eq1| |rhs|) (|eq2| |rhs|))))
            NIL))
    (|if| (|has| S |SemiGroup|)
      (|;|
        (|;|
          (|;|
            (==
              (* (|:| |eq1| $) (|:| |eq2| $))
              (=
                (* (|eq1| |lhs|) (|eq2| |lhs|))
                (* (|eq1| |rhs|) (|eq2| |rhs|))))
            (==
              (* (|:| |l| S) (|:| |eqn| $))
              (= (* |l| (|eqn| |lhs|)) (* |l| (|eqn| |rhs|))))
            (==
              (* (|:| |l| S) (|:| |eqn| $))
              (= (* |l| (|eqn| |lhs|)) (* |l| (|eqn| |rhs|))))
            (==
              (* (|:| |eqn| $) (|:| |l| S))
              (= (* (|eqn| |lhs|) |l|) (* (|eqn| |rhs|) |l|)))
            NIL))
      (|if| (|has| S |Monoid|)
        (|;|
          (|;|
            (|;|
              (== 1 (|equation| (|,| (|elt| S 1) (|elt| S 1))))
              (==
                (|recip| |eq1|)
                (|;|
                  (|;|
                    (=> (|case| (|:=| |lh| (|recip| (|lhs| |eq1|))) "failed"))

```

```

        "failed")
      (=> (|case| (|:=| |rh| (|recip| (|rhs| |eq|))) "failed")
        "failed"))
    (|construct| (|,| (|::| |lh| S) (|::| |rh| S))))))
  (==
    (|leftOne| |eq|)
    (|;|
      (=> (|case| (|:=| |re| (|recip| (|lhs| |eq|))) "failed")
        "failed")
      (= 1 (* (|rhs| |eq|) |re|))))))
  (==
    (|rightOne| |eq|)
    (|;|
      (=> (|case| (|:=| |re| (|recip| (|rhs| |eq|))) "failed")
        "failed")
      (= (* (|lhs| |eq|) |re| 1))))
  NIL))
(|if| (|has| S |Group|)
  (|;|
    (|;|
      (==
        (|inv| |eq|)
        (|construct| (|,| (|inv| (|lhs| |eq|)) (|inv| (|rhs| |eq|))))
        (== (|leftOne| |eq|) (= 1 (* (|rhs| |eq|) (|inv| (|rhs| |eq|)))))
        (== (|rightOne| |eq|) (= (* (|lhs| |eq|) (|inv| (|rhs| |eq|)) 1)))
        NIL))
    (|if| (|has| S |Ring|)
      (|;|
        (==
          (|characteristic| (|@Tuple|))
          ((|elt| S |characteristic|) (|@Tuple|)))
          (== (* (|::| |i| |Integer|) (|::| |eq| $)) (* (|::| |i| S) |eq|)))
          NIL))
      (|if| (|has| S |IntegralDomain|)
        (==
          (|factorAndSplit| |eq|)
          (|;|
            (|;|
              (=>
                (|has| S (|::| |factor| (-> S (|Factored| S))))
                (|;|
                  (|:=| |eq0| (|rightZero| |eq|))
                  (COLLECT
                    (IN |rcf| (|factors| (|factor| (|lhs| |eq0|))))
                    (|construct| (|equation| (|,| (|rcf| |factor|) 0))))))
                (=>
                  (|has| S (|Polynomial| |Integer|))
                  (|;|
                    (|;|
                      (|;|

```

```

(|:=| |eq0| (|rightZero| |eq|))
(==> MF
  (|MultivariateFactorize|
    (|,|
      (|,| (|,| |Symbol| (|IndexedExponents| |Symbol|)) |Integer|)
      (|Polynomial| |Integer|))))
(|:=|
  (|:| |p| (|Polynomial| |Integer|))
  (|pretend| (|lhs| |eq0|) (|Polynomial| |Integer|)))
(COLLECT
  (IN |rcf| (|factors| ((|elt| MF |factor|) |p|)))
  (|construct|
    (|equation| (|,| (|pretend| (|rcf| |factor|) S) 0))))))
(|construct| |eq|))
NIL))
(|if| (|has| S (|PartialDifferentialRing| |Symbol|))
  (==
    (|:| (|differentiate| (|,| (|:| |eq| $) (|:| |sym| |Symbol|))) $)
    (|construct|
      (|,|
        (|differentiate| (|,| (|lhs| |eq|) |sym|))
        (|differentiate| (|,| (|rhs| |eq|) |sym|))))))
    NIL))
(|if| (|has| S |Field|)
  (|;|
    (|;|
      (== (|dimension| (|@Tuple|)) (|::| 2 |CardinalNumber|))
      (==
        (/ (|:| |eq1| $) (|:| |eq2| $))
        (= (/ (|eq1| |lhs|) (|eq2| |lhs|)) (/ (|eq1| |rhs|) (|eq2| |rhs|))))))
      (==
        (|inv| |eq|)
        (|construct| (|,| (|inv| (|lhs| |eq|)) (|inv| (|rhs| |eq|))))))
      NIL))
(|if| (|has| S |ExpressionSpace|)
  (==
    (|subst| (|,| |eq1| |eq2|))
    (|;|
      (|:=| |eq3| (|pretend| |eq2| (|Equation| S)))
      (|construct|
        (|,|
          (|subst| (|,| (|lhs| |eq1|) |eq3|))
          (|subst| (|,| (|rhs| |eq1|) |eq3|))))))
    NIL))))))

```

10.1.6 defun spad

```
[spad-reader p??]
[spad addBinding (vol5)]
[spad makeInitialModemapFrame (vol5)]
[spad init-boot/spad-reader (vol5)]
[initialize-prepare p73]
[prepare p76]
[PARSE-NewExpr p403]
[pop-stack-1 p462]
[s-process p490]
[ioclear p??]
[spad shut (vol5)]
[$noSubsumption p??]
[$InteractiveFrame p??]
[$InitialDomainsInScope p??]
[$InteractiveMode p??]
[$spad p489]
[$boot p??]
[curoutstream p??]
[*fileactq-apply* p??]
[line p535]
[optionlist p??]
[echo-meta p??]
[/editfile p??]
[*comp370-apply* p??]
[*eof* p??]
[file-closed p??]
[boot-line-stack p??]
[spad-reader p??]
```

— defun spad —

```
(defun spad (&optional (*spad-input-file* nil) (*spad-output-file* nil)
  &aux (*comp370-apply* #'print-defun)
        (*fileactq-apply* #'print-defun)
        ($spad t) ($boot nil) (optionlist nil) (*eof* nil)
        (file-closed nil) (/editfile *spad-input-file*)
        (|$noSubsumption| |$noSubsumption|) in-stream out-stream)
  (declare (special echo-meta /editfile *comp370-apply* *eof* curoutstream
    file-closed |$noSubsumption| |$InteractiveFrame|
    |$InteractiveMode| optionlist
    boot-line-stack *fileactq-apply* $spad $boot))
  ;; only rebind |$InteractiveFrame| if compiling
  (progv (if (not |$InteractiveMode|) '(|$InteractiveFrame|))
    (if (not |$InteractiveMode|)
      (list (|addBinding| '|$DomainsInScope|
```



```

        '((fluid . |true|))
        (|addBinding| '|$Information| nil
          (|makeInitialModemapFrame|))))))
(init-boot/spad-reader)
(unwind-protect
  (progn
    (setq in-stream (if *spad-input-file*
                        (open *spad-input-file* :direction :input)
                        *standard-input*))
    (initialize-prepare in-stream)
    (setq out-stream (if *spad-output-file*
                        (open *spad-output-file* :direction :output)
                        *standard-output*))
    (when *spad-output-file*
      (format out-stream "~&,,, -- Mode:Lisp; Package:Boot  --~%~%")
      (print-package "BOOT"))
    (setq curoutstream out-stream)
    (loop
      (if (or *eof* file-closed) (return nil))
      (catch 'spad_reader
        (if (setq boot-line-stack (prepare in-stream))
          (let ((line (cdar boot-line-stack)))
            (declare (special line))
            (|PARSE-NewExpr|)
            (let ((parseout (pop-stack-1)) )
              (when parseout
                (let ((*standard-output* out-stream))
                  (s-process parseout))
                (format out-stream "~&"))))
          )))
    (ioclear in-stream out-stream)))
    (if *spad-input-file* (shut in-stream))
    (if *spad-output-file* (shut out-stream)))
  t))

```

10.1.7 defun Interpreter interface to the compiler

```

[curstrm p??]
[def-rename p493]
[new2OldLisp p458]
[parseTransform p93]
[postTransform p351]
[displayPreCompilationErrors p456]
[prettyprint p??]
[s-process processInteractive (vol5)]

```

```

[compTopLevel p494]
[def-process p??]
[displaySemanticErrors p??]
[terpri p??]
[get-internal-run-time p??]
[$Index p??]
[$macroassoc p??]
[$newspad p??]
[$PolyMode p??]
[$EmptyMode p131]
[$compUniquelyIfTrue p??]
[$currentFunction p??]
[$postStack p??]
[$topOp p??]
[$semanticErrorStack p??]
[$warningStack p??]
[$exitMode p??]
[$exitModeStack p??]
[$returnMode p??]
[$leaveMode p??]
[$leaveLevelStack p??]
[$top-level p??]
[$insideFunctorIfTrue p??]
[$insideExpressionIfTrue p??]
[$insideCoerceInteractiveHardIfTrue p??]
[$insideWhereIfTrue p??]
[$insideCategoryIfTrue p??]
[$insideCapsuleFunctionIfTrue p??]
[$form p??]
[$DomainFrame p??]
[$e p??]
[$EmptyEnvironment p??]
[$genFVar p??]
[$genSDVar p??]
[$VariableCount p??]
[$previousTime p??]
[$LocalFrame p??]
[$Translation p??]
[$TranslateOnly p??]
[$PrintOnly p??]
[$currentLine p??]
[$InteractiveFrame p??]
[curoutstream p??]

```

— **defun s-process** —

```

(defun s-process (x)
  (prog ((|Index| 0)
        ($macroassoc ())
        ($newspad t)
        (|$PolyMode| |$EmptyMode|)
        (|$compUniquelyIfTrue| nil)
        |$currentFunction|
        (|$postStack| nil)
        |$topOp|
        (|$semanticErrorStack| ())
        (|$warningStack| ())
        (|$exitMode| |$EmptyMode|)
        (|$exitModeStack| ())
        (|$returnMode| |$EmptyMode|)
        (|$leaveMode| |$EmptyMode|)
        (|$leaveLevelStack| ())
        $top_level |$insideFunctorIfTrue| |$insideExpressionIfTrue|
        |$insideCoerceInteractiveHardIfTrue| |$insideWhereIfTrue|
        |$insideCategoryIfTrue| |$insideCapsuleFunctionIfTrue| |$form|
        (|$DomainFrame| '((NIL)))
        (|$e| |$EmptyEnvironment|)
        (|$genFVar| 0)
        (|$genSDVar| 0)
        (|$VariableCount| 0)
        (|$previousTime| (get-internal-run-time))
        (|$LocalFrame| '((NIL)))
        (curstrm curoutstream) |$s| |$x| |$m| u)
    (declare (special |$Index| $macroassoc $newspad |$PolyMode| |$EmptyMode|
                     |$compUniquelyIfTrue| |$currentFunction| |$postStack| |$topOp|
                     |$semanticErrorStack| |$warningStack| |$exitMode| |$exitModeStack|
                     |$returnMode| |$leaveMode| |$leaveLevelStack| $top_level
                     |$insideFunctorIfTrue| |$insideExpressionIfTrue| | | | | | |
                     |$insideCoerceInteractiveHardIfTrue| |$insideWhereIfTrue|
                     |$insideCategoryIfTrue| |$insideCapsuleFunctionIfTrue| |$form|
                     |$DomainFrame| |$e| |$EmptyEnvironment| |$genFVar| |$genSDVar|
                     |$VariableCount| |$previousTime| |$LocalFrame|
                     curstrm |$s| |$x| |$m| curoutstream $traceflag |$Translation|
                     |$TranslateOnly| |$PrintOnly| |$currentLine| |$InteractiveFrame|))
      (setq $traceflag t)
      (if (not x) (return nil))
      (if $boot
        (setq x (def-rename (|new2OldLisp| x)))
        (setq x (|parseTransform| (|postTransform| x))))
      (when |$TranslateOnly| (return (setq |$Translation| x)))
      (when |$postStack| (|displayPreCompilationErrors|) (return nil))
      (when |$PrintOnly|
        (format t "~S =====>~%" |$currentLine|)
        (return (prettyprint x)))
      (if (not $boot)
        (if |$InteractiveMode|

```

```

(|processInteractive| x nil)
  (when (setq u (|compTopLevel| x |$EmptyMode| |$InteractiveFrame|))
    (setq |$InteractiveFrame| (third u)))
  (def-process x))
(when |$semanticErrorStack| (|displaySemanticErrors|))
(terpri))

```

10.1.8 defun print-defun

```

[is-console p467]
[print-full p??]
[vmisp::optionlist p??]
[$PrettyPrint p??]

```

— defun print-defun —

```

(defun print-defun (name body)
  (let* ((sp (assoc 'vmisp::compiler-output-stream vmisp::optionlist))
        (st (if sp (cdr sp) *standard-output*)))
    (declare (special vmisp::optionlist |$PrettyPrint|))
    (when (and (is-console st) (symbolp name) (fboundp name)
              (not (compiled-function-p (symbol-function name)))))
      (compile name))
    (when (or |$PrettyPrint| (not (is-console st)))
      (print-full body st) (force-output st))))

```

10.1.9 defun def-rename

```

[def-rename1 p494]

```

— defun def-rename —

```

(defun def-rename (x)
  (def-rename1 x))

```

10.1.10 defun def-rename1

[def-rename1 p494]

— defun def-rename1 —

```
(defun def-rename1 (x)
  (cond
    ((symbolp x)
     (let ((y (get x 'rename))) (if y (first y) x)))
    ((and (listp x) x)
     (if (eqcar x 'quote)
         x
         (cons (def-rename1 (first x)) (def-rename1 (cdr x)))))
    (x)))
```

—————

10.1.11 defun compTopLevel

[newComp p??]
 [compOrCroak p496]
 [\$NRTderivedTargetIfTrue p??]
 [\$killOptimizeIfTrue p??]
 [\$forceAdd p??]
 [\$compTimeSum p??]
 [\$resolveTimeSum p??]
 [\$packagesUsed p??]
 [\$envHashTable p??]

— defun compTopLevel —

```
(defun |compTopLevel| (form mode env)
  (let (|$NRTderivedTargetIfTrue| |$killOptimizeIfTrue| |$forceAdd|
        |$compTimeSum| |$resolveTimeSum| |$packagesUsed| |$envHashTable|
        t1 t2 t3 val newmode)
    (declare (special |$NRTderivedTargetIfTrue| |$killOptimizeIfTrue|
                      |$forceAdd| |$compTimeSum| |$resolveTimeSum|
                      |$packagesUsed| |$envHashTable| ))
    (setq |$NRTderivedTargetIfTrue| nil)
    (setq |$killOptimizeIfTrue| nil)
    (setq |$forceAdd| nil)
    (setq |$compTimeSum| 0)
    (setq |$resolveTimeSum| 0)
    (setq |$packagesUsed| NIL)
    (setq |$envHashTable| (make-hashtable 'equal)))
```

```

(dolist (u (car (car env)))
  (dolist (v (cdr u))
    (hput |$envHashTable| (cons (car u) (cons (car v) nil)) t)))
(cond
  ((or (and (consp form) (eq (qfirst form) 'def))
       (and (consp form) (eq (qfirst form) '|where|)
            (progn
              (setq t1 (qrest form))
              (and (consp t1)
                    (progn
                     (setq t2 (qfirst t1))
                     (and (consp t2) (eq (qfirst t2) 'def)))))))
    (setq t3 (|compOrCroak| form mode env))
    (setq val (car t3))
    (setq newmode (second t3))
    (cons val (cons newmode (cons env nil))))
  (t (|compOrCroak| form mode env))))

```

Given:

CohenCategory(): Category == SetCategory with

```

kind:(CExpr)->Boolean
operand:(CExpr,Integer)->CExpr
numberOfOperand:(CExpr)->Integer
construct:(CExpr,CExpr)->CExpr

```

the resulting call looks like:

```

(|compOrCroak|
  (DEF (|CohenCategory|)
    ((|Category|)
     (NIL)
     (|Join|
      (|SetCategory|)
      (CATEGORY |package|
        (SIGNATURE |kind| ((|Boolean|) |CExpr|))
        (SIGNATURE |operand| (|CExpr| |CExpr| (|Integer|)))
        (SIGNATURE |numberOfOperand| ((|Integer|) |CExpr|))
        (SIGNATURE |construct| (|CExpr| |CExpr| |CExpr|))))
    |$EmptyMode|
    (((
      (|$DomainsInScope|
       (FLUID . |true|)
       (special |$EmptyMode| |$NoValueMode|))))))

```

This compiler call expects the first argument `x` to be a DEF form to compile, The second argument, `m`, is the mode. The third argument, `e`, is the environment.

10.1.12 defun compOrCroak

[compOrCroak1 p496]

— defun compOrCroak —

```
(defun |compOrCroak| (form mode env)
  (|compOrCroak1| form mode env nil nil))
```

—————

This results in a call to the inner function with

```
(|compOrCroak1|
  (DEF (|CohenCategory|)
    ((|Category|))
    (NIL)
    (|Join|
      (|SetCategory|)
      (CATEGORY |package|
        (SIGNATURE |kind| ((|Boolean|) |CEpr|))
        (SIGNATURE |operand| (|CEpr| |CEpr| (|Integer|)))
        (SIGNATURE |numberOfOperand| ((|Integer|) |CEpr|))
        (SIGNATURE |construct| (|CEpr| |CEpr| |CEpr|))))
    |$EmptyMode|
    (((
      |$DomainsInScope|
      (FLUID . |true|)
      (special |$EmptyMode| |$NoValueMode|))))
    NIL
    NIL
    |comp|)
```

The inner function augments the environment with information from the compiler stack `$compStack` and `$compErrorMessageStack`. Note that these variables are passed in the argument list so they get preserved on the call stack. The calling function gets called for every inner form so we use this implicit stacking to retain the information.

10.1.13 defun compOrCroak1

[comp p497]

[compOrCroak1,compactify p527]

[stackSemanticError p??]

```

[mkErrorExpr p??]
[displaySemanticErrors p??]
[say p??]
[displayComp p??]
[userError p??]
[$compStack p??]
[$compErrorMessageStack p??]
[$level p??]
[$s p??]
[$scanIfTrue p??]
[$exitModeStack p??]
[compOrCroak p496]

```

— defun compOrCroak1 —

```

(defun |compOrCroak1| (form mode env |$compStack| |$compErrorMessageStack|)
  (declare (special |$compStack| |$compErrorMessageStack|))
  (let (td errorMessage)
    (declare (special |$level| |$s| |$scanIfTrue| |$exitModeStack|))
    (cond
      ((setq td (catch '|compOrCroak| (|comp| form mode env))) td)
      (t
       (setq |$compStack|
              (cons (list form mode env |$exitModeStack|) |$compStack|))
       (setq |$s| (|compOrCroak1,compactify| |$compStack|))
       (setq |$level| (|#| |$s|))
       (setq errorMessage
              (if |$compErrorMessageStack|
                  (car |$compErrorMessageStack|)
                  '|unspecified error|))
       (cond
         (|$scanIfTrue|
          (|stackSemanticError| errorMessage (|mkErrorExpr| |$level|))
          (list '|failedCompilation| mode env ))
         (t
          (|displaySemanticErrors|)
          (say "***** comp fails at level " |$level| " with expression: *****")
          (|displayComp| |$level|)
          (|userError| errorMessage))))))

```

10.1.14 defun comp

```

[compNoStacking p498]
[$compStack p??]

```


[\$exitModeStack p??]

— **defun comp** —

```
(defun |comp| (form mode env)
  (let (td)
    (declare (special |$compStack| |$exitModeStack|))
    (if (setq td (|compNoStacking| form mode env))
        (setq |$compStack| nil)
        (push (list form mode env |$exitModeStack|) |$compStack|))
    td))
```

—————

10.1.15 **defun compNoStacking**

\$Representation is bound in compDefineFunctor, set by doIt. This hack says that when something is undeclared, \$ is preferred to the underlying representation – RDJ 9/12/83

```
[comp2 p499]
[compNoStacking1 p498]
[$compStack p??]
[$Representation p??]
[$EmptyMode p131]
```

— **defun compNoStacking** —

```
(defun |compNoStacking| (form mode env)
  (let (td)
    (declare (special |$compStack| |$Representation| |$EmptyMode|))
    (if (setq td (|comp2| form mode env))
        (if (and (equal mode |$EmptyMode|) (equal (second td) |$Representation|))
            (list (car td) '$ (third td))
            td)
        (|compNoStacking1| form mode env |$compStack|))))
```

—————

10.1.16 **defun compNoStacking1**

```
[get p??]
[comp2 p499]
[$compStack p??]
```

— **defun compNoStacking1** —

```
(defun |compNoStacking1| (form mode env |$compStack|)
  (declare (special |$compStack|))
  (let (u td)
    (if (setq u (|get| (if (eq mode '$) '|Rep| mode) '|value| env))
      (if (setq td (|comp2| form (car u) env))
          (list (car td) mode (third td))
          nil)
      nil)))
```

10.1.17 defun comp2

```
[comp3 p500]
[isDomainForm p329]
[isFunctor p229]
[insert p??]
[opOf p??]
[nequal p??]
[addDomain p228]
[$bootStrapMode p??]
[$packagesUsed p??]
[$lisplib p??]
```

— defun comp2 —

```
(defun |comp2| (form mode env)
  (let (tmp1)
    (declare (special |$bootStrapMode| |$packagesUsed| $lisplib))
    (when (setq tmp1 (|comp3| form mode env))
      (destructuring-bind (y mprime env) tmp1
        (when (and $lisplib (|isDomainForm| form env) (|isFunctor| form))
          (setq |$packagesUsed| (|insert| (list (|opOf| form)) |$packagesUsed|)))
        ; isDomainForm test needed to prevent error while compiling Ring
        ; $bootStrapMode-test necessary for compiling Ring in $bootStrapMode
        (if (and (nequal mode mprime)
                  (or |$bootStrapMode| (|isDomainForm| mprime env)))
            (list y mprime (|addDomain| mprime env))
            (list y mprime env))))))
```

10.1.18 defun comp3

```

[addDomain p228]
[compWithMappingMode p517]
[compAtom p503]
[getmode p??]
[applyMapping p??]
[compApply p??]
[compColon p267]
[compCoerce p343]
[stringPrefix? p??]
[comp3 pname (vol5)]
[compTypeOf p502]
[compExpression p507]
[comp3 member (vol5)]
[getDomainsInScope p230]
[$e p??]
[$insideCompTypeOf p??]

```

— **defun comp3** —

```

(defun |comp3| (form mode |$e|)
  (declare (special |$e|))
  (let (env a op ml u sig varlist tmp3 body tt xprime tmp1 mprime tmp2 eprime)
    (declare (special |$insideCompTypeOf|))
    (setq |$e| (|addDomain| mode |$e|))
    (setq env |$e|)
    (cond
      ((and (consp mode) (eq (qfirst mode) '|Mapping|))
        (|compWithMappingMode| form mode env))
      ((and (consp mode) (eq (qfirst mode) 'quote)
        (progn
          (setq tmp1 (qrest mode))
          (and (consp tmp1) (eq (qrest tmp1) nil)
            (progn (setq a (qfirst tmp1)) t))))
        (when (equal form a) (list form mode |$e|)))
      ((stringp mode)
        (when (and (atom form)
          (or (equal mode form) (equal mode (princ-to-string form))))
          (list mode mode env )))
      ((or (null form) (atom form)) (|compAtom| form mode env))
      (t
        (setq op (car form))
        (cond
          ((and (progn
            (setq tmp1 (|getmode| op env))
            (and (consp tmp1)
              (eq (qfirst tmp1) '|Mapping|)

```

```

        (progn (setq ml (qrest tmp1)) t)))
      (setq u (|applyMapping| form mode env ml)))
    u)
  ((and (consp op) (eq (qfirst op) 'kappa)
    (progn
      (setq tmp1 (qrest op))
      (and (consp tmp1)
        (progn
          (setq sig (qfirst tmp1))
          (setq tmp2 (qrest tmp1))
          (and (consp tmp2)
            (progn
              (setq varlist (qfirst tmp2))
              (setq tmp3 (qrest tmp2))
              (and (consp tmp3)
                (eq (qrest tmp3) nil)
                (progn
                  (setq body (qfirst tmp3))
                  t))))))))
      (|compApply| sig varlist body (cdr form) mode env))
    ((eq op '|:|) (|compColon| form mode env))
    ((eq op '|::|) (|compCoerce| form mode env))
    ((and (null (eq |$insideCompTypeOf| t))
      (|stringPrefix?| "TypeOf" (pname op)))
      (|compTypeOf| form mode env))
    (t
      (setq tt (|compExpression| form mode env))
      (cond
        ((and (consp tt)
          (progn
            (setq xprime (qfirst tt))
            (setq tmp1 (qrest tt))
            (and (consp tmp1)
              (progn
                (setq mprime (qfirst tmp1))
                (setq tmp2 (qrest tmp1))
                (and (consp tmp2)
                  (eq (qrest tmp2) nil)
                  (progn
                    (setq eprime (qfirst tmp2))
                    t))))
              (null (|member| mprime (|getDomainsInScope| eprime))))
            (list xprime mprime (|addDomain| mprime eprime)))
          (t tt)))))))))

```

10.1.19 defun compTypeOf

```
[eqsubstlist p??]
[get p??]
[put p??]
[comp3 p500]
[$insideCompTypeOf p??]
[$FormalMapVariableList p242]
```

— defun compTypeOf —

```
(defun |compTypeOf| (form mode env)
  (let (|$insideCompTypeOf| op arg1 newModemap)
    (declare (special |$insideCompTypeOf| |$FormalMapVariableList|))
    (setq op (car form))
    (setq arg1 (cdr form))
    (setq |$insideCompTypeOf| t)
    (setq newModemap
      (eqsubstlist arg1 |$FormalMapVariableList| (|get| op '|modemap| env)))
    (setq env (|put| op '|modemap| newModemap env))
    (|comp3| form mode env)))
```

10.1.20 defun compColonInside

```
[addDomain p228]
[comp p497]
[coerce p337]
[stackWarning p??]
[opOf p??]
[stackSemanticError p??]
[$newCompilerUnionFlag p??]
[$EmptyMode p131]
```

— defun compColonInside —

```
(defun |compColonInside| (form mode env mprime)
  (let (mpp warningMessage td tprime)
    (declare (special |$newCompilerUnionFlag| |$EmptyMode|))
    (setq env (|addDomain| mprime env))
    (when (setq td (|comp| form |$EmptyMode| env))
      (cond
        ((equal (setq mpp (second td)) mprime)
         (setq warningMessage
           (list '|| mprime '| -- should replace by @|))))
```

```

(setq td (list (car td) mprime (third td)))
(when (setq tprime (|coerce| td mode))
  (cond
    (warningMessage (|stackWarning| warningMessage))
    ((and (|newCompilerUnionFlag| (eq (|opOf| mpp) '|Union|))
      (setq tprime
        (|stackSemanticError|
          (list '|cannot pretend | form '| of mode | mpp '| to mode | mprime )
            nil))))
    (t
      (|stackWarning|
        (list '|:| mprime '| -- should replace by pretend|))))
  tprime)))

```

10.1.21 defun compAtom

```

[compAtomWithModemap p??]
[get p??]
[modeIsAggregateOf p??]
[compList p506]
[compVector p335]
[convert p504]
[isSymbol p??]
[compSymbol p505]
[primitiveType p504]
[primitiveType p504]
[$Expression p??]

```

— defun compAtom —

```

(defun |compAtom| (form mode env)
  (prog (tmp1 tmp2 r td tt)
    (declare (special |$Expression|))
    (return
      (cond
        ((setq td
          (|compAtomWithModemap| form mode env (|get| form '|modemap| env))) td)
        ((eq form '|nil|)
          (setq td
            (cond
              ((progn
                (setq tmp1 (|modeIsAggregateOf| '|List| mode env))
                (and (consp tmp1)
                  (progn
                    (setq tmp2 (qrest tmp1))

```

```

        (and (consp tmp2)
              (eq (qrest tmp2) nil)
              (progn
                (setq r (qfirst tmp2)) t))))
    (|compList| form (list '|List| r) env))
  ((progn
    (setq tmp1 (|modeIsAggregateOf| '|Vector| mode env))
    (and (consp tmp1)
          (progn
            (setq tmp2 (qrest tmp1))
            (and (consp tmp2) (eq (qrest tmp2) nil))
            (progn
              (setq r (qfirst tmp2)) t))))
    (|compVector| form (list '|Vector| r) env))))
  (when td (|convert| td mode)))
(t
 (setq tt
  (cond
   ((|isSymbol| form) (or (|compSymbol| form mode env) (return nil)))
   ((and (equal mode |$Expression|)
         (|primitiveType| form)) (list form mode env ))
   ((stringp form) (list form form env ))
   (t (list form (or (|primitiveType| form) (return nil)) env ))))
  (|convert| tt mode))))

```

10.1.22 defun convert

[resolve p347]
[coerce p337]

— defun convert —

```

(defun |convert| (td mode)
  (let (res)
    (when (setq res (|resolve| (second td) mode))
      (|coerce| td res))))

```

10.1.23 defun primitiveType

[\$DoubleFloat p??]
[\$NegativeInteger p??]

```

[$PositiveInteger p??]
[$NonNegativeInteger p??]
[$String p330]
[$EmptyMode p131]

```

— **defun primitiveType** —

```

(defun |primitiveType| (form)
  (declare (special |$DoubleFloat| |$NegativeInteger| |$PositiveInteger|
                    |$NonNegativeInteger| |$String| |$EmptyMode|))
  (cond
    ((null form) |$EmptyMode|)
    ((stringp form) |$String|)
    ((integerp form)
     (cond
       ((eql form 0) |$NonNegativeInteger|)
       ((> form 0) |$PositiveInteger|)
       (t |$NegativeInteger|)))
    ((floatp form) |$DoubleFloat|)
    (t nil)))

```

—————

10.1.24 defun compSymbol

```

[getmode p??]
[get p??]
[NRTgetLocalIndex p??]
[compSymbol member (vol5)]
[isFunction p??]
[errorRef p??]
[stackMessage p??]
[$Symbol p??]
[$Expression p??]
[$FormalMapVariableList p242]
[$compForModeIfTrue p??]
[$formalArgList p??]
[$NoValueMode p131]
[$functorLocalParameters p??]
[$Boolean p??]
[$NoValue p??]

```

— **defun compSymbol** —

```

(defun |compSymbol| (form mode env)
  (let (v mprime newmode)

```



```

(declare (special |$Symbol| |$Expression| |$FormalMapVariableList|
                  |$compForModeIfTrue| |$formalArgList| |$NoValueMode|
                  |$functorLocalParameters| |$Boolean| |$NoValue|))

(cond
  ((eq form '|$NoValue|') (list '|$NoValue|' |$NoValueMode| env ))
  ((|isFluid| form)
   (setq newmode (|getmode| form env))
   (when newmode (list form (|getmode| form env) env)))
  ((eq form '|true|') (list '(quote t) |$Boolean| env ))
  ((eq form '|false|') (list nil |$Boolean| env ))
  ((or (equal form mode)
        (|get| form '|isLiteral| env)) (list (list 'quote form) form env))
  ((setq v (|get| form '|value| env))
   (cond
     ((member form |$functorLocalParameters|)
      ; s will be replaced by an ELT form in beforeCompile
      (|NRTgetLocalIndex| form)
      (list form (second v) env))
     (t
      ; form has been SETQd
      (list form (second v) env))))
  ((setq mprime (|getmode| form env))
   (cond
     ((and (null (|member| form |$formalArgList|))
            (null (member form |$FormalMapVariableList|))
            (null (|isFunction| form env))
            (null (eq |$compForModeIfTrue| t)))
      (|errorRef| form)))
     (list form mprime env ))
  ((member form |$FormalMapVariableList|)
   (|stackMessage| (list '|no mode found for| form )))
  ((or (equal mode |$Expression|) (equal mode |$Symbol|))
   (list (list 'quote form) mode env ))
  ((null (|isFunction| form env)) (|errorRef| form))))

```

10.1.25 defun compList

[comp p497]

— defun compList —

```

(defun |compList| (form mode env)
  (let (tmp1 tmp2 t0 failed (newmode (second mode)))
    (if (null form)
      (list nil mode env)
      (progn

```

```

(setq t0
  (do ((t3 form (cdr t3)) (x nil))
      ((or (atom t3) failed) (unless failed (nreverse0 tmp2)))
      (setq x (car t3))
      (if (setq tmp1 (|comp| x newmode env))
          (progn
            (setq newmode (second tmp1))
            (setq env (third tmp1))
            (push tmp1 tmp2))
          (setq failed t))))
(unless failed
  (cons
    (cons 'list (loop for texpr in t0 collect (car texpr)))
    (list (list 'List newmode env))))))

```

10.1.26 defun compExpression

```

[getl p??]
[compForm p507]
[$insideExpressionIfTrue p??]

```

— defun compExpression —

```

(defun |compExpression| (form mode env)
  (let (|$insideExpressionIfTrue| fn)
    (declare (special |$insideExpressionIfTrue|))
    (setq |$insideExpressionIfTrue| t)
    (if (and (atom (car form)) (setq fn (getl (car form) 'special)))
        (funcall fn form mode env)
        (|compForm| form mode env))))

```

10.1.27 defun compForm

```

[compForm1 p508]
[compArgumentsAndTryAgain p517]
[stackMessageIfNone p??]

```

— defun compForm —

```

(defun |compForm| (form mode env)
  (cond

```

```

(|compForm1| form mode env))
(|compArgumentsAndTryAgain| form mode env))
(t (|stackMessageIfNone| (list '|cannot compile| '|%b| form '|%d| )))))

```

10.1.28 defun compForm1

```

[length p??]
[outputComp p328]
[compOrCroak p496]
[compExpressionList p512]
[coerceable p341]
[comp p497]
[coerce p337]
[compForm2 p513]
[augModemapsFromDomain1 p232]
[getFormModemaps p510]
[nreverse0 p??]
[addDomain p228]
[compToApply p??]
[$NumberOfArgsIfInteger p??]
[$Expression p??]
[$EmptyMode p131]

```

— defun compForm1 —

```

(defun |compForm1| (form mode env)
  (let (|$NumberOfArgsIfInteger| op arg1 domain tmp1 opprime ans mmList td
        tmp2 tmp3 tmp4 tmp5 tmp6 tmp7)
    (declare (special |$NumberOfArgsIfInteger| |$Expression| |$EmptyMode|))
    (setq op (car form))
    (setq arg1 (cdr form))
    (setq |$NumberOfArgsIfInteger| (|#| arg1))
    (cond
      ((eq op '|error|)
        (list
          (cons op
            (dolist (x arg1 (nreverse0 tmp4))
              (setq tmp2 (|outputComp| x env))
              (setq env (third tmp2))
              (push (car tmp2) tmp4)))
            mode env))
      ((and (consp op) (eq (qfirst op) '|elt|)
        (progn
          (setq tmp3 (qrest op))

```

```

      (and (consp tmp3)
        (progn
          (setq domain (qfirst tmp3))
          (setq tmp1 (qrest tmp3))
          (and (consp tmp1)
            (eq (qrest tmp1) nil)
            (progn
              (setq opprime (qfirst tmp1))
              t))))))
    (cond
      ((eq domain '|Lisp|)
        (list
          (cons opprime
            (dolist (x arg1 (nreverse tmp7))
              (setq tmp2 (|compOrCroak| x |$EmptyMode| env))
              (setq env (third tmp2))
              (push (car tmp2) tmp7)))
            mode env))
        ((and (equal domain |$Expression|) (eq opprime '|construct|))
          (|compExpressionList| arg1 mode env))
        ((and (eq opprime '|collect|) (|coerceable| domain mode env))
          (when (setq td (|comp| (cons opprime arg1) domain env))
            (|coerce| td mode)))
        ((and (consp domain) (eq (qfirst domain) '|Mapping|)
          (setq ans
            (|compForm2| (cons opprime arg1) mode
              (setq env (|augModemapsFromDomain1| domain domain env))
              (dolist (x (|getFormModemaps| (cons opprime arg1) env)
                (nreverse0 tmp6))
                (when
                  (and (consp x)
                    (and (consp (qfirst x)) (equal (qcaar x) domain)))
                    (push x tmp6)))))))
            ans)
          ((setq ans
            (|compForm2| (cons opprime arg1) mode
              (setq env (|addDomain| domain env))
              (dolist (x (|getFormModemaps| (cons opprime arg1) env)
                (nreverse0 tmp5))
                (when
                  (and (consp x)
                    (and (consp (qfirst x)) (equal (qcaar x) domain)))
                    (push x tmp5))))))
            ans)
          ((and (eq opprime '|construct|) (|coerceable| domain mode env))
            (when (setq td (|comp| (cons opprime arg1) domain env))
              (|coerce| td mode)))
          (t nil)))
    (t
      (setq env (|addDomain| mode env))

```

```
(cond
  ((and (setq mmList (|getFormModemaps| form env))
        (setq td (|compForm2| form mode env mmList)))
   td)
  (t
   (|compToApply| op arg1 mode env))))))
```

10.1.29 defun getFormModemaps

```
[qcar p??]
[qcdr p??]
[getFormModemaps p510]
[nreverse0 p??]
[get p??]
[nequal p??]
[eltModemapFilter p511]
[last p??]
[length p??]
[stackMessage p??]
[$insideCategoryPackageIfTrue p??]
```

— defun getFormModemaps —

```
(defun |getFormModemaps| (form env)
  (let (op arg1 domain op1 modemapList nargs finalModemapList)
    (declare (special |$insideCategoryPackageIfTrue|))
    (setq op (car form))
    (setq arg1 (cdr form))
    (cond
      ((and (consp op) (eq (qfirst op) '|elt|) (CONSP (qrest op))
            (consp (qcddr op)) (eq (qcdddr op) nil))
       (setq op1 (third op))
       (setq domain (second op))
       (loop for x in (|getFormModemaps| (cons op1 arg1) env)
         when (and (consp x) (consp (qfirst x)) (equal (qcaar x) domain))
         collect x))
      ((null (atom op)) nil)
      (t
       (setq modemapList (|get| op '|modemap| env))
       (when |$insideCategoryPackageIfTrue|
         (setq modemapList
          (loop for x in modemapList
            when (and (consp x) (consp (qfirst x)) (nequal (qcaar x) '$))
            collect x))))))
  (cond
```

```

((eq op '|elt|)
 (setq modemapList (|eltModemapFilter| (|last| arg1) modemapList env)))
((eq op '|setelt|)
 (setq modemapList (|seteltModemapFilter| (CADR arg1) modemapList env)))
(setq nargs (|#| arg1))
(setq finalModemapList
 (loop for mm in modemapList
  when (equal (|#| (cddar mm)) nargs)
  collect mm))
(when (and modemapList (null finalModemapList))
 (|stackMessage|
 (list '|no modemap for| '|%b| op '|%d| '|with| | nargs '| arguments|)))
finalModemapList))

```

10.1.30 defun eltModemapFilter

```

[qcar p??]
[qcdr p??]
[isConstantId p??]
[stackMessage p??]

```

— defun eltModemapFilter —

```

(defun |eltModemapFilter| (name mmList env)
 (let (z)
  (if (|isConstantId| name env)
   (cond
    ((setq z
     (loop for mm in mmList
      when (and (consp mm) (consp (qfirst mm)) (consp (qcdr mm))
              (consp (qcddar mm))
              (consp (qcdddar mm))
              (equal (fourth (first mm)) name))
      collect mm))
     z)
    (t
     (|stackMessage|
      (list '|selector variable: | name '| is undeclared and unbound|))
      nil))
   mmList)))

```

10.1.31 defun seteltModemapFilter

```
[isConstantId p??]
[stackMessage p??]
```

— **defun seteltModemapFilter** —

```
(defun |seteltModemapFilter| (name mmList env)
  (let (z)
    (if (|isConstantId| name env)
      (cond
        ((setq z
          (loop for mm in mmList
            when (equal (car (cdddar mm)) name)
            collect mm))
          z)
      (t
        (|stackMessage|
          (list '|selector variable: | name '| is undeclared and unbound|))
          nil)))
    mmList)))
```

—————

10.1.32 defun compExpressionList

```
[nreverse0 p??]
[comp p497]
[convert p504]
[$Expression p??]
```

— **defun compExpressionList** —

```
(defun |compExpressionList| (arg1 m env)
  (let (tmp1 tlst)
    (declare (special |$Expression|))
    (setq tlst
      (prog (result)
        (return
          (do ((tmp2 arg1 (cdr tmp2)) (x nil))
            ((or (atom tmp2)) (nreverse0 result))
            (setq x (car tmp2))
            (setq result
              (cons
                (progn
                  (setq tmp1 (or (|comp| x |$Expression| env) (return '|failed|)))
                  (setq env (third tmp1))
```

```

      tmp1)
      result))))))
(unless (eq tlst '|failed|)
  (|convert|
    (list (cons 'list
      (prog (result)
        (return
          (do ((tmp3 tlst (cdr tmp3)) (y nil))
            ((or (atom tmp3)) (nreverse0 result))
            (setq y (car tmp3))
            (setq result (cons (car y) result))))))
      |$Expression| env)
    m))))

```

10.1.33 defun compForm2

```

[take p??]
[length p??]
[nreverse0 p??]
[sublis p??]
[assoc p??]
[PredImplies p??]
[isSimple p??]
[compUniquely p516]
[compFormPartiallyBottomUp p516]
[compForm3 p515]
[$EmptyMode p131]
[$TriangleVariableList p??]

```

— defun compForm2 —

```

(defun |compForm2| (form mode env modemapList)
  (let (op argl sargl aList dc cond nsig v ncond deleteList newList td t1
        partialModeList tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 tmp7)
    (declare (special |$EmptyMode| |$TriangleVariableList|))
    (setq op (car form))
    (setq argl (cdr form))
    (setq sargl (take (|#| argl) |$TriangleVariableList|))
    (setq aList (mapcar #'(lambda (x y) (cons x y)) sargl argl))
    (setq modemaplist (sublis aList modemapList))
    ; now delete any modemaps that are subsumed by something else, provided
    ; the conditions are right (i.e. subsumer true whenever subsumee true)
    (dolist (u modemapList)
      (cond
        ((and (consp u)

```



```

(progn
  (setq tmp6 (qfirst u))
  (and (consp tmp6) (progn (setq dc (qfirst tmp6)) t)))
(progn
  (setq tmp7 (qrest u))
  (and (consp tmp7) (eq (qrest tmp7) nil)
    (progn
      (setq tmp1 (qfirst tmp7))
      (and (consp tmp1)
        (progn
          (setq cond (qfirst tmp1))
          (setq tmp2 (qrest tmp1))
          (and (consp tmp2) (eq (qrest tmp2) nil)
            (progn
              (setq tmp3 (qfirst tmp2))
              (and (consp tmp3) (eq (qfirst tmp3) '|Subsumed|)
                (progn
                  (setq tmp4 (qrest tmp3))
                  (and (consp tmp4)
                    (progn
                      (setq tmp5 (qrest tmp4))
                      (and (consp tmp5)
                        (eq (qrest tmp5) nil)
                        (progn
                          (setq nsig (qfirst tmp5))
                          t))))))))))))))
      (setq v (|assoc| (cons dc nsig) modemapList))
      (consp v)
      (progn
        (setq tmp6 (qrest v))
        (and (consp tmp6) (eq (qrest tmp6) nil)
          (progn
            (setq tmp7 (qfirst tmp6))
            (and (consp tmp7)
              (progn
                (setq ncond (qfirst tmp7))
                t))))))
        (setq deleteList (cons u deleteList))
        (unless (|PredImplies| ncond cond)
          (setq newList (push '(. (car u) (. cond (elt ,dc nil))) newList))))))
    (when deleteList
      (setq modemapList
        (remove-if #'(lambda (x) (member x deletelist)) modemapList)))
    ; it is important that subsumed ops (newList) be considered last
    (when newList (setq modemapList (append modemapList newList)))
    (setq tl
      (loop for x in arg1
        while (and (|isSimple| x)
          (setq td (|compUniquely| x |$EmptyMode| env)))
        collect td

```

```

      do (setq env (third td))))
(cond
  ((some #'identity tl)
   (setq partialModeList (loop for x in tl collect (when x (second x)))))
  (or
   (|compFormPartiallyBottomUp| form mode env modemapList partialModeList)
   (|compForm3| form mode env modemapList)))
(t (|compForm3| form mode env modemapList))))

```

10.1.34 defun compForm3

```

[compFormWithModemap p??]
[compUniquely p516]
[$compUniquelyIfTrue p??]

```

— defun compForm3 —

```

(defun |compForm3| (form mode env modemapList)
  (let (op arg1 mml tt)
    (declare (special |$compUniquelyIfTrue|))
    (setq op (car form))
    (setq arg1 (cdr form))
    (setq tt
      (let (result)
        (maplist #'(lambda (mlst)
          (setq result (or result
            (|compFormWithModemap| form mode env (car (setq mml mlst)))))
          modemapList)
        result)))
    (when |$compUniquelyIfTrue|
      (if (let (result)
        (mapcar #'(lambda (mm)
          (setq result (or result (|compFormWithModemap| form mode env mm))))
        (rest mml))
        result)
        (throw '|compUniquely| nil)
        tt))
    tt))

```

10.1.35 defun compFormPartiallyBottomUp

[compForm3 p515]
[compFormMatch p516]

— defun compFormPartiallyBottomUp —

```
(defun |compFormPartiallyBottomUp| (form mode env modemapList partialModeList)
  (let (mmList)
    (when (setq mmList (loop for mm in modemapList
                            when (|compFormMatch| mm partialModeList)
                            collect mm))
      (|compForm3| form mode env mmList))))
```

10.1.36 defun compFormMatch

— defun compFormMatch —

```
(defun |compFormMatch| (mm partialModeList)
  (labels (
    (ismatch (a b)
      (cond
        ((null b) t)
        ((null (car b)) (|compFormMatch,match| (cdr a) (cdr b)))
        ((and (equal (car a) (car b)) (ismatch (cdr a) (cdr b))))))
    (and (consp mm) (consp (qfirst mm)) (consp (qcddar mm))
      (ismatch (qcddar mm) partialModeList))))
```

10.1.37 defun compUniquely

[compUniquely p516]
[comp p497]
[\$compUniquelyIfTrue p??]

— defun compUniquely —

```
(defun |compUniquely| (x m env)
  (let (|$compUniquelyIfTrue|)
    (declare (special |$compUniquelyIfTrue|))
    (setq |$compUniquelyIfTrue| t)
```

```
(catch '|compUniquely| (|comp| x m env)))
```

10.1.38 defun compArgumentsAndTryAgain

```
[comp p497]
[compForm1 p508]
[$EmptyMode p131]
```

— defun compArgumentsAndTryAgain —

```
(defun |compArgumentsAndTryAgain| (form mode env)
  (let (arg1 tmp1 a tmp2 tmp3 u)
    (declare (special |$EmptyMode|))
    (setq arg1 (cdr form))
    (cond
      ((and (consp form) (eq (qfirst form) '|elt|))
        (progn
          (setq tmp1 (qrest form))
          (and (consp tmp1)
            (progn
              (setq a (qfirst tmp1))
              (setq tmp2 (qrest tmp1))
              (and (consp tmp2) (eq (qrest tmp2) nil))))))
        (when (setq tmp3 (|comp| a |$EmptyMode| env))
          (setq env (third tmp3))
          (|compForm1| form mode env)))
      (t
        (setq u
          (dolist (x arg1)
            (setq tmp3 (or (|comp| x |$EmptyMode| env) (return '|failed|)))
            (setq env (third tmp3))
            tmp3))
          (unless (eq u '|failed|)
            (|compForm1| form mode env))))))
```

10.1.39 defun compWithMappingMode

```
[compWithMappingMode1 p518]
[$formalArgList p??]
```

— defun compWithMappingMode —

```
(defun |compWithMappingMode| (form mode oldE)
  (declare (special |$formalArgList|))
  (|compWithMappingMode1| form mode oldE |$formalArgList|))
```

10.1.40 defun compWithMappingMode1

```
[isFunctor p229]
[get p??]
[qcar p??]
[qcdr p??]
[extendsCategoryForm p??]
[compLambda p307]
[stackAndThrow p??]
[take p??]
[compMakeDeclaration p525]
[hasFormalMapVariable p524]
[comp p497]
[extractCodeAndConstructTriple p523]
[optimizeFunctionDef p203]
[comp-tran p??]
[freelist p526]
[$formalArgList p??]
[$killOptimizeIfTrue p??]
[$funname p??]
[$funnameTail p??]
[$QuickCode p??]
[$EmptyMode p131]
[$FormalMapVariableList p242]
[$CategoryFrame p??]
[$formatArgList p??]
```

— defun compWithMappingMode1 —

```
(defun |compWithMappingMode1| (form mode oldE |$formalArgList|)
  (declare (special |$formalArgList|))
  (prog (|$killOptimizeIfTrue| $funname $funnameTail mprime s1 tmp1 tmp2
        tmp3 tmp4 tmp5 tmp6 target argModeList nx oldstyle ress v11 v1 e tt
        u frees i scode locals body vec expandedFunction fname uu)
    (declare (special |$killOptimizeIfTrue| $funname $funnameTail
                      |$QuickCode| |$EmptyMode| |$FormalMapVariableList|
                      |$CategoryFrame| |$formatArgList|))
    (return
     (seq
```

```

(progn
  (setq mprime (second mode))
  (setq s1 (cddr mode))
  (setq |$killOptimizeIfTrue| t)
  (setq e oldE)
  (cond
    ((|isFunctor| form)
     (cond
       ((and (progn
              (setq tmp1 (|get| form '|modemap| |$CategoryFrame|))
              (and (consp tmp1)
                   (progn
                     (setq tmp2 (qfirst tmp1))
                     (and (consp tmp2)
                          (progn
                            (setq tmp3 (qfirst tmp2))
                            (and (consp tmp3)
                                 (progn
                                   (setq tmp4 (qrest tmp3))
                                   (and (consp tmp4)
                                        (progn
                                          (setq target (qfirst tmp4))
                                          (setq argModeList (qrest tmp4))
                                          t))))))
                          (progn
                            (setq tmp5 (qrest tmp2))
                            (and (consp tmp5) (eq (qrest tmp5) nil))))))))
              (progn (t1)
                      (setq t1 t)
                      (return
                       (do ((t2 nil (null t1))
                           (t3 argModeList (cdr t3))
                           (newmode nil)
                           (t4 s1 (cdr t4))
                           (s nil))
                           ((or t2 (atom t3)
                                (progn (setq newmode (car t3)) nil)
                                (atom t4)
                                (progn (setq s (car t4)) nil))
                               t1)
                       (seq (exit
                            (setq t1
                              (and t1 (|extendsCategoryForm| '$ s newmode))))))
                           (|extendsCategoryForm| '$ target mprime))
                       (return (list form mode e )))
                      (t nil)))
       (t
        (when (stringp form) (setq form (intern form)))
        (setq ress nil)
        (setq oldstyle t)

```

```

(cond
  ((and (consp form)
        (eq (qfirst form) '+->)
        (progn
          (setq tmp1 (qrest form))
          (and (consp tmp1)
               (progn
                 (setq v1 (qfirst tmp1))
                 (setq tmp2 (qrest tmp1))
                 (and (consp tmp2)
                      (eq (qrest tmp2) nil)
                      (progn (setq nx (qfirst tmp2)) t)))))))
    (setq oldstyle nil)
  (cond
    ((and (consp v1) (eq (qfirst v1) '|:|))
     (setq ress (|compLambda| form mode oldE))
     ress)
    (t
     (setq v1
      (cond
        ((and (consp v1)
              (eq (qfirst v1) '|@Tuple|)
              (progn (setq v11 (qrest v1)) t))
         v11)
        (t v1)))
     (setq v1
      (cond
        ((symbolp v1) (cons v1 nil))
        ((and
          (listp v1)
          (prog (t5)
                (setq t5 t)
                (return
                 (do ((t7 nil (null t5))
                     (t6 v1 (cdr t6))
                     (v nil))
                     ((or t7 (atom t6) (progn (setq v (car t6)) nil)) t5)
                     (seq
                      (exit
                       (setq t5 (and t5 (symbolp v))))))))))
         v1)
        (t
         (|stackAndThrow| (cons '|bad +-> arguments:| (list v1 ))))))
     (setq |$formatArgList| (append v1 |$formalArgList|))
     (setq form nx))))
  (t
   (setq v1 (take (|#| s1) |$FormalMapVariableList|)))
  (cond
    (ress ress)
    (t

```

```

(do ((t8 sl (cdr t8)) (m nil) (t9 vl (cdr t9)) (v nil))
  ((or (atom t8)
       (progn (setq m (car t8)) nil)
       (atom t9)
       (progn (setq v (car t9)) nil))
   nil)
  (seq (exit (progn
    (setq tmp6
      (|compMakeDeclaration| (list '|| v m ) |$EmptyMode| e))
    (setq e (third tmp6))
    tmp6))))))
(cond
  ((and oldstyle
        (null (null vl))
        (null (|hasFormalMapVariable| form vl))))
  (return
   (progn
    (setq tmp6 (or (|comp| (cons form vl) mprime e) (return nil)))
    (setq u (car tmp6))
    (|extractCodeAndConstructTriple| u mode oldE))))
  ((and (null vl) (setq tt (|comp| (cons form nil) mprime e)))
   (return
    (progn
     (setq u (car tt))
     (|extractCodeAndConstructTriple| u mode oldE))))
  (t
   (setq tmp6 (or (|comp| form mprime e) (return nil)))
   (setq u (car tmp6))
   (setq uu (|optimizeFunctionDef| '(nil (lambda ,vl ,u))))
; -- At this point, we have a function that we would like to pass.
; -- Unfortunately, it makes various free variable references outside
; -- itself. So we build a mini-vector that contains them all, and
; -- pass this as the environment to our inner function.
   (setq $funname nil)
   (setq $funnameTail (list nil))
   (setq expandedFunction (comp-tran (second uu)))
   (setq frees (freelist expandedFunction vl nil e))
   (setq expandedFunction
    (cond
      ((eql (|#| frees) 0)
       (cons 'lambda (cons (append vl (list '$$))
                            (caddr expandedFunction))))
      ((eql (|#| frees) 1)
       (setq vec (caar frees))
       (cons 'lambda (cons (append vl (list vec))
                            (caddr expandedFunction))))
      (t
       (setq scode nil)
       (setq vec nil)
       (setq locals nil)

```



```

(setq i -1)
(do ((t0 frees (cdr t0)) (v nil))
    ((or (atom t0) (progn (setq v (car t0)) nil)) nil)
  (seq
   (exit
    (progn
     (setq i (plus i 1))
     (setq vec (cons (car v) vec))
     (setq scode
      (cons
       (cons 'setq
        (cons (car v)
         (cons
          (cons
           (cond
            (|$QuickCode| 'qrefelt)
            (t 'elt)))
           (cons '$$ (cons i nil)))
          nil)))
       scode))
     (setq locals (cons (car v) locals))))))
(setq body (cddr expandedFunction))
(cond
 (locals
  (cond
   ((and (consp body)
        (progn
         (setq tmp1 (qfirst body))
         (and (consp tmp1)
              (eq (qfirst tmp1) 'declare))))
    (setq body
     (cons (car body)
      (cons
       (cons 'prog
        (cons locals
         (append scode
          (cons
           (cons 'return
            (cons
             (cons 'progn
              (cdr body))
              nil))
            nil)))
          nil))))
      nil))))
  (t
   (setq body
    (cons
     (cons 'prog
      (cons locals
       (append scode

```

```

      (cons
        (cons 'return
          (cons
            (cons 'progn body)
            nil))
          nil))))
    nil))))))
  (setq vec (cons 'vector (nreverse vec)))
  (cons 'lambda (cons (append vl (list '$$)) body))))
(setq fname (list 'closedfn expandedFunction))
(setq uu
  (cond
    (frees (list 'cons fname vec))
    (t (list 'list fname))))
(list uu mode oldE)))))))))

```

10.1.41 defun extractCodeAndConstructTriple

— defun extractCodeAndConstructTriple —

```

(defun |extractCodeAndConstructTriple| (form mode oldE)
  (let (tmp1 a fn op env)
    (cond
      ((and (consp form) (eq (qfirst form) '|call|))
        (progn
          (setq tmp1 (qrest form))
          (and (consp tmp1)
            (progn (setq fn (qfirst tmp1)) t))))
      (cond
        ((and (consp fn) (eq (qfirst fn) '|applyFun|))
          (progn
            (setq tmp1 (qrest fn))
            (and (consp tmp1) (eq (qrest tmp1) nil)
              (progn (setq a (qfirst tmp1)) t))))
          (setq fn a)))
      (list fn mode oldE))
    (t
      (setq op (car form))
      (setq env (car (reverse (cdr form)))))
      (list (list 'cons (list '|function| op) env) mode oldE))))

```

10.1.42 defun hasFormalMapVariable

[hasFormalMapVariable ScanOrPairVec (vol5)]
 [\$formalMapVariables p??]

— defun hasFormalMapVariable —

```
(defun |hasFormalMapVariable| (x vl)
  (let (|$formalMapVariables|)
    (declare (special |$formalMapVariables|))
    (when (setq |$formalMapVariables| vl)
      (|ScanOrPairVec| #'(lambda (y) (member y |$formalMapVariables|)) x))))
```

—————

10.1.43 defun argsToSig

— defun argsToSig —

```
(defun |argsToSig| (args)
  (let (tmp1 v tmp2 tt sig1 arg1 bad)
    (cond
      ((and (consp args) (eq (qfirst args) '|:|))
        (progn
          (setq tmp1 (qrest args))
          (and (consp tmp1)
            (progn
              (setq v (qfirst tmp1))
              (setq tmp2 (qrest tmp1))
              (and (consp tmp2)
                (eq (qrest tmp2) nil)
                (progn
                  (setq tt (qfirst tmp2))
                  t))))))
        (list (list v) (list tt)))
      (t
        (setq sig1 nil)
        (setq arg1 nil)
        (setq bad nil)
        (dolist (arg args)
          (cond
            ((and (consp arg) (eq (qfirst arg) '|:|))
              (progn
                (setq tmp1 (qrest arg))
                (and (consp tmp1)
                  (progn
```

```

      (setq v (qfirst tmp1))
      (setq tmp2 (qrest tmp1))
      (and (consp tmp2) (eq (qrest tmp2) nil)
        (progn
          (setq tt (qfirst tmp2))
          t))))))
    (setq sig1 (cons tt sig1))
    (setq arg1 (cons v arg1))
    (t (setq bad t))))
  (cond
    (bad (list nil nil ))
    (t (list (reverse arg1) (reverse sig1))))))

```

10.1.44 defun compMakeDeclaration

```

[compColon p267]
[$insideExpressionIfTrue p??]

```

— defun compMakeDeclaration —

```

(defun |compMakeDeclaration| (form mode env)
  (let (|$insideExpressionIfTrue|)
    (declare (special |$insideExpressionIfTrue|))
    (setq |$insideExpressionIfTrue| nil)
    (|compColon| form mode env)))

```

10.1.45 defun modifyModeStack

```

[say p??]
[copy p??]
[setelt p??]
[resolve p347]
[$reportExitModeStack p??]
[$exitModeStack p??]

```

— defun modifyModeStack —

```

(defun |modifyModeStack| (m index)
  (declare (special |$exitModeStack| |$reportExitModeStack|))
  (if |$reportExitModeStack|
    (say "exitModeStack: " (copy |$exitModeStack|)

```

```
" ==> "
(progn
  (setelt |$exitModeStack| index
    (|resolve| m (elt |$exitModeStack| index)))
  |$exitModeStack|)
(setelt |$exitModeStack| index
  (|resolve| m (elt |$exitModeStack| index))))
```

10.1.46 defun Create a list of unbound symbols

We walk argument *u* looking for symbols that are unbound. If we find a symbol we add it to the free list. If it occurs in a prog then it is bound and we remove it from the free list. Multiple instances of a single symbol in the free list are represented by the alist (symbol . count) [freelist p526]

```
[freelist assq (vol5)]
[freelist identp (vol5)]
[getmode p??]
[unionq p??]
```

— defun freelist —

```
(defun freelist (u bound free e)
  (let (v op)
    (if (atom u)
      (cond
        ((null (identp u)) free)
        ((member u bound) free)
        ; more than 1 free becomes alist (name . number)
        ((setq v (assq u free)) (rplacd v (+ 1 (cdr v))) free)
        ((null (|getmode| u e)) free)
        (t (cons (cons u 1) free)))
      (progn
        (setq op (car u))
        (cond
          ((member op '(quote go |function|)) free)
          ((eq op 'lambda) ; lambdas bind symbols
            (setq bound (unionq bound (second u)))
            (dolist (v (cddr u))
              (setq free (freelist v bound free e))))
          ((eq op 'prog) ; progs bind symbols
            (setq bound (unionq bound (second u)))
            (dolist (v (cddr u))
              (unless (atom v)
                (setq free (freelist v bound free e))))))
          ((eq op 'seq)
```

```

(dolist (v (cdr u))
  (unless (atom v)
    (setq free (freelist v bound free e))))
((eq op 'cond)
  (dolist (v (cdr u))
    (dolist (vv v)
      (setq free (freelist vv bound free e))))
  (t
    (when (atom op) (setq u (cdr u))) ; atomic functions aren't descended
    (dolist (v u)
      (setq free (freelist v bound free e))))
  free)))

```

10.1.47 defun compOrCroak1,compactify

[compOrCroak1,compactify p527]
 [lassoc p??]

— defun compOrCroak1,compactify —

```

(defun |compOrCroak1,compactify| (al)
  (cond
    ((null al) nil)
    ((lassoc (caar al) (cdr al)) (|compOrCroak1,compactify| (cdr al)))
    (t (cons (car al) (|compOrCroak1,compactify| (cdr al))))))

```

10.1.48 defun Compiler/Interpreter interface

[ncINTERPFILE SpadInterpretStream (vol5)]
 [\$EchoLines p??]
 [\$ReadingFile p??]

— defun ncINTERPFILE —

```

(defun |ncINTERPFILE| (file echo)
  (let ((|$EchoLines| echo) (|$ReadingFile| t))
    (declare (special |$EchoLines| |$ReadingFile|))
    (|SpadInterpretStream| 1 file nil)))

```

10.1.49 defun compileSpadLispCmd

```
[compileSpadLispCmd pathname (vol5)]
[compileSpadLispCmd pathnameType (vol5)]
[compileSpadLispCmd selectOptionLC (vol5)]
[compileSpadLispCmd namestring (vol5)]
[compileSpadLispCmd terminateSystemCommand (vol5)]
[compileSpadLispCmd fnameMake (vol5)]
[compileSpadLispCmd pathnameDirectory (vol5)]
[compileSpadLispCmd pathnameName (vol5)]
[compileSpadLispCmd fnameReadable? (vol5)]
[compileSpadLispCmd localdatabase (vol5)]
[throwKeyedMsg p??]
[object2String p??]
[compileSpadLispCmd sayKeyedMsg (vol5)]
[recompile-lib-file-if-necessary p529]
[spadPrompt p??]
[$options p??]
```

— defun compileSpadLispCmd —

```
(defun |compileSpadLispCmd| (args)
  (let (path optlist optname optargs beQuiet dolibrary lsp)
    (declare (special |$options|))
    (setq path (|pathname| (|fnameMake| (car args) "code" "lsp")))
    (cond
      ((null (probe-file path))
        (|throwKeyedMsg| 's2il0003 (cons (|namestring| args) nil)))
      (t
        (setq optlist '(|quiet| |noquiet| |library| |nolibrary|))
        (setq beQuiet nil)
        (setq dolibrary t)
        (dolist (opt |$options|)
          (setq optname (car opt))
          (setq optargs (cdr opt))
          (case (|selectOptionLC| optname optlist nil)
            (|quiet|      (setq beQuiet t))
            (|noquiet|    (setq beQuiet nil))
            (|library|    (setq dolibrary t))
            (|nolibrary| (setq dolibrary nil))
            (t
              (|throwKeyedMsg| 's2iz0036
                (list (strconc " " (|object2String| optname)))))))
        (setq lsp
          (|fnameMake|
            (|pathnameDirectory| path)
            (|pathnameName| path)
            (|pathnameType| path))))))
```

```

(cond
  ((|fnameReadable?| lsp)
   (unless beQuiet (|sayKeyedMsg| 's2iz0089 (list (|namestring| lsp))))
   (recompile-lib-file-if-necessary lsp))
  (t
   (|sayKeyedMsg| 's2il0003 (list (|namestring| lsp)))))
(cond
  (dolibrary
   (unless beQuiet (|sayKeyedMsg| 's2iz0090 (list (|pathnameName| path))))
   (localdatabase (list (|pathnameName| (car args))) nil))
  ((null beQuiet) (|sayKeyedMsg| 's2iz0084 nil))
  (t nil))
(|terminateSystemCommand|
 |spadPrompt|))))

```

10.1.50 defun recompile-lib-file-if-necessary

```

[compile-lib-file p530]
[*lisp-bin-filetype* p??]

```

— defun recompile-lib-file-if-necessary —

```

(defun recompile-lib-file-if-necessary (lfile)
  (let* ((bfile (make-pathname :type *lisp-bin-filetype* :defaults lfile))
         (bdate (and (probe-file bfile) (file-write-date bfile)))
         (ldate (and (probe-file lfile) (file-write-date lfile))))
    (declare (special *lisp-bin-filetype*))
    (unless (and ldate bdate (> bdate ldate))
      (compile-lib-file lfile)
      (list bfile))))

```

10.1.51 defun spad-fixed-arg

— defun spad-fixed-arg —

```

(defun spad-fixed-arg (fname )
  (and (equal (symbol-package fname) (find-package "BOOT"))
       (not (get fname 'compiler::spad-var-arg))
       (search ";" (symbol-name fname))
       (or (get fname 'compiler::fixed-args)

```



```

      (setf (get fname 'compiler::fixed-args) t)))
  nil)

```

10.1.52 defun compile-lib-file

— defun compile-lib-file —

```

(defun compile-lib-file (fn &rest opts)
  (unwind-protect
    (progn
      (trace (compiler::fast-link-proclaimed-type-p
              :exitcond nil
              :entrycond (spad-fixed-arg (car system::arglist))))
      (trace (compiler::t1defun
              :exitcond nil
              :entrycond (spad-fixed-arg (caar system::arglist))))
      (apply #'compile-file fn opts))
    (untrace compiler::fast-link-proclaimed-type-p compiler::t1defun)))

```

10.1.53 defun compileFileQuietly

if `$InteractiveMode` then use a null outputstream [`$InteractiveMode p??`]
 [`*standard-output* p??`]

— defun compileFileQuietly —

```

(defun |compileFileQuietly| (fn)
  (let (
    (*standard-output*
     (if |$InteractiveMode| (make-broadcast-stream
                           *standard-output*)))
    (declare (special *standard-output* |$InteractiveMode|))
    (compile-file fn)))

```

10.1.54 defvar \$byConstructors

— initvars —

```
(defvar |$byConstructors| () "list of constructors to be compiled")
```

—————

10.1.55 defvar \$constructorsSeen

— initvars —

```
(defvar |$constructorsSeen| () "list of constructors found")
```

—————

Chapter 11

Level 1

11.0.56 `defvar $current-fragment`

A string containing remaining chars from `readline`; needed because Symbolics `read-line` returns embedded newlines in a c-m-Y.

— **initvars** —

```
(defvar current-fragment nil)
```

—————

11.0.57 `defun read-a-line`

```
[subseq p??]  
[Line-New-Line p??]  
[read-a-line p533]  
[*eof* p??]  
[File-Closed p??]
```

— **defun read-a-line** —

```
(defun read-a-line (&optional (stream t))  
  (let (cp)  
    (declare (special *eof* File-Closed))  
    (if (and Current-Fragment (> (length Current-Fragment) 0))  
        (let ((line (with-input-from-string  
                      (s Current-Fragment :index cp :start 0)  
                      (read-line s nil nil))))  
          (setq Current-Fragment (subseq Current-Fragment cp))  
          line)  
        (read-line stream))
```

```
(prog nil
  (when (stream-eof in-stream)
    (setq File-Closed t)
    (setq *eof* t)
    (Line-New-Line (make-string 0) Current-Line)
    (return nil))
  (when (setq Current-Fragment (read-line stream))
    (return (read-a-line stream))))))
```

Chapter 12

Level 0

12.1 Line Handling

12.1.1 Line Buffer

The philosophy of lines is that

- NEXT LINE will always return a non-blank line or fail.
- Every line is terminated by a blank character.

Hence there is always a current character, because there is never a non-blank line, and there is always a separator character between tokens on separate lines. Also, when a line is read, the character pointer is always positioned ON the first character.

12.1.2 defstruct \$line

— initvars —

```
(defstruct line "Line of input file to parse."  
  (buffer (make-string 0) :type string)  
  (current-char #\Return :type character)  
  (current-index 1 :type fixnum)  
  (last-index 0 :type fixnum)  
  (number 0 :type fixnum))
```

—————

12.1.3 defvar \$current-line

The current input line.

— **initvars** —

```
(defvar current-line (make-line))
```

12.1.4 defmacro line-clear

[*\$line* p535]

— **defmacro line-clear** —

```
(defmacro line-clear (line)
  '(let ((l ,line))
    (setf (line-buffer l) (make-string 0))
    (setf (line-current-char l) #\return)
    (setf (line-current-index l) 1)
    (setf (line-last-index l) 0)
    (setf (line-number l) 0)))
```

12.1.5 defun line-print

[*\$line* p535]

[*\$out-stream* p??]

— **defun line-print** —

```
(defun line-print (line)
  (declare (special out-stream))
  (format out-stream "~&~5D> ~A~%" (Line-Number line) (Line-Buffer line))
  (format out-stream "~v@T~%" (+ 7 (Line-Current-Index line))))
```

12.1.6 defun line-at-end-p

[*\$line* p535]

— **defun line-at-end-p** —

```
(defun line-at-end-p (line)
  "Tests if line is empty or positioned past the last character."
  (>= (line-current-index line) (line-last-index line)))
```

12.1.7 defun line-past-end-p

[*\$line* p535]

— **defun line-past-end-p** —

```
(defun line-past-end-p (line)
  "Tests if line is empty or positioned past the last character."
  (> (line-current-index line) (line-last-index line)))
```

12.1.8 defun line-next-char

[*\$line* p535]

— **defun line-next-char** —

```
(defun line-next-char (line)
  (elt (line-buffer line) (1+ (line-current-index line))))
```

12.1.9 defun line-advance-char

[*\$line* p535]

— **defun line-advance-char** —

```
(defun line-advance-char (line)
  (setf (line-current-char line)
        (elt (line-buffer line) (incf (line-current-index line)))))
```

12.1.10 defun line-current-segment

[*\$line* p535]

— defun line-current-segment —

```
(defun line-current-segment (line)
  "Buffer from current index to last index."
  (if (line-at-end-p line)
      (make-string 0)
      (subseq (line-buffer line)
               (line-current-index line)
               (line-last-index line))))
```

—————

12.1.11 defun line-new-line

[*\$line* p535]

— defun line-new-line —

```
(defun line-new-line (string line &optional (linenum nil))
  "Sets string to be the next line stored in line."
  (setf (line-last-index line) (1- (length string)))
  (setf (line-current-index line) 0)
  (setf (line-current-char line)
        (or (and (> (length string) 0) (elt string 0)) #\Return))
  (setf (line-buffer line) string)
  (setf (line-number line) (or linenum (1+ (line-number line)))))
```

—————

12.1.12 defun next-line

[*\$in-stream* p??]
 [*\$line-handler* p??]

— defun next-line —

```
(defun next-line (&optional (in-stream t))
  (declare (special in-stream line-handler))
  (funcall Line-Handler in-stream))
```

—————

12.1.13 defun Advance-Char

[Line-At-End-P p??]
 [Line-Advance-Char p??]
 [next-line p538]
 [current-char p449]
 [\$in-stream p??]
 [\$line p535]

— defun Advance-Char —

```
(defun Advance-Char ()
  "Advances IN-STREAM, invoking Next Line if necessary."
  (declare (special in-stream))
  (loop
    (cond
      ((not (Line-At-End-P Current-Line))
       (return (Line-Advance-Char Current-Line)))
      ((next-line in-stream)
       (return (current-char)))
      ((return nil)))))
```

—————

12.1.14 defun storeblanks

— defun storeblanks —

```
(defun storeblanks (line n)
  (do ((i 0 (1+ i)))
      ((= i n) line)
    (setf (char line i) #\ )))
```

—————

12.1.15 defun initial-substring

[mismatch p??]

— defun initial-substring —

```
(defun initial-substring (pattern line)
  (let ((ind (mismatch pattern line)))
```

```
(or (null ind) (eq1 ind (size pattern))))))
```

12.1.16 defun get-a-line

```
[is-console p467]
[get-a-line mkprompt (vol5)]
[read-a-line p533]
```

— defun get-a-line —

```
(defun get-a-line (stream)
  (when (is-console stream) (princ (mkprompt)))
  (let ((l1 (read-a-line stream)))
    (if (and (stringp l1) (adjustable-array-p l1))
        (make-array (array-dimensions l1) :element-type 'string-char
                     :adjustable t :initial-contents l1)
        l1)))
```

Chapter 13

The Chunks

— Compiler —

```
(in-package "BOOT")

\getchunk{initvars}

\getchunk{LEDNUDTables}
\getchunk{GLIPHTable}
\getchunk{RENAME TOKTable}
\getchunk{GENERICTable}

\getchunk{defmacro bang}
\getchunk{defmacro line-clear}
\getchunk{defmacro must}
\getchunk{defmacro nth-stack}
\getchunk{defmacro pop-stack-1}
\getchunk{defmacro pop-stack-2}
\getchunk{defmacro pop-stack-3}
\getchunk{defmacro pop-stack-4}
\getchunk{defmacro reduce-stack-clear}
\getchunk{defmacro stack-/empty}
\getchunk{defmacro star}

\getchunk{defun action}
\getchunk{defun addArgumentConditions}
\getchunk{defun addclose}
\getchunk{defun addConstructorModemaps}
\getchunk{defun addDomain}
\getchunk{defun addEltModemap}
\getchunk{defun addEmptyCapsuleIfNecessary}
\getchunk{defun addModemapKnown}
```

```

\getchunk{defun addModemap}
\getchunk{defun addModemap0}
\getchunk{defun addModemap1}
\getchunk{defun addNewDomain}
\getchunk{defun add-parens-and-semis-to-line}
\getchunk{defun addSuffix}
\getchunk{defun Advance-Char}
\getchunk{defun advance-token}
\getchunk{defun alistSize}
\getchunk{defun allLASSOCs}
\getchunk{defun aplTran}
\getchunk{defun aplTran1}
\getchunk{defun aplTranList}
\getchunk{defun argsToSig}
\getchunk{defun assignError}
\getchunk{defun AssocBarGensym}
\getchunk{defun augLisplibModemapsFromCategory}
\getchunk{defun augmentLisplibModemapsFromFunctor}
\getchunk{defun augModemapsFromCategory}
\getchunk{defun augModemapsFromCategoryRep}
\getchunk{defun augModemapsFromDomain}
\getchunk{defun augModemapsFromDomain1}
\getchunk{defun autoCoerceByModemap}

\getchunk{defun blankp}
\getchunk{defun bootStrapError}
\getchunk{defun bumperrorcount}

\getchunk{defun canReturn}
\getchunk{defun char-eq}
\getchunk{defun char-ne}
\getchunk{defun checkAndDeclare}
\getchunk{defun checkWarning}
\getchunk{defun coerce}
\getchunk{defun coerceable}
\getchunk{defun coerceByModemap}
\getchunk{defun coerceEasy}
\getchunk{defun coerceExit}
\getchunk{defun coerceExtraHard}
\getchunk{defun coerceHard}
\getchunk{defun coerceSubset}
\getchunk{defun comma2Tuple}
\getchunk{defun comp}
\getchunk{defun comp2}
\getchunk{defun comp3}
\getchunk{defun compAdd}
\getchunk{defun compArgumentConditions}
\getchunk{defun compArgumentsAndTryAgain}
\getchunk{defun compAtom}
\getchunk{defun compAtSign}

```

```

\getchunk{defun compBoolean}
\getchunk{defun compCapsule}
\getchunk{defun compCapsuleInner}
\getchunk{defun compCapsuleItems}
\getchunk{defun compCase}
\getchunk{defun compCase1}
\getchunk{defun compCat}
\getchunk{defun compCategory}
\getchunk{defun compCategoryItem}
\getchunk{defun compCoerce}
\getchunk{defun compCoerce1}
\getchunk{defun compColon}
\getchunk{defun compColonInside}
\getchunk{defun compCons}
\getchunk{defun compCons1}
\getchunk{defun compConstruct}
\getchunk{defun compConstructorCategory}
\getchunk{defun compDefine}
\getchunk{defun compDefine1}
\getchunk{defun compDefineAddSignature}
\getchunk{defun compDefineCapsuleFunction}
\getchunk{defun compDefineCategory}
\getchunk{defun compDefineCategory1}
\getchunk{defun compDefineCategory2}
\getchunk{defun compDefineFunctor}
\getchunk{defun compDefineFunctor1}
\getchunk{defun compDefineLisplib}
\getchunk{defun compDefWhereClause}
\getchunk{defun compElt}
\getchunk{defun compExit}
\getchunk{defun compExpression}
\getchunk{defun compExpressionList}
\getchunk{defun compForm}
\getchunk{defun compForm1}
\getchunk{defun compForm2}
\getchunk{defun compForm3}
\getchunk{defun compFormMatch}
\getchunk{defun compForMode}
\getchunk{defun compFormPartiallyBottomUp}
\getchunk{defun compFromIf}
\getchunk{defun compFunctorBody}
\getchunk{defun compHas}
\getchunk{defun compHasFormat}
\getchunk{defun compIf}
\getchunk{defun compile}
\getchunk{defun compileCases}
\getchunk{defun compileConstructor}
\getchunk{defun compileConstructor1}
\getchunk{defun compileDocumentation}
\getchunk{defun compileFileQuietly}

```

```
\getchunk{defun compile-lib-file}
\getchunk{defun compiler}
\getchunk{defun compilerDoit}
\getchunk{defun compilerDoitWithScreenedLisplib}
\getchunk{defun compileSpad2Cmd}
\getchunk{defun compileSpadLispCmd}
\getchunk{defun compileTimeBindingOf}
\getchunk{defun compImport}
\getchunk{defun compInternalFunction}
\getchunk{defun compIs}
\getchunk{defun compJoin}
\getchunk{defun compLambda}
\getchunk{defun compLeave}
\getchunk{defun compList}
\getchunk{defun compMacro}
\getchunk{defun compMakeCategoryObject}
\getchunk{defun compMakeDeclaration}
\getchunk{defun compNoStacking}
\getchunk{defun compNoStacking1}
\getchunk{defun compOrCroak}
\getchunk{defun compOrCroak1}
\getchunk{defun compOrCroak1,compactify}
\getchunk{defun compPretend}
\getchunk{defun compQuote}
\getchunk{defun compRepeatOrCollect}
\getchunk{defun compReduce}
\getchunk{defun compReduce1}
\getchunk{defun compReturn}
\getchunk{defun compSeq}
\getchunk{defun compSeqItem}
\getchunk{defun compSeq1}
\getchunk{defun compSetq}
\getchunk{defun compSetq1}
\getchunk{defun compSingleCapsuleItem}
\getchunk{defun compString}
\getchunk{defun compSubDomain}
\getchunk{defun compSubDomain1}
\getchunk{defun compSymbol}
\getchunk{defun compSubsetCategory}
\getchunk{defun compSuchthat}
\getchunk{defun compTopLevel}
\getchunk{defun compTuple2Record}
\getchunk{defun compTypeOf}
\getchunk{defun compUniquely}
\getchunk{defun compVector}
\getchunk{defun compWhere}
\getchunk{defun compWithMappingMode}
\getchunk{defun compWithMappingModel}
\getchunk{defun constructMacro}
\getchunk{defun containsBang}
```

```

\getchunk{defun convert}
\getchunk{defun convertOpAlist2compilerInfo}
\getchunk{defun convertOrCroak}
\getchunk{defun current-char}
\getchunk{defun current-symbol}
\getchunk{defun current-token}

\getchunk{defun decodeScripts}
\getchunk{defun deepestExpression}
\getchunk{defun def-rename}
\getchunk{defun def-rename1}
\getchunk{defun disallowNilAttribute}
\getchunk{defun displayMissingFunctions}
\getchunk{defun displayPreCompilationErrors}
\getchunk{defun doIt}
\getchunk{defun doItIf}
\getchunk{defun dollarTran}
\getchunk{defun domainMember}
\getchunk{defun drop}

\getchunk{defun eltModemapFilter}
\getchunk{defun encodeItem}
\getchunk{defun encodeFunctionName}
\getchunk{defun EqualBarGensym}
\getchunk{defun errhuh}
\getchunk{defun escape-keywords}
\getchunk{defun escaped}
\getchunk{defun evalAndRwriteLispForm}
\getchunk{defun evalAndSub}
\getchunk{defun extractCodeAndConstructTriple}

\getchunk{defun flattenSignatureList}
\getchunk{defun finalizeLisplib}
\getchunk{defun fincomblock}
\getchunk{defun fixUpPredicate}
\getchunk{defun floatexpid}
\getchunk{defun formal2Pattern}
\getchunk{defun freelist}

\getchunk{defun get-a-line}
\getchunk{defun getAbbreviation}
\getchunk{defun getArgumentMode}
\getchunk{defun getArgumentModeOrMoan}
\getchunk{defun getCaps}
\getchunk{defun getCategoryOpsAndAtts}
\getchunk{defun getConstructorOpsAndAtts}
\getchunk{defun getDomainsInScope}
\getchunk{defun getFormModemaps}
\getchunk{defun getFunctorOpsAndAtts}
\getchunk{defun getInverseEnvironment}

```



```

\getchunk{defun getModemap}
\getchunk{defun getModemapList}
\getchunk{defun getModemapListFromDomain}
\getchunk{defun getOperationAlist}
\getchunk{defun getScriptName}
\getchunk{defun getSignature}
\getchunk{defun getSignatureFromMode}
\getchunk{defun getSlotFromCategoryForm}
\getchunk{defun getSlotFromFunctor}
\getchunk{defun getSpecialCaseAssoc}
\getchunk{defun getSuccessEnvironment}
\getchunk{defun getTargetFromRhs}
\getchunk{defun get-token}
\getchunk{defun getToken}
\getchunk{defun getUnionMode}
\getchunk{defun getUniqueModemap}
\getchunk{defun getUniqueSignature}
\getchunk{defun genDomainOps}
\getchunk{defun genDomainViewList0}
\getchunk{defun genDomainViewList}
\getchunk{defun genDomainView}
\getchunk{defun giveFormalParametersValues}

\getchunk{defun hackforis}
\getchunk{defun hackforis1}
\getchunk{defun hasAplExtension}
\getchunk{defun hasFormalMapVariable}
\getchunk{defun hasFullSignature}
\getchunk{defun hasSigInTargetCategory}
\getchunk{defun hasType}

\getchunk{defun indent-pos}
\getchunk{defun infixtok}
\getchunk{defun initialize-prepare}
\getchunk{defun initial-substring}
\getchunk{defun initial-substring-p}
\getchunk{defun initializeLisplib}
\getchunk{defun insertModemap}
\getchunk{defun interactiveModemapForm}
\getchunk{defun is-console}
\getchunk{defun isDomainConstructorForm}
\getchunk{defun isDomainForm}
\getchunk{defun isDomainSubst}
\getchunk{defun isFunctor}
\getchunk{defun isListConstructor}
\getchunk{defun isMacro}
\getchunk{defun isSuperDomain}
\getchunk{defun isTokenDelimiter}
\getchunk{defun isUnionMode}

```

```

\getchunk{defun killColons}

\getchunk{defun line-advance-char}
\getchunk{defun line-at-end-p}
\getchunk{defun line-current-segment}
\getchunk{defun line-next-char}
\getchunk{defun line-past-end-p}
\getchunk{defun line-print}
\getchunk{defun line-new-line}
\getchunk{defun lispize}
\getchunk{defun lisplibDoRename}
\getchunk{defun lisplibWrite}
\getchunk{defun loadIfNecessary}
\getchunk{defun loadLibIfNecessary}

\getchunk{defun macroExpand}
\getchunk{defun macroExpandInPlace}
\getchunk{defun macroExpandList}
\getchunk{defun makeCategoryForm}
\getchunk{defun makeCategoryPredicates}
\getchunk{defun makeFunctorArgumentParameters}
\getchunk{defun makeSimplePredicateOrNil}
\getchunk{defun make-symbol-of}
\getchunk{defun match-advance-string}
\getchunk{defun match-current-token}
\getchunk{defun match-next-token}
\getchunk{defun match-string}
\getchunk{defun match-token}
\getchunk{defun maxSuperType}
\getchunk{defun mergeModemap}
\getchunk{defun mergeSignatureAndLocalVarALists}
\getchunk{defun meta-syntax-error}
\getchunk{defun mkAbbrev}
\getchunk{defun mkAlistOfExplicitCategoryOps}
\getchunk{defun mkCategoryPackage}
\getchunk{defun mkConstructor}
\getchunk{defun mkDatabasePred}
\getchunk{defun mkEvalableCategoryForm}
\getchunk{defun mkExplicitCategoryFunction}
\getchunk{defun mkList}
\getchunk{defun mkNewModemapList}
\getchunk{defun mkOpVec}
\getchunk{defun mkRepetitionAssoc}
\getchunk{defun mkUnion}
\getchunk{defun modifyModeStack}
\getchunk{defun modeEqual}
\getchunk{defun modeEqualSubst}
\getchunk{defun modemapPattern}
\getchunk{defun moveORsOutside}
\getchunk{defun mustInstantiate}

```

```

\getchunk{defun ncINTERPFILE}
\getchunk{defun next-char}
\getchunk{defun next-line}
\getchunk{defun next-tab-loc}
\getchunk{defun next-token}
\getchunk{defun new20ldLisp}
\getchunk{defun nonblankloc}

\getchunk{defun optCall}
\getchunk{defun optCallEval}
\getchunk{defun optCallSpecially}
\getchunk{defun optCatch}
\getchunk{defun optCond}
\getchunk{defun optCONDtail}
\getchunk{defun optEQ}
\getchunk{defun optIF2COND}
\getchunk{defun optimize}
\getchunk{defun optimizeFunctionDef}
\getchunk{defun optional}
\getchunk{defun optLESSP}
\getchunk{defun optMINUS}
\getchunk{defun optMkRecord}
\getchunk{defun optPackageCall}
\getchunk{defun optPredicateIfTrue}
\getchunk{defun optQSMINUS}
\getchunk{defun optRECORDCOPY}
\getchunk{defun optRECORDELT}
\getchunk{defun optSETRECORDELT}
\getchunk{defun optSEQ}
\getchunk{defun optSPADCALL}
\getchunk{defun optSpecialCall}
\getchunk{defun optSuchthat}
\getchunk{defun optXLAMCond}
\getchunk{defun opt-}
\getchunk{defun orderByDependency}
\getchunk{defun orderPredicateItems}
\getchunk{defun orderPredTran}
\getchunk{defun outputComp}

\getchunk{defun PARSE-AnyId}
\getchunk{defun PARSE-Application}
\getchunk{defun parse-argument-designator}
\getchunk{defun parse-identifier}
\getchunk{defun parse-keyword}
\getchunk{defun parse-number}
\getchunk{defun parse-spadstring}
\getchunk{defun parse-string}
\getchunk{defun PARSE-Category}
\getchunk{defun PARSE-Command}

```

```

\getchunk{defun PARSE-CommandTail}
\getchunk{defun PARSE-Conditional}
\getchunk{defun PARSE-Data}
\getchunk{defun PARSE-ElseClause}
\getchunk{defun PARSE-Enclosure}
\getchunk{defun PARSE-Exit}
\getchunk{defun PARSE-Expr}
\getchunk{defun PARSE-Expression}
\getchunk{defun PARSE-Float}
\getchunk{defun PARSE-FloatBase}
\getchunk{defun PARSE-FloatBasePart}
\getchunk{defun PARSE-FloatExponent}
\getchunk{defun PARSE-FloatTok}
\getchunk{defun PARSE-Form}
\getchunk{defun PARSE-FormalParameter}
\getchunk{defun PARSE-FormalParameterTok}
\getchunk{defun PARSE-getSemanticForm}
\getchunk{defun PARSE-GlyphTok}
\getchunk{defun PARSE-Import}
\getchunk{defun PARSE-Infix}
\getchunk{defun PARSE-InfixWith}
\getchunk{defun PARSE-IntegerTok}
\getchunk{defun PARSE-Iterator}
\getchunk{defun PARSE-IteratorTail}
\getchunk{defun PARSE-Label}
\getchunk{defun PARSE-LabelExpr}
\getchunk{defun PARSE-Leave}
\getchunk{defun PARSE-LedPart}
\getchunk{defun PARSE-leftBindingPowerOf}
\getchunk{defun PARSE-Loop}
\getchunk{defun PARSE-Name}
\getchunk{defun PARSE-NBGlyphTok}
\getchunk{defun PARSE-NewExpr}
\getchunk{defun PARSE-NudPart}
\getchunk{defun PARSE-OpenBrace}
\getchunk{defun PARSE-OpenBracket}
\getchunk{defun PARSE-Operation}
\getchunk{defun PARSE-Option}
\getchunk{defun PARSE-Prefix}
\getchunk{defun PARSE-Primary}
\getchunk{defun PARSE-Primary1}
\getchunk{defun PARSE-PrimaryNoFloat}
\getchunk{defun PARSE-PrimaryOrQM}
\getchunk{defun PARSE-Qualification}
\getchunk{defun PARSE-Quad}
\getchunk{defun PARSE-Reduction}
\getchunk{defun PARSE-ReductionOp}
\getchunk{defun PARSE-Return}
\getchunk{defun PARSE-rightBindingPowerOf}
\getchunk{defun PARSE-ScriptItem}

```

```

\getchunk{defun PARSE-Scripts}
\getchunk{defun PARSE-Seg}
\getchunk{defun PARSE-Selector}
\getchunk{defun PARSE-SemiColon}
\getchunk{defun PARSE-Sequence}
\getchunk{defun PARSE-Sequence1}
\getchunk{defun PARSE-Sexpr}
\getchunk{defun PARSE-Sexpr1}
\getchunk{defun PARSE-SpecialCommand}
\getchunk{defun PARSE-SpecialKeyWord}
\getchunk{defun PARSE-Statement}
\getchunk{defun PARSE-String}
\getchunk{defun PARSE-Suffix}
\getchunk{defun PARSE-TokenCommandTail}
\getchunk{defun PARSE-TokenList}
\getchunk{defun PARSE-TokenOption}
\getchunk{defun PARSE-TokTail}
\getchunk{defun PARSE-VarForm}
\getchunk{defun PARSE-With}
\getchunk{defun parsepiles}
\getchunk{defun parseAnd}
\getchunk{defun parseAtom}
\getchunk{defun parseAtSign}
\getchunk{defun parseCategory}
\getchunk{defun parseCoerce}
\getchunk{defun parseColon}
\getchunk{defun parseConstruct}
\getchunk{defun parseDEF}
\getchunk{defun parseDollarGreaterEqual}
\getchunk{defun parseDollarGreaterThan}
\getchunk{defun parseDollarLessEqual}
\getchunk{defun parseDollarNotEqual}
\getchunk{defun parseDropAssertions}
\getchunk{defun parseEquivalence}
\getchunk{defun parseExit}
\getchunk{defun postFlatten}
\getchunk{defun postFlattenLeft}
\getchunk{defun postForm}
\getchunk{defun parseGreaterEqual}
\getchunk{defun parseGreaterThan}
\getchunk{defun parseHas}
\getchunk{defun parseHasRhs}
\getchunk{defun parseIf}
\getchunk{defun parseIf,ifTran}
\getchunk{defun parseImplies}
\getchunk{defun parseIn}
\getchunk{defun parseInBy}
\getchunk{defun parseIs}
\getchunk{defun parseIsnt}
\getchunk{defun parseJoin}

```

```

\getchunk{defun parseLeave}
\getchunk{defun parseLessEqual}
\getchunk{defun parseLET}
\getchunk{defun parseLETD}
\getchunk{defun parseLhs}
\getchunk{defun parseMDEF}
\getchunk{defun parseNot}
\getchunk{defun parseNotEqual}
\getchunk{defun parseOr}
\getchunk{defun parsePretend}
\getchunk{defun parseprint}
\getchunk{defun parseReturn}
\getchunk{defun parseSegment}
\getchunk{defun parseSeq}
\getchunk{defun parseTran}
\getchunk{defun parseTranCheckForRecord}
\getchunk{defun parseTranList}
\getchunk{defun parseTransform}
\getchunk{defun parseType}
\getchunk{defun parseVCONS}
\getchunk{defun parseWhere}
\getchunk{defun Pop-Reduction}
\getchunk{defun postAdd}
\getchunk{defun postAtom}
\getchunk{defun postAtSign}
\getchunk{defun postBigFloat}
\getchunk{defun postBlock}
\getchunk{defun postBlockItem}
\getchunk{defun postBlockItemList}
\getchunk{defun postCapsule}
\getchunk{defun postCategory}
\getchunk{defun postcheck}
\getchunk{defun postCollect}
\getchunk{defun postCollect,finish}
\getchunk{defun postColon}
\getchunk{defun postColonColon}
\getchunk{defun postComma}
\getchunk{defun postConstruct}
\getchunk{defun postDef}
\getchunk{defun postDefArgs}
\getchunk{defun postError}
\getchunk{defun postExit}
\getchunk{defun postIf}
\getchunk{defun postin}
\getchunk{defun postIn}
\getchunk{defun postInSeq}
\getchunk{defun postIteratorList}
\getchunk{defun postJoin}
\getchunk{defun postMakeCons}
\getchunk{defun postMapping}

```

```

\getchunk{defun postMDef}
\getchunk{defun postOp}
\getchunk{defun postPretend}
\getchunk{defun postQUOTE}
\getchunk{defun postReduce}
\getchunk{defun postRepeat}
\getchunk{defun postScripts}
\getchunk{defun postScriptsForm}
\getchunk{defun postSemiColon}
\getchunk{defun postSignature}
\getchunk{defun postSlash}
\getchunk{defun postTran}
\getchunk{defun postTranList}
\getchunk{defun postTranScripts}
\getchunk{defun postTranSegment}
\getchunk{defun postTransform}
\getchunk{defun postTransformCheck}
\getchunk{defun postTuple}
\getchunk{defun postTupleCollect}
\getchunk{defun postType}
\getchunk{defun postWhere}
\getchunk{defun postWith}
\getchunk{defun print-package}
\getchunk{defun preparse}
\getchunk{defun preparse1}
\getchunk{defun preparse-echo}
\getchunk{defun preparseReadLine}
\getchunk{defun preparseReadLine1}
\getchunk{defun primitiveType}
\getchunk{defun print-defun}
\getchunk{defun processFunctor}
\getchunk{defun push-reduction}
\getchunk{defun putDomainsInScope}
\getchunk{defun putInLocalDomainReferences}

\getchunk{defun quote-if-string}

\getchunk{defun read-a-line}
\getchunk{defun recompile-lib-file-if-necessary}
\getchunk{defun replaceExitEtc}
\getchunk{defun removeSuperfluousMapping}
\getchunk{defun replaceVars}
\getchunk{defun resolve}
\getchunk{defun reportOnFunctorCompilation}
\getchunk{defun /rf-1}
\getchunk{defun /RQ,LIB}
\getchunk{defun rwriteLispForm}

\getchunk{defun setDefOp}
\getchunk{defun seteltModemapFilter}

```

```

\getchunk{defun setqMultiple}
\getchunk{defun setqMultipleExplicit}
\getchunk{defun setqSetelt}
\getchunk{defun setqSingle}
\getchunk{defun signatureTran}
\getchunk{defun skip-blanks}
\getchunk{defun skip-ifblock}
\getchunk{defun skip-to-endif}
\getchunk{defun spad}
\getchunk{defun spadCompileOrSetq}
\getchunk{defun spad-fixed-arg}
\getchunk{defun splitEncodedFunctionName}
\getchunk{defun stack-clear}
\getchunk{defun stack-load}
\getchunk{defun stack-pop}
\getchunk{defun stack-push}
\getchunk{defun storeblanks}
\getchunk{defun stripOffArgumentConditions}
\getchunk{defun stripOffSubdomainConditions}
\getchunk{defun subrname}
\getchunk{defun substituteCategoryArguments}
\getchunk{defun substNames}
\getchunk{defun substVars}
\getchunk{defun s-process}

\getchunk{defun token-install}
\getchunk{defun token-lookahead-type}
\getchunk{defun token-print}
\getchunk{defun transformOperationAlist}
\getchunk{defun transIs}
\getchunk{defun transIs1}
\getchunk{defun translablel}
\getchunk{defun translablel1}
\getchunk{defun TruthP}
\getchunk{defun try-get-token}
\getchunk{defun tuple2List}

\getchunk{defun underscore}
\getchunk{defun unget-tokens}
\getchunk{defun unknownTypeError}
\getchunk{defun uncons}
\getchunk{defun unTuple}
\getchunk{defun updateCategoryFrameForCategory}
\getchunk{defun updateCategoryFrameForConstructor}

\getchunk{defun wrapDomainSub}
\getchunk{defun writeLib1}

\getchunk{postvars}

```


Bibliography

- [1] Jenks, R.J. and Sutor, R.S. “Axiom – The Scientific Computation System” Springer-Verlag New York (1992) ISBN 0-387-97855-0
- [2] Knuth, Donald E., “Literate Programming” Center for the Study of Language and Information ISBN 0-937073-81-4 Stanford CA (1992)
- [3] Daly, Timothy, “The Axiom Wiki Website”
<http://axiom.axiom-developer.org>
- [4] Watt, Stephen, “Aldor”,
<http://www.aldor.org>
- [5] Lamport, Leslie, “Latex – A Document Preparation System”, Addison-Wesley, New York ISBN 0-201-52983-1
- [6] Ramsey, Norman “Noweb – A Simple, Extensible Tool for Literate Programming”
<http://www.eecs.harvard.edu/~nr/noweb>
- [7] Daly, Timothy, ”The Axiom Literate Documentation”
<http://axiom.axiom-developer.org/axiom-website/documentation.html>
- [8] Pratt, Vaughn “Top down operator precedence” POPL ’73 Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages
hall.org.ua/halls/wizzard/pdf/Vaughan.Pratt.TDOP.pdf
- [9] Floyd, R. W. “Semantic Analysis and Operator Precedence” JACM 10, 3, 316-333 (1963)

Chapter 14

Index

Index

- `+- >`, 307
 - `defplist`, 307
- `- >`, 376
 - `defplist`, 376
- `<=`, 121
 - `defplist`, 121
- `==>`, 377
 - `defplist`, 377
- `=>`, 373
 - `defplist`, 373
- `>`, 108
 - `defplist`, 108
- `>=`, 106, 107
 - `defplist`, 106, 107
- `*comp370-apply*`
 - usedby `spad`, 489
- `*eof*`
 - usedby `read-a-line`, 533
 - usedby `spad`, 489
- `*fileactq-apply*`
 - usedby `spad`, 489
- `*lisp-bin-filetype*`
 - usedby `recompile-lib-file-if-necessary`, 529
- `*standard-output*`
 - usedby `compileFileQuietly`, 530
- `*terminal-io*`
 - usedby `is-console`, 467
- `., 368`
 - `defplist`, 368
- `-, 217`
 - `defplist`, 217
- `/, 383`
 - `defplist`, 383
- `/RQ,LIB`, 479
 - calledby `compilerDoit`, 478
 - calls `/rf-1`, 479
 - calls `echo-meta[5]`, 479
 - uses `$lisplib`, 479
 - defun, 479
- `/editfile`
 - usedby `/rf-1`, 480
 - usedby `compAdd`, 248
 - usedby `compFunctorBody`, 191
 - usedby `compileSpad2Cmd`, 476
 - usedby `compiler`, 473
 - usedby `initializeLisplib`, 173
 - usedby `spad`, 489
- `/major-version`
 - usedby `initializeLisplib`, 173
- `/rf-1`, 480
 - calledby `/RQ,LIB`, 479
 - calls `makeInputFilename[5]`, 480
 - calls `ncINTERPFILE`, 480
 - calls `spad[5]`, 480
 - uses `/editfile`, 480
 - uses `echo-meta`, 480
 - defun, 480
- `/rf[5]`
 - called by `compilerDoit`, 478
- `/rq[5]`
 - called by `compilerDoit`, 478
- `., 100, 266, 366`
 - `defplist`, 100, 266, 366
- `::, 99, 343, 367`
 - `defplist`, 99, 343, 367
- `:BF:, 361`
 - `defplist`, 361
- `;;, 381`
 - `defplist`, 381
- `==, 370`
 - `defplist`, 370
- `$*eof*`
 - local ref `preparseReadLine`, 85
- `$/editfile`

- local ref finalizeLisplib, 174
- \$BasicPredicates, 207
 - local ref optPredicateIfTrue, 207
 - defvar, 207
- \$Boolean
 - local ref compArgumentConditions, 286
 - local ref compHas, 294
 - local ref doItIf, 257
 - usedby compCase1, 260
 - usedby compIf, 297
 - usedby compIs, 304
 - usedby compReduce1, 312
 - usedby compRepeatOrCollect, 315
 - usedby compSubDomain1, 332
 - usedby compSuchthat, 334
 - usedby compSymbol, 505
- \$CapsuleDomainsInScope
 - local def compDefineCapsuleFunction, 281
 - local def putDomainsInScope, 231
 - local ref getDomainsInScope, 230
- \$CapsuleModemapFrame
 - local def addModemapKnown, 245
 - local def addModemap, 245
 - local def compDefineCapsuleFunction, 280
 - local ref addModemap, 245
- \$CategoryFrame
 - local def updateCategoryFrameForCategory, 113
 - local def updateCategoryFrameForConstructor, 112
 - local ref isFunctor, 229
 - local ref loadLibIfNecessary, 111
 - local ref mkEvaluableCategoryForm, 141
 - local ref updateCategoryFrameForCategory, 113
 - local ref updateCategoryFrameForConstructor, 112
 - usedby compDefineFunctor1, 182
 - usedby compSubDomain1, 332
 - usedby compWithMappingModel1, 518
 - usedby parseHasRhs, 110
 - usedby parseHas, 109
- \$CategoryNames
 - local ref mkEvaluableCategoryForm, 141
- \$Category
 - local ref augModemapsFromDomain, 232
 - local ref compMakeCategoryObject, 180
 - local ref mkEvaluableCategoryForm, 141
 - usedby compConstructorCategory, 274
 - usedby compDefine1, 275
 - usedby compJoin, 305
- \$CheckVectorList
 - usedby compDefineFunctor1, 182
 - usedby displayMissingFunctions, 193
- \$ConditionalOperators
 - usedby genDomainOps, 198
 - usedby makeFunctorArgumentParameters, 194
- \$ConstructorCache
 - local ref compileConstructor1, 153
- \$ConstructorNames
 - usedby compDefine1, 275
- \$DomainFrame
 - usedby s-process, 491
- \$DomainsInScope
 - local ref compDefineCapsuleFunction, 280
- \$DoubleFloat
 - usedby primitiveType, 504
- \$DummyFunctorNames
 - local ref augModemapsFromDomain, 232
 - local ref mustInstantiate, 266
- \$EchoLineStack
 - usedby fincomblock, 466
 - usedby preparse-echo, 88
 - usedby preparseReadLine1, 87
- \$EchoLines
 - usedby ncINTERPFILE, 527
- \$EmptyEnvironment
 - local ref augLisplibModemapsFromCategory, 155
 - local ref compHasFormat, 295
 - local ref getInverseEnvironment, 302
 - local ref getSuccessEnvironment, 300
 - usedby genDomainViewList, 196
 - usedby s-process, 491
- \$EmptyMode, 131
 - local ref coerceEasy, 338
 - local ref compHasFormat, 295
 - local ref compileDocumentation, 172
 - local ref doIt, 253
 - local ref getSuccessEnvironment, 300
 - local ref makeCategoryForm, 270

- local ref mkEvaluableCategoryForm, 141
- local ref resolve, 347
- local ref setqMultipleExplicit, 325
- local ref setqMultiple, 323
- usedby compAdd, 248
- usedby compArgumentsAndTryAgain, 517
- usedby compCase1, 260
- usedby compColonInside, 502
- usedby compCons1, 271
- usedby compDefine1, 275
- usedby compDefineAddSignature, 133
- usedby compDefineCategory1, 137
- usedby compForm1, 508
- usedby compForm2, 513
- usedby compIs, 304
- usedby compMacro, 309
- usedby compNoStacking, 498
- usedby compPretend, 311
- usedby compSetq1, 321
- usedby compSubDomain1, 332
- usedby compWhere, 336
- usedby compWithMappingMode1, 518
- usedby primitiveType, 505
- usedby s-process, 491
- usedby setqSingle, 326
- defvar, 131
- \$EmptyVector
 - usedby compVector, 335
- \$Exit
 - local ref coerceEasy, 338
- \$Expression
 - local ref coerceExtraHard, 340
 - local ref compExpressionList, 512
 - local ref outputComp, 328
 - usedby compAtom, 503
 - usedby compForm1, 508
 - usedby compSymbol, 505
- \$FormalMapVariableList, 242
 - local ref compDefineCategory2, 142
 - local ref compHasFormat, 295
 - local ref finalizeLisplib, 175
 - local ref getSignatureFromMode, 279
 - local ref getSlotFromCategoryForm, 177
 - local ref interactiveModemapForm, 158
 - local ref isDomainConstructorForm, 330
 - local ref substVars, 166
 - usedby compColon, 267
 - usedby compDefineFunctor1, 182
 - usedby compSymbol, 505
 - usedby compTypeOf, 502
 - usedby compWithMappingMode1, 518
 - usedby makeCategoryPredicates, 138
 - usedby mkCategoryPackage, 139
 - usedby mkOpVec, 199
 - usedby substNames, 243
 - defvar, 242
- \$GensymAssoc
 - local def EqualBarGensym, 224
 - local ref EqualBarGensym, 224
- \$Index
 - usedby s-process, 491
- \$InitialDomainsInScope
 - usedby spad, 489
- \$InteractiveFrame
 - usedby s-process, 491
 - usedby spad, 489
- \$InteractiveMode
 - local def addConstructorModemaps, 234
 - local ref addModemap, 245
 - local ref coerce, 337
 - local ref displayPreCompilationErrors, 456
 - local ref isFunctor, 229
 - local ref loadLibIfNecessary, 111
 - local ref mkNewModemapList, 238
 - local ref optCatch, 220
 - local ref optSPADCALL, 219
 - usedby bumperrorcount, 457
 - usedby compileFileQuietly, 530
 - usedby compileSpad2Cmd, 476
 - usedby dollarTran, 451
 - usedby parseAnd, 97
 - usedby parseAtSign, 98
 - usedby parseCoerce, 100
 - usedby parseColon, 100
 - usedby parseHas, 109
 - usedby parseIf,ifTran, 114
 - usedby parseNot, 124
 - usedby postBigFloat, 362
 - usedby postDef, 371
 - usedby postError, 356
 - usedby postMDef, 377
 - usedby postReduce, 379

- usedby spad, 489
- usedby tuple2List, 462
- \$LocalDomainAlist
 - local def doIt, 254
 - local ref doIt, 253
 - usedby compDefineFunctor1, 182
- \$LocalFrame
 - usedby s-process, 491
- \$NRTaddForm
 - usedby compAdd, 248
 - usedby compDefineFunctor1, 182
 - usedby compFuncBody, 191
 - usedby compSubDomain, 331
- \$NRTaddList
 - usedby compDefineFunctor1, 182
- \$NRTattributeAlist
 - usedby compDefineFunctor1, 182
- \$NRTbase
 - usedby compDefineFunctor1, 182
- \$NRTdeltaLength
 - usedby compDefineFunctor1, 182
- \$NRTdeltaListComp
 - usedby compDefineFunctor1, 182
- \$NRTdeltaList
 - usedby compDefineFunctor1, 182
- \$NRTderivedTargetIfTrue
 - usedby compTopLevel, 494
- \$NRTdomainFormList
 - usedby compDefineFunctor1, 182
- \$NRTfloadTimeAlist
 - usedby compDefineFunctor1, 182
- \$NRTopt
 - local ref doIt, 253
- \$NRTslot1Info
 - usedby compDefineFunctor1, 183
- \$NRTslot1PredicateList
 - local def finalizeLisplib, 175
 - usedby compDefineFunctor1, 183
- \$NegativeInteger
 - usedby primitiveType, 505
- \$NoValueMode, 131
 - local ref coerceEasy, 338
 - local ref resolve, 347
 - local ref setqMultipleExplicit, 325
 - local ref setqMultiple, 323
 - usedby compDefine1, 275
- usedby compImport, 304
- usedby compMacro, 309
- usedby compRepeatOrCollect, 315
- usedby compSeq1, 318
- usedby compSymbol, 505
- usedby setqSingle, 326
- defvar, 131
- \$NoValue
 - usedby compSymbol, 505
 - usedby parseAtom, 94
- \$NonMentionableDomainNames
 - local ref doIt, 253
- \$NonNegativeInteger
 - usedby primitiveType, 505
- \$NumberOfArgsIfInteger
 - usedby compForm1, 508
- \$One
 - usedby compElt, 292
- \$PatternVariableList
 - local ref augLisplibModemapsFromCategory, 155
 - local ref augmentLisplibModemapsFromFunctor, 189
 - local ref formal2Pattern, 190
 - local ref interactiveModemapForm, 158
 - local ref modemapPattern, 167
- \$PolyMode
 - usedby s-process, 491
- \$PositiveInteger
 - usedby primitiveType, 505
- \$PrettyPrint
 - usedby print-defun, 493
- \$PrintOnly
 - usedby s-process, 491
- \$QuickCode
 - local ref doIt, 254
 - local ref optCall, 209
 - local ref optSpecialCall, 212
 - local ref putInLocalDomainReferences, 154
 - usedby compDefineFunctor1, 183
 - usedby compWithMappingMode1, 518
 - usedby compileSpad2Cmd, 476
- \$QuickLet
 - usedby compileSpad2Cmd, 476
 - usedby setqSingle, 326
- \$ReadingFile

- usedby ncINTERPFILE, 527
- \$Representation**
 - local def doIt, 254
 - local ref doIt, 253
 - usedby compDefineFunctor1, 183
 - usedby compNoStacking, 498
- \$Rep**
 - local ref coerce, 337
 - local ref mkUnion, 347
- \$SpecialDomainNames**
 - local ref isDomainForm, 329
 - usedby addEmptyCapsuleIfNecessary, 134
- \$StringCategory**
 - usedby compString, 331
- \$String**
 - local ref coerceHard, 339
 - local ref resolve, 347
 - usedby primitiveType, 505
- \$Symbol**
 - usedby compSymbol, 505
- \$TranslateOnly**
 - usedby s-process, 491
- \$Translation**
 - usedby s-process, 491
- \$TriangleVariableList**
 - local ref compDefineCategory2, 142
 - usedby compForm2, 513
 - usedby makeCategoryPredicates, 138
- \$Undef**
 - local ref optSpecialCall, 212
- \$VariableCount**
 - usedby s-process, 491
- \$Void**
 - local ref coerceEasy, 338
- \$Zero**
 - usedby compElt, 292
- \$abbreviationTable**
 - local def getAbbreviation, 277
 - local ref getAbbreviation, 277
- \$addFormLhs**
 - usedby compAdd, 248
 - usedby compSubDomain, 331
- \$addForm**
 - local def compDefineCategory2, 143
 - usedby compAdd, 248
 - usedby compCapsuleInner, 251
- usedby compDefineFunctor1, 183
- usedby compSubDomain, 331
- \$algebraOutputStream**
 - local ref compDefineLisplib, 170
- \$alternateViewList**
 - usedby makeFunctorArgumentParameters, 194
- \$argumentConditionList**
 - local def addArgumentConditions, 285
 - local def compArgumentConditions, 286
 - local def compDefineCapsuleFunction, 280
 - local def stripOffArgumentConditions, 288
 - local def stripOffSubdomainConditions, 287
 - local ref addArgumentConditions, 285
 - local ref compArgumentConditions, 286
 - local ref stripOffArgumentConditions, 287
 - local ref stripOffSubdomainConditions, 287
- \$atList**
 - local def compCategory, 262
 - local ref compCategoryItem, 263
 - local ref compCategory, 262
- \$attributesName**
 - usedby compDefineFunctor1, 183
- \$base**
 - local def augModemapsFromCategoryRep, 244
 - local def augModemapsFromCategory, 237
- \$body**
 - local ref addArgumentConditions, 285
- \$bootStrapMode**
 - local ref coerceHard, 339
 - local ref optCall, 209
 - usedby comp2, 499
 - usedby compAdd, 248
 - usedby compCapsule, 250
 - usedby compColon, 267
 - usedby compDefineCategory1, 137
 - usedby compDefineFunctor1, 182
 - usedby compFunctorBody, 191
- \$boot**
 - local ref PARSE-FloatTok, 439
 - local ref PARSE-Primary1, 421
 - usedby PARSE-Quad, 425
 - usedby PARSE-Selector, 419

- usedby PARSE-TokTail, 416
- usedby aplTran1, 387
- usedby aplTran, 387
- usedby postAtom, 353
- usedby postBigFloat, 362
- usedby postColonColon, 367
- usedby postDef, 371
- usedby postForm, 356
- usedby postIf, 373
- usedby postMDef, 377
- usedby quote-if-string, 442
- usedby spad, 489
- usedby tuple2List, 462
- \$byConstructors, 531
 - local ref prepare1, 81
 - usedby compilerDoit, 478
 - defvar, 531
- \$byteAddress
 - usedby compDefineFunctor1, 183
- \$byteVec
 - usedby compDefineFunctor1, 183
- \$categoryPredicateList
 - usedby compDefineCategory1, 137
 - usedby mkCategoryPackage, 139
- \$clamList
 - local def compileConstructor1, 153
 - local ref compileConstructor1, 153
- \$comblocklist, 465
 - usedby fincomblock, 466
 - usedby prepare, 77
 - defvar, 465
- \$compErrorMessageStack
 - usedby compOrCroak1, 497
- \$compForModeIfTrue
 - local def compForMode, 307
 - usedby compSymbol, 505
- \$compStack
 - local def compOrCroak1, 497
 - local ref compNoStacking1, 498
 - local ref compNoStacking, 498
 - local ref comp, 498
- \$compTimeSum
 - usedby compTopLevel, 494
- \$compUniquelyIfTrue
 - local def compUniquely, 516
 - local ref compForm3, 515
- usedby s-process, 491
- \$compileDocumentation
 - local ref compDefineLisplib, 170
- \$compileOnlyCertainItems
 - local ref compDefineCapsuleFunction, 280
 - local ref compile, 146
 - usedby compDefineFunctor1, 183
 - usedby compileSpad2Cmd, 476
- \$condAList
 - usedby compDefineFunctor1, 183
- \$constructorLineNumber
 - usedby prepare, 77
- \$constructorsSeen, 531
 - local ref prepare1, 81
 - usedby compilerDoit, 478
 - defvar, 531
- \$currentFunction
 - usedby s-process, 491
- \$currentLine
 - usedby s-process, 491
- \$defOp
 - usedby parseTransform, 93
 - usedby postError, 356
 - usedby postTransformCheck, 355
 - usedby setDefOp, 386
- \$definition
 - local def compDefineCategory2, 142
 - local ref compDefineCategory2, 142
- \$defstack, 393
 - defvar, 393
- \$doNotCompileJustPrint
 - local ref compile, 146
- \$docList
 - usedby postDef, 371
 - usedby prepare, 77
- \$domainShell
 - local def compDefineCategory2, 143
 - local ref augLisplibModemapsFromCategory, 155
 - local ref hasSigInTargetCategory, 290
 - usedby compDefineCategory, 169
 - usedby compDefineFunctor1, 183
 - usedby compDefineFunctor, 181
 - usedby getOperationAlist, 242
- \$echolinestack, 72
 - local ref prepare1, 81

- usedby initialize-prepare, 73
- defvar, 72
- \$elt
 - local def putInLocalDomainReferences, 154
- \$endTestList
 - usedby compReduce1, 312
- \$envHashTable
 - usedby compTopLevel, 494
- \$env
 - usedby displayMissingFunctions, 193
- \$erase
 - local ref initializeLisplib, 173
- \$exitModeStack
 - usedby compExit, 293
 - usedby compLeave, 308
 - usedby compOrCroak1, 497
 - usedby compRepeatOrCollect, 315
 - usedby compReturn, 317
 - usedby compSeq1, 318
 - usedby compSeq, 318
 - usedby comp, 498
 - usedby modifyModeStack, 525
 - usedby s-process, 491
- \$exitMode
 - local ref coerceExit, 342
 - usedby s-process, 491
- \$extraParms
 - local def compDefineCategory2, 142
 - local ref compDefineCategory2, 142
- \$e
 - local def addEltModemap, 237
 - local def augmentLisplibModemapsFrom-Functor, 189
 - local def coerceHard, 339
 - local def compCapsuleItems, 252
 - local def compHas, 294
 - local def doItIf, 257
 - local def doIt, 254
 - local def mkEvaluableCategoryForm, 141
 - local ref addModemapKnown, 245
 - local ref addModemap, 245
 - local ref augmentLisplibModemapsFrom-Functor, 189
 - local ref coerceHard, 339
 - local ref compCapsuleItems, 252
 - local ref compHasFormat, 295
 - local ref compHas, 294
 - local ref compMakeCategoryObject, 180
 - local ref compSingleCapsuleItem, 252
 - local ref compileDocumentation, 172
 - local ref compile, 146
 - local ref doItIf, 257
 - local ref doIt, 253
 - local ref getSignature, 288
 - local ref getSlotFromFunctor, 179
 - local ref mkAlistOfExplicitCategoryOps, 156
 - local ref mkDatabasePred, 191
 - local ref mkEvaluableCategoryForm, 141
 - local ref optCallSpecially, 211
 - local ref signatureTran, 161
 - usedby comp3, 500
 - usedby compReduce1, 312
 - usedby genDomainOps, 198
 - usedby genDomainView, 197
 - usedby getOperationAlist, 242
 - usedby mkCategoryPackage, 139
 - usedby s-process, 491
- \$fcopy
 - local ref compileDocumentation, 172
- \$filep
 - local ref compDefineLisplib, 170
- \$finalEnv
 - local def compDefineCapsuleFunction, 280
 - local def replaceExitEtc, 319
 - local ref replaceExitEtc, 319
 - usedby compSeq1, 318
- \$forceAdd
 - local ref mergeModemap, 239
 - local ref mkNewModemapList, 238
 - usedby compTopLevel, 494
 - usedby makeFunctorArgumentParameters, 194
- \$formalArgList
 - local def compDefineCapsuleFunction, 281
 - local def compDefineCategory2, 142
 - local ref compDefineCapsuleFunction, 280
 - local ref compDefineCategory2, 142
 - usedby compDefine1, 275
 - usedby compReduce, 312
 - usedby compRepeatOrCollect, 315

- usedby compSymbol, 505
- usedby compWithMappingMode1, 518
- usedby compWithMappingMode, 517
- usedby displayMissingFunctions, 193
- \$formalMapVariables
 - local def hasFormalMapVariable, 524
- \$formatArgList
 - usedby compWithMappingMode1, 518
- \$form
 - local def compDefineCapsuleFunction, 280
 - local def compDefineCategory2, 142
 - local ref compDefineCategory2, 142
 - local ref compHasFormat, 295
 - usedby compCapsuleInner, 251
 - usedby compDefine1, 275
 - usedby compDefineFunctor1, 183
 - usedby s-process, 491
 - usedby setqSingle, 326
- \$fromCoerceable
 - local ref autoCoerceByModemap, 346
 - local ref coerceable, 342
 - local ref coerce, 337
- \$frontier
 - local def compDefineCategory2, 142
- \$functionLocations
 - local def compDefineCapsuleFunction, 281
 - local ref compDefineCapsuleFunction, 280
 - local ref transformOperationAlist, 178
 - usedby compDefineFunctor1, 183
- \$functionName
 - local ref addArgumentConditions, 285
- \$functionStats
 - local def compDefineCapsuleFunction, 280
 - local def compDefineCategory2, 142
 - local def compile, 146
 - local ref compDefineCapsuleFunction, 280
 - local ref compile, 146
 - usedby compDefineFunctor1, 183
 - usedby reportOnFunctorCompilation, 193
- \$functorForm
 - local def compDefineCategory2, 143
 - local ref addModemap0, 246
 - local ref compile, 146
 - local ref getSpecialCaseAssoc, 285
 - usedby compAdd, 248
 - usedby compCapsule, 250
 - usedby compDefineFunctor1, 183
 - usedby compFunctorBody, 191
 - usedby getOperationAlist, 242
- \$functorLocalParameters
 - local def doItIf, 257
 - local def doIt, 254
 - local ref doItIf, 257
 - local ref doIt, 253
 - usedby compCapsuleInner, 251
 - usedby compDefineFunctor1, 183
 - usedby compSymbol, 505
- \$functorSpecialCases
 - local ref getSpecialCaseAssoc, 285
 - usedby compDefineFunctor1, 183
- \$functorStats
 - local def compDefineCategory2, 142
 - local ref compDefineCapsuleFunction, 280
 - usedby compDefineFunctor1, 183
 - usedby reportOnFunctorCompilation, 193
- \$functorTarget
 - usedby compDefineFunctor1, 183
- \$functorsUsed
 - local def doIt, 254
 - local ref doIt, 253
 - usedby compDefineFunctor1, 183
- \$funnameTail
 - usedby compWithMappingMode1, 518
- \$funname
 - usedby compWithMappingMode1, 518
- \$f
 - usedby compileSpad2Cmd, 476
- \$genFVar
 - usedby compDefineFunctor1, 183
 - usedby s-process, 491
- \$genSDVar
 - usedby compDefineFunctor1, 183
 - usedby s-process, 491
- \$genno
 - local def aplTran, 387
 - local def doIt, 254
- \$getDomainCode
 - local def compDefineCategory2, 142
 - local ref compileCases, 283
 - local ref doItIf, 257
 - local ref optCallSpecially, 211
 - usedby compCapsuleInner, 251

- usedby compDefineFunctor1, 183
 - usedby genDomainOps, 198
 - usedby genDomainView, 197
- \$goGetList
 - usedby compDefineFunctor1, 183
- \$headerDocumentation
 - usedby postDef, 371
 - usedby preparse, 77
- \$in-stream
 - local ref Advance-Char, 539
 - local ref next-line, 538
 - local ref preparse1, 81
- \$index, 72
 - local def preparse1, 81
 - local ref preparse1, 81
 - usedby initialize-preparse, 73
 - usedby preparseReadLine1, 87
 - usedby preparse, 77
 - defvar, 72
- \$initCapsuleErrorCount
 - local def compDefineCapsuleFunction, 280
- \$initList
 - usedby compReduce1, 312
- \$insideCapsuleFunctionIfTrue
 - local def compDefineCapsuleFunction, 280
 - local ref CapsuleModemapFrame, 245
 - local ref addEltModemap, 237
 - local ref addModemap, 245
 - local ref compile, 146
 - local ref getDomainsInScope, 230
 - local ref putDomainsInScope, 231
 - local ref spadCompileOrSetq, 152
 - usedby compDefine1, 275
 - usedby s-process, 491
- \$insideCategoryIfTrue
 - local def compDefineCategory2, 142
 - usedby compCapsuleInner, 251
 - usedby compColon, 267
 - usedby compDefine1, 275
 - usedby s-process, 491
- \$insideCategoryPackageIfTrue
 - local ref getFormModemaps, 510
 - usedby compCapsuleInner, 251
 - usedby compDefineCategory1, 137
 - usedby compDefineFunctor1, 183
- \$insideCoerceInteractiveHardIfTrue
 - usedby s-process, 491
- \$insideCompTypeOf
 - usedby comp3, 500
 - usedby compTypeOf, 502
- \$insideConstructIfTrue
 - local ref parseColon, 100
 - usedby parseConstruct, 95
- \$insideExpressionIfTrue
 - local def compDefineCapsuleFunction, 281
 - usedby compCapsule, 250
 - usedby compColon, 267
 - usedby compDefine1, 275
 - usedby compExpression, 507
 - usedby compMakeDeclaration, 525
 - usedby compSeq1, 318
 - usedby compWhere, 336
 - usedby s-process, 491
- \$insideFunctorIfTrue
 - local ref compileCases, 283
 - usedby compColon, 267
 - usedby compDefine1, 275
 - usedby compDefineCategory, 169
 - usedby compDefineFunctor1, 183
 - usedby getOperationAlist, 242
 - usedby s-process, 491
- \$insidePostCategoryIfTrue
 - usedby postCategory, 363
 - usedby postWith, 386
- \$insideSetqSingleIfTrue
 - usedby setqSingle, 326
- \$insideWhereIfTrue
 - usedby compDefine1, 275
 - usedby compWhere, 336
 - usedby s-process, 491
- \$is-eqlist, 394
 - defvar, 394
- \$is-gensymlist, 394
 - defvar, 394
- \$is-spill-list, 393
 - defvar, 393
- \$is-spill, 393
 - defvar, 393
- \$isOpPackageName
 - usedby compDefineFunctor1, 183
- \$keywords
 - local ref escape-keywords, 443

- \$skillOptimizeIfTrue
 - usedby compTopLevel, 494
 - usedby compWithMappingMode1, 518
- \$leaveLevelStack
 - usedby compLeave, 309
 - usedby compRepeatOrCollect, 315
 - usedby s-process, 491
- \$leaveMode
 - usedby s-process, 491
- \$level
 - usedby compOrCroak1, 497
- \$lhsOfColon
 - local def evalAndSub, 241
 - usedby compColon, 267
 - usedby compSubsetCategory, 333
- \$lhs
 - usedby parseDEF, 101
 - usedby parseMDEF, 123
- \$libFile
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref compDefineCategory2, 142
 - local ref compilerDoitWithScreenedLisplib, 349
 - local ref finalizeLisplib, 174
 - local ref initializeLisplib, 173
 - local ref rwriteLispForm, 168
 - usedby compDefineFunctor1, 183
- \$line-handler
 - local ref next-line, 538
- \$linelist, 72
 - local ref preparse1, 81
 - usedby initialize-preparse, 73
 - usedby preparseReadLine1, 87
 - defvar, 72
- \$line
 - usedby Advance-Char, 539
 - usedby current-char, 449
 - usedby line-advance-char, 537
 - usedby line-at-end-p, 536
 - usedby line-clear, 536
 - usedby line-new-line, 538
 - usedby line-next-char, 537
 - usedby line-past-end-p, 537
 - usedby line-print, 536, 538
 - usedby match-advance-string, 441
 - usedby match-string, 440
- \$lisplibAbbreviation
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref finalizeLisplib, 175
 - usedby compDefineFunctor1, 183
- \$lisplibAncestors
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref finalizeLisplib, 175
 - usedby compDefineFunctor1, 183
- \$lisplibAttributes
 - local ref finalizeLisplib, 175
- \$lisplibCategoriesExtended
 - local def compDefineLisplib, 170
 - usedby compDefineFunctor1, 183
- \$lisplibCategory
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 170
 - local def finalizeLisplib, 175
 - local ref compDefineCategory2, 142
 - local ref finalizeLisplib, 174
 - usedby compDefineCategory1, 137
 - usedby compDefineCategory, 169
 - usedby compDefineFunctor1, 183
- \$lisplibForm
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref finalizeLisplib, 174, 175
 - usedby compDefineFunctor1, 183
- \$lisplibFunctionLocations
 - usedby compDefineFunctor1, 184
- \$lisplibItemsAlreadyThere
 - local ref compile, 146
- \$lisplibKind
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref compDefineLisplib, 170
 - local ref finalizeLisplib, 174
 - usedby compDefineFunctor1, 183
- \$lisplibMissingFunctions
 - usedby compDefineFunctor1, 183

- \$lisplibModemapAlist
 - local def augLisplibModemapsFromCategory, 155
 - local def augmentLisplibModemapsFromFunctor, 189
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref augLisplibModemapsFromCategory, 155
 - local ref augmentLisplibModemapsFromFunctor, 189
 - local ref finalizeLisplib, 175
- \$lisplibModemap
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref finalizeLisplib, 174, 175
 - usedby compDefineFunctor1, 183
- \$lisplibOpAlist
 - local def initializeLisplib, 173
- \$lisplibOperationAlist
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref getSlotFromFunctor, 179
 - usedby compDefineFunctor1, 183
- \$lisplibParents
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 170
 - local ref finalizeLisplib, 175
 - usedby compDefineFunctor1, 183
- \$lisplibPredicates
 - local def compDefineLisplib, 170
 - local ref finalizeLisplib, 175
- \$lisplibSignatureAlist
 - local def encodeFunctionName, 148
 - local def initializeLisplib, 173
 - local ref encodeFunctionName, 148
 - local ref finalizeLisplib, 175
- \$lisplibSlot1
 - local def compDefineLisplib, 170
 - local ref finalizeLisplib, 175
 - usedby compDefineFunctor1, 183
- \$lisplibSuperDomain
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref finalizeLisplib, 175
- usedby compSubDomain1, 332
- \$lisplibVariableAlist
 - local def compDefineLisplib, 170
 - local def initializeLisplib, 173
 - local ref finalizeLisplib, 175
- \$lisplib
 - local def compDefineLisplib, 170
 - local ref compDefineCategory2, 142
 - local ref compile, 146
 - local ref encodeFunctionName, 148
 - local ref lisplibWrite, 181
 - local ref rwriteLispForm, 168
 - usedby /RQ,LIB, 479
 - usedby comp2, 499
 - usedby compDefineCategory, 169
 - usedby compDefineFunctor1, 182
 - usedby compDefineFunctor, 181
- \$lookupFunction
 - usedby compDefineFunctor1, 183
- \$macroIfTrue
 - local ref compile, 146
 - usedby compDefine, 274
 - usedby compMacro, 309
- \$macroassoc
 - usedby s-process, 491
- \$maxSignatureLineNumber
 - usedby postDef, 371
 - usedby prepare, 77
- \$mutableDomains
 - usedby compDefineFunctor1, 183
- \$mutableDomain
 - local ref compileConstructor1, 153
 - usedby compDefineFunctor1, 183
- \$mvl
 - usedby makeCategoryPredicates, 138
- \$myFunctorBody
 - local def compCapsuleItems, 252
 - usedby compDefineFunctor1, 183
- \$m
 - usedby compileSpad2Cmd, 476
- \$newCompilerUnionFlag
 - usedby compColonInside, 502
 - usedby compPretend, 311
- \$newComp
 - usedby compileSpad2Cmd, 476
- \$newConlist

- local def compDefineLisplib, 170
 - local ref compDefineLisplib, 170
 - usedby compileSpad2Cmd, 476
 - usedby compiler, 473
- \$newspad
 - usedby s-process, 491
- \$noEnv
 - local ref setqMultiple, 323
 - usedby compColon, 267
- \$noParseCommands
 - local ref PARSE-SpecialCommand, 405
- \$noSubsumption
 - usedby spad, 489
- \$optimizableConstructorNames
 - local ref optCallSpecially, 211
- \$options
 - usedby compileSpad2Cmd, 476
 - usedby compileSpadLispCmd, 528
 - usedby compiler, 473
 - usedby mkCategoryPackage, 139
- \$op
 - local def compDefineCapsuleFunction, 281
 - local def compDefineCategory2, 142
 - local def compDefineLisplib, 170
 - local ref compDefineCapsuleFunction, 280
 - local ref compDefineCategory2, 142
 - usedby compDefine1, 275
 - usedby compDefineFunctor1, 183
 - usedby compSubDomain1, 332
 - usedby parseDollarGreaterEqual, 104
 - usedby parseDollarGreaterThan, 104
 - usedby parseDollarLessEqual, 105
 - usedby parseDollarNotEqual, 106
 - usedby parseGreaterEqual, 107
 - usedby parseGreaterThan, 108
 - usedby parseLessEqual, 122
 - usedby parseNotEqual, 125
 - usedby parseTran, 93
 - usedby reportOnFunctorCompilation, 193
- \$out-stream
 - local ref line-print, 536
 - local ref print-package, 461
- \$packagesUsed
 - local def doIt, 254
 - local ref doIt, 253
 - usedby comp2, 499
- usedby compAdd, 248
 - usedby compDefine, 274
 - usedby compTopLevel, 494
- \$pairlis
 - local def finalizeLisplib, 175
 - usedby compDefineFunctor1, 183
- \$postStack
 - local ref displayPreCompilationErrors, 456
 - usedby postError, 356
 - usedby s-process, 491
- \$predAlist
 - usedby compDefWhereClause, 200
- \$predl
 - local ref doItIf, 257
 - local ref doIt, 253
- \$pred
 - local ref compCapsuleItems, 252
 - local ref compSingleCapsuleItem, 252
- \$prefix
 - local ref compile, 146
 - usedby compDefine1, 275
 - usedby compDefineCategory2, 142
- \$preparse-last-line, 72
 - local def preparse1, 81
 - local ref preparse1, 81
 - usedby initialize-preparse, 73
 - usedby preparseReadLine1, 87
 - usedby preparse, 77
 - defvar, 72
- \$preparseReportIfTrue
 - usedby preparse, 77
- \$previousTime
 - usedby s-process, 491
- \$profileAlist
 - usedby compDefineFunctor, 181
- \$profileCompiler
 - local ref compDefineCapsuleFunction, 280
 - local ref finalizeLisplib, 175
 - usedby compDefineFunctor, 181
 - usedby setqSingle, 326
- \$reportExitModeStack
 - usedby modifyModeStack, 525
- \$reportOptimization
 - local ref optimizeFunctionDef, 204
- \$resolveTimeSum
 - usedby compTopLevel, 494

- \$returnMode
 - local def compDefineCapsuleFunction, 281
 - local ref compDefineCapsuleFunction, 280
 - usedby compReturn, 317
 - usedby s-process, 491
- \$savableItems
 - local def compile, 146
- \$saveableItems
 - local ref compilerDoitWithScreenedLisplib, 349
 - local ref compile, 146
- \$scanIfTrue
 - usedby compOrCroak1, 497
 - usedby compileSpad2Cmd, 476
- \$semanticErrorStack
 - local ref compDefineCapsuleFunction, 280
 - usedby reportOnFunctorCompilation, 193
 - usedby s-process, 491
- \$setelt
 - usedby compDefineFunctor1, 183
- \$sideEffectsList
 - usedby compReduce1, 312
- \$sigAlist
 - usedby compDefWhereClause, 200
- \$sigList
 - local def compCategory, 262
 - local ref compCategoryItem, 263
 - local ref compCategory, 262
- \$signatureOfForm
 - local def compCapsuleItems, 252
 - local def compDefineCapsuleFunction, 281
 - local ref compDefineCapsuleFunction, 280
 - local ref compile, 146
 - local ref doIt, 254
- \$signature
 - usedby compCapsuleInner, 251
 - usedby compDefineFunctor1, 184
- \$skipme
 - local def preparse1, 81
 - usedby preparse, 77
- \$sourceFileTypes
 - usedby compileSpad2Cmd, 476
- \$spad-errors
 - usedby bumperrorcount, 457
- \$spadLibFT
 - local ref compDefineLisplib, 170
- local ref compileDocumentation, 172
- local ref finalizeLisplib, 175
- local ref lisplibDoRename, 172
- \$spad
 - usedby quote-if-string, 442
 - usedby spad, 489
- \$specialCaseKeyList
 - local def compileCases, 284
 - local ref optCallSpecially, 211
- \$splitUpItemsAlreadyThere
 - local ref compile, 146
- \$stack
 - usedby reduce-stack, 454
 - usedby stack-/empty, 89
 - usedby stack-clear, 89
 - usedby stack-load, 89
 - usedby stack-pop, 90
 - usedby stack-push, 89
- \$suffix
 - local def compCapsuleItems, 252
 - local def compile, 146
 - local ref compile, 146
- \$s
 - usedby compOrCroak1, 497
- \$template
 - usedby compDefineFunctor1, 184
- \$tokenCommands
 - local ref PARSE-SpecialCommand, 405
- \$token
 - usedby current-token, 91
 - usedby make-symbol-of, 446
 - usedby match-advance-string, 441
 - usedby next-token, 91
 - usedby prior-token, 90
 - usedby token-install, 92
 - usedby token-print, 92
 - usedby valid-tokens, 91
- \$top-level
 - local def compCapsuleItems, 252
 - local def compCategory, 262
 - local def compDefineCategory2, 142
 - usedby compDefineFunctor1, 182
 - usedby s-process, 491
- \$stopOp
 - local ref displayPreCompilationErrors, 456
 - usedby s-process, 491

- usedby setDefOp, 386
- \$tripleCache
 - usedby compDefine, 274
- \$tripleHits
 - usedby compDefine, 274
- \$true
 - local ref addArgumentConditions, 285
 - local ref optCONDtail, 206
 - local ref optIF2COND, 208
- \$tv1
 - usedby makeCategoryPredicates, 138
- \$uncondAlist
 - usedby compDefineFunctor1, 184
- \$until
 - usedby compReduce1, 312
 - usedby compRepeatOrCollect, 315
- \$viewNames
 - usedby compDefineFunctor1, 184
- \$v1, 394
 - defvar, 394
- \$warningStack
 - usedby reportOnFunctorCompilation, 193
 - usedby s-process, 491
- , 31
- abbreviation?
 - calledby parseHasRhs, 110
- abbreviationsSpad2Cmd
 - calledby mkCategoryPackage, 139
- action, 453
 - calledby PARSE-AnyId, 430
 - calledby PARSE-Category, 410
 - calledby PARSE-CommandTail, 407
 - calledby PARSE-Data, 428
 - calledby PARSE-FloatExponent, 423
 - calledby PARSE-GlyphTok, 430
 - calledby PARSE-Infix, 415
 - calledby PARSE-NBGlyphTok, 429
 - calledby PARSE-NewExpr, 403
 - calledby PARSE-OpenBrace, 432
 - calledby PARSE-OpenBracket, 432
 - calledby PARSE-Operation, 413
 - calledby PARSE-Prefix, 414
 - calledby PARSE-ReductionOp, 417
 - calledby PARSE-Sexpr1, 428
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-SpecialKeyword, 404
 - calledby PARSE-Suffix, 434
 - calledby PARSE-TokTail, 416
 - calledby PARSE-TokenCommandTail, 406
 - calledby PARSE-TokenList, 406
 - defun, 453
- add, 358
 - defplist, 358
- add-parens-and-semis-to-line, 84
 - calledby parsepiles, 84
 - calls addclose, 84
 - calls drop, 84
 - calls infixtok, 84
 - calls nonblankloc, 84
 - defun, 84
- addArgumentConditions, 285
 - calledby compDefineCapsuleFunction, 280
 - calls mkq, 285
 - calls qcar, 285
 - calls qcdr, 285
 - calls systemErrorHere, 285
 - local def \$argumentConditionList, 285
 - local ref \$argumentConditionList, 285
 - local ref \$body, 285
 - local ref \$functionName, 285
 - local ref \$true, 285
 - defun, 285
- addBinding
 - calledby addModemap1, 246
 - calledby compDefineCategory2, 142
 - calledby getSuccessEnvironment, 300
 - calledby setqMultiple, 322
- addBinding[5]
 - called by setqSingle, 326
 - called by spad, 489
- addclose, 464
 - calledby add-parens-and-semis-to-line, 84
 - calls suffix, 464
 - defun, 464
- addConstructorModemaps, 234
 - calledby augModemapsFromDomain1, 233
 - calls addModemap, 234
 - calls get1, 234
 - calls msubst, 234
 - calls putDomainsInScope, 234

- calls qcar, 234
- calls qcdr, 234
- local def \$InteractiveMode, 234
- defun, 234
- addContour
 - calledby compWhere, 336
- addDomain, 228
 - calledby comp2, 499
 - calledby comp3, 500
 - calledby compAtSign, 343
 - calledby compCapsule, 250
 - calledby compCase, 260
 - calledby compCoerce, 343
 - calledby compColonInside, 502
 - calledby compColon, 267
 - calledby compDefineCapsuleFunction, 280
 - calledby compElt, 292
 - calledby compForm1, 508
 - calledby compImport, 304
 - calledby compPretend, 310
 - calledby compSubDomain1, 332
 - calls addNewDomain, 228
 - calls constructor?, 228
 - calls domainMember, 228
 - calls getDomainsInScope, 228
 - calls getmode, 228
 - calls identp, 228
 - calls isCategoryForm, 228
 - calls isFunctor, 228
 - calls isLiteral, 228
 - calls member, 228
 - calls qslessp, 228
 - calls unknownTypeError, 228
 - defun, 228
- addEltModemap, 237
 - calledby addModemap0, 246
 - calls addModemap1, 237
 - calls makeLiteral, 237
 - calls qcar, 237
 - calls qcdr, 237
 - calls systemErrorHere, 237
 - local def \$e, 237
 - local ref \$insideCapsuleFunctionIfTrue, 237
 - defun, 237
- addEmptyCapsuleIfNecessary, 134
 - calledby compDefine1, 275
 - calls kar, 134
 - uses \$SpecialDomainNames, 134
 - defun, 134
- addInformation
 - calledby compCapsuleInner, 250
- addModemap, 245
 - calledby addConstructorModemaps, 234
 - calledby augModemapsFromCategoryRep, 243
 - calledby genDomainOps, 198
 - calledby updateCategoryFrameForCategory, 113
 - calledby updateCategoryFrameForConstructor, 112
 - calls addModemap0, 245
 - calls knownInfo, 245
 - local def \$CapsuleModemapFrame, 245
 - local ref \$CapsuleModemapFrame, 245
 - local ref \$InteractiveMode, 245
 - local ref \$e, 245
 - local ref \$insideCapsuleFunctionIfTrue, 245
 - defun, 245
- addModemap0, 246
 - calledby addModemapKnown, 245
 - calledby addModemap, 245
 - calls addEltModemap, 246
 - calls addModemap1, 246
 - calls qcar, 246
 - local ref \$functorForm, 246
 - defun, 246
- addModemap1, 246
 - calledby addEltModemap, 237
 - calledby addModemap0, 246
 - calls addBinding, 246
 - calls augProplist, 246
 - calls getProplist, 246
 - calls lassoc, 246
 - calls mkNewModemapList, 246
 - calls msubst, 246
 - calls unErrorRef, 246
 - defun, 246
- addModemapKnown, 245
 - calledby augModemapsFromCategory, 237
 - calls addModemap0, 245

- local def \$CapsuleModemapFrame, 245
 - local ref \$e, 245
 - defun, 245
- addNewDomain, 231
 - calledby addDomain, 228
 - calledby augModemapsFromDomain, 232
 - calls augModemapsFromDomain, 231
 - defun, 231
- addoptions
 - calledby initializeLisplib, 173
- addStats
 - calledby compDefineCapsuleFunction, 280
 - calledby compile, 146
 - calledby reportOnFunctorCompilation, 192
- addSuffix, 278
 - calledby mkAbbrev, 277
 - defun, 278
- Advance-Char, 539
 - calls Line-Advance-Char, 539
 - calls Line-At-End-P, 539
 - calls current-char, 539
 - calls next-line, 539
 - local ref \$in-stream, 539
 - uses \$line, 539
 - defun, 539
- advance-char
 - calledby skip-blanks, 440
- advance-token, 448
 - calledby PARSE-AnyId, 430
 - calledby PARSE-FloatExponent, 423
 - calledby PARSE-GlyphTok, 430
 - calledby PARSE-Infix, 415
 - calledby PARSE-NBGlyphTok, 429
 - calledby PARSE-OpenBrace, 432
 - calledby PARSE-OpenBracket, 432
 - calledby PARSE-Prefix, 415
 - calledby PARSE-ReductionOp, 417
 - calledby PARSE-Suffix, 434
 - calledby PARSE-TokenList, 406
 - calledby parse-argument-designator, 461
 - calledby parse-identifier, 459
 - calledby parse-keyword, 460
 - calledby parse-number, 460
 - calledby parse-spadstring, 458
 - calledby parse-string, 459
 - calls copy-token, 448
 - calls current-token, 448
 - calls try-get-token, 448
 - uses current-token, 448
 - uses valid-tokens, 448
 - defun, 448
- alistSize, 278
 - calledby mkAbbrev, 277
 - defun, 278
- allLASSOCs, 190
 - calledby augmentLisplibModemapsFrom-Functor, 189
 - defun, 190
- and, 97
- defplist, 97
- aplTran, 387
 - calledby postTransform, 351
 - calls aplTran1, 387
 - calls containsBang, 387
 - local def \$genno, 387
 - uses \$boot, 387
 - defun, 387
- aplTran1, 387
 - calledby aplTran1, 387
 - calledby aplTranList, 389
 - calledby aplTran, 387
 - calledby hasAplExtension, 389
 - calls aplTran1, 387
 - calls aplTranList, 387
 - calls hasAplExtension, 387
 - calls nreverse0, 387
 - calls , 387
 - uses \$boot, 387
 - defun, 387
- aplTranList, 389
 - calledby aplTran1, 387
 - calledby aplTranList, 389
 - calls aplTran1, 389
 - calls aplTranList, 389
 - defun, 389
- applyMapping
 - calledby comp3, 500
- argsToSig, 524
 - calledby compLambda, 307
 - defun, 524
- assignError, 328
 - calledby setqSingle, 326

- calls stackMessage, 328
- defun, 328
- assoc
 - calledby augModemapsFromCategoryRep, augmentLisplibModemapsFromFunctor, 189
 - 243
 - calledby compColon, 267
 - calledby compDefineAddSignature, 133
 - calledby compForm2, 513
 - calledby mkCategoryPackage, 139
 - calledby mkNewModemapList, 238
 - calledby mkOpVec, 199
 - calledby stripOffSubdomainConditions, 287
 - calledby transformOperationAlist, 178
- AssocBarGensym, 200
 - calledby mkOpVec, 199
 - calls EqualBarGensym, 200
 - defun, 200
- assocleft
 - calledby compDefWhereClause, 200
 - calledby compileCases, 283
 - calledby mkAlistOfExplicitCategoryOps, 156
- assocright
 - calledby compDefWhereClause, 200
 - calledby compileCases, 283
- assq
 - calledby getAbbreviation, 277
 - calledby makeFunctorArgumentParameters, 194
 - calledby mkOpVec, 199
- assq[5]
 - called by freelist, 526
- atEndOfLine
 - calledby PARSE-TokenCommandTail, 406
- augLisplibModemapsFromCategory, 155
 - calledby compDefineCategory2, 142
 - calls interactiveModemapForm, 155
 - calls isCategoryForm, 155
 - calls lassoc, 155
 - calls member, 155
 - calls mkAlistOfExplicitCategoryOps, 155
 - calls mkpf, 155
 - calls sublis, 155
 - local def \$lisplibModemapAlist, 155
 - local ref \$EmptyEnvironment, 155
 - local ref \$PatternVariableList, 155
 - local ref \$domainShell, 155
 - local ref \$lisplibModemapAlist, 155
 - defun, 155
- augModemapsFromCategory, 237
 - calledby augModemapsFromDomain1, 233
 - calledby compDefineFunctor1, 182
 - calledby genDomainView, 197
 - calls addModemapKnown, 237
 - calls compilerMessage, 237
 - calls evalAndSub, 237
 - calls putDomainsInScope, 237
 - local def \$base, 237
 - defun, 237
- augModemapsFromCategoryRep, 243
 - calledby compDefineFunctor1, 182
 - calls addModemap, 243
 - calls assoc, 243
 - calls compilerMessage, 243
 - calls evalAndSub, 243
 - calls isCategory, 243
 - calls msubst, 243
 - calls putDomainsInScope, 243
 - local def \$base, 244
 - defun, 243
- augModemapsFromDomain, 232
 - calledby addNewDomain, 231
 - calls addNewDomain, 232
 - calls augModemapsFromDomain1, 232
 - calls getDomainsInScope, 232

- calls getdatabase, 232
- calls kar, 232
- calls listOrVectorElementNode, 232
- calls member, 232
- calls opOf, 232
- calls stripUnionTags, 232
- local ref \$Category, 232
- local ref \$DummyFunctorNames, 232
- defun, 232
- augModemapsFromDomain1, 232
 - calledby augModemapsFromDomain, 232
 - calledby compForm1, 508
 - calledby setqSingle, 326
 - calls addConstructorModemaps, 233
 - calls augModemapsFromCategory, 233
 - calls getl, 232
 - calls getmodeOrMapping, 233
 - calls getmode, 233
 - calls kar, 232
 - calls stackMessage, 233
 - calls substituteCategoryArguments, 233
 - defun, 232
- augProplist
 - calledby addModemap1, 246
- autoCoerceByModemap, 345
 - calledby coerceExtraHard, 340
 - calls getModemapList, 345
 - calls get, 346
 - calls member, 345
 - calls modeEqual, 345
 - calls qcar, 345
 - calls qcdr, 345
 - calls stackMessage, 346
 - local ref \$fromCoerceable, 346
 - defun, 345
- Bang, 452
 - defmacro, 452
- bang
 - calledby PARSE-Category, 409
 - calledby PARSE-CommandTail, 407
 - calledby PARSE-Conditional, 437
 - calledby PARSE-Form, 417
 - calledby PARSE-Import, 411
 - calledby PARSE-IteratorTail, 433
 - calledby PARSE-Seg, 436
 - calledby PARSE-Sexpr1, 429
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-TokenCommandTail, 405
- bfp-
 - calledby PARSE-FloatTok, 439
- blankp, 465
 - calledby nonblankloc, 468
 - defun, 465
- Block, 362
 - defplist, 362
- boot-line-stack
 - usedby spad, 489
- bootStrapError, 192
 - calledby compCapsule, 250
 - calledby compFunctorBody, 191
 - calls mkDomainConstructor, 192
 - calls mkq, 192
 - calls namestring, 192
 - defun, 192
- bpiname
 - calledby compileTimeBindingOf, 213
 - calledby subname, 208
- bright
 - calledby checkAndDeclare, 290
 - calledby compDefineLisplib, 169
 - calledby displayMissingFunctions, 193
 - calledby doIt, 253
 - calledby hasSigInTargetCategory, 290
 - calledby optimizeFunctionDef, 204
 - calledby parseInBy, 118
 - calledby postForm, 356
 - calledby spadCompileOrSetq, 152
- browserAutoloadOnceTrigger
 - calledby compileSpad2Cmd, 476
- buildFunctor
 - calledby processFunctor, 251
- bumperrorcount, 457
 - calledby postError, 356
 - uses \$InteractiveMode, 457
 - uses \$spad-errors, 457
 - defun, 457
- call, 209
 - defplist, 209
- cannotDo
 - calledby doIt, 253

- canReturn, 298
 - calledby canReturn, 298
 - calledby compIf, 296
 - calls canReturn, 298
 - calls qcar, 298
 - calls qcdr, 298
 - calls say, 298
 - calls systemErrorHere, 298
 - defun, 298
- capsule, 250
 - defplist, 250
- CapsuleModemapFrame
 - local ref \$insideCapsuleFunctionIfTrue, 245
- case, 259
 - defplist, 259
- catch, 220
 - defplist, 220
- catches
 - compOrCroak1, 497
 - compUniquely, 516
 - preparse1, 81
 - spad, 489
- category, 98, 262, 363
 - defplist, 98, 262, 363
- char-eq, 450
 - calledby PARSE-FloatBase, 422
 - calledby PARSE-TokTail, 416
 - defun, 450
- char-ne, 450
 - calledby PARSE-FloatBase, 422
 - calledby PARSE-Selector, 419
 - defun, 450
- chaseInferences
 - calledby compHas, 294
- checkAndDeclare, 289
 - calledby compDefineCapsuleFunction, 280
 - calls bright, 290
 - calls getArgumentMode, 289
 - calls modeEqual, 289
 - calls put, 289
 - calls sayBrightly, 289
 - defun, 289
- checkWarning, 461
 - calledby postCapsule, 359
 - calls concat, 461
 - calls postError, 461
 - defun, 461
- clearClams
 - calledby compileConstructor, 153
- clearConstructorCache
 - calledby compileConstructor1, 153
- coerce, 337
 - calledby coerceExit, 342
 - calledby coerceExtraHard, 340
 - calledby coerceable, 342
 - calledby compAtSign, 343
 - calledby compCase, 260
 - calledby compCoerce1, 344
 - calledby compCoerce, 343
 - calledby compColonInside, 502
 - calledby compForm1, 508
 - calledby compHas, 294
 - calledby compIf, 296
 - calledby compIs, 304
 - calledby convert, 504
 - calls coerceEasy, 337
 - calls coerceHard, 337
 - calls coerceSubset, 337
 - calls isSomeDomainVariable, 337
 - calls keyedSystemError, 337
 - calls msubst, 337
 - calls rplac, 337
 - calls stackMessage, 337
 - local ref \$InteractiveMode, 337
 - local ref \$Rep, 337
 - local ref \$fromCoerceable, 337
 - defun, 337
- coerceable, 341
 - calledby compForm1, 508
 - calls coerce, 342
 - calls pmatch, 341
 - calls sublis, 342
 - local ref \$fromCoerceable, 342
 - defun, 341
- coerceByModemap, 345
 - calledby compCoerce1, 344
 - calls genDeltaEntry, 345
 - calls isSubset, 345
 - calls modeEqual, 345
 - calls qcar, 345
 - calls qcdr, 345

- defun, 345
- coerceEasy, 338
 - calledby coerce, 337
 - calls modeEqualSubst, 338
 - local ref \$EmptyMode, 338
 - local ref \$Exit, 338
 - local ref \$NoValueMode, 338
 - local ref \$Void, 338
 - defun, 338
- coerceExit, 342
 - calledby compRepeatOrCollect, 315
 - calls coerce, 342
 - calls replaceExitEsc, 342
 - calls resolve, 342
 - local ref \$exitMode, 342
 - defun, 342
- coerceExtraHard, 340
 - calledby coerceHard, 339
 - calls autoCoerceByModemap, 340
 - calls coerce, 340
 - calls hasType, 340
 - calls isUnionMode, 340
 - calls member, 340
 - calls qcar, 340
 - calls qcdr, 340
 - local ref \$Expression, 340
 - defun, 340
- coerceHard, 339
 - calledby coerce, 337
 - calls coerceExtraHard, 339
 - calls extendsCategoryForm, 339
 - calls getmode, 339
 - calls get, 339
 - calls isCategoryForm, 339
 - calls modeEqual, 339
 - local def \$e, 339
 - local ref \$String, 339
 - local ref \$bootStrapMode, 339
 - local ref \$e, 339
 - defun, 339
- coerceSubset, 338
 - calledby coerce, 337
 - calls eval, 338
 - calls get, 338
 - calls isSubset, 338
 - calls lassoc, 338
 - calls maxSuperType, 338
 - calls msubst, 338
 - calls opOf, 338
 - defun, 338
- collect, 314, 365
 - calledby floatexpid, 451
 - defplist, 314, 365
- comma2Tuple, 368
 - calledby postComma, 368
 - calledby postConstruct, 369
 - calls postFlatten, 368
 - defun, 368
- comp, 497
 - calledby compAdd, 248
 - calledby compArgumentsAndTryAgain, 517
 - calledby compAtSign, 343
 - calledby compBoolean, 300
 - calledby compCase1, 260
 - calledby compCoerce1, 344
 - calledby compColonInside, 502
 - calledby compCons1, 271
 - calledby compDefWhereClause, 200
 - calledby compDefineAddSignature, 133
 - calledby compExit, 293
 - calledby compExpressionList, 512
 - calledby compForMode, 307
 - calledby compForm1, 508
 - calledby compFromIf, 297
 - calledby compHasFormat, 295
 - calledby compIs, 304
 - calledby compLeave, 308
 - calledby compList, 506
 - calledby compOrCroak1, 496
 - calledby compPretend, 310
 - calledby compReduce1, 312
 - calledby compRepeatOrCollect, 315
 - calledby compReturn, 317
 - calledby compSeqItem, 320
 - calledby compSubsetCategory, 333
 - calledby compSuchthat, 334
 - calledby compUniquely, 516
 - calledby compVector, 335
 - calledby compWhere, 336
 - calledby compWithMappingMode1, 518
 - calledby compileConstructor1, 153
 - calledby doItIf, 257

- calledby getSuccessEnvironment, 300
- calledby outputComp, 328
- calledby setqSetelt, 326
- calledby setqSingle, 326
- calledby spadCompileOrSetq, 152
- calls compNoStacking, 497
- local ref \$compStack, 498
- uses \$exitModeStack, 498
- defun, 497
- comp-tran
 - calledby compWithMappingMode1, 518
- comp2, 499
 - calledby compNoStacking1, 498
 - calledby compNoStacking, 498
 - calls addDomain, 499
 - calls comp3, 499
 - calls insert, 499
 - calls isDomainForm, 499
 - calls isFunctor, 499
 - calls nequal, 499
 - calls opOf, 499
 - uses \$bootStrapMode, 499
 - uses \$lisplib, 499
 - uses \$packagesUsed, 499
 - defun, 499
- comp3, 500
 - calledby comp2, 499
 - calledby compTypeOf, 502
 - calls addDomain, 500
 - calls applyMapping, 500
 - calls compApply, 500
 - calls compAtom, 500
 - calls compCoerce, 500
 - calls compColon, 500
 - calls compExpression, 500
 - calls compTypeOf, 500
 - calls compWithMappingMode, 500
 - calls getDomainsInScope, 500
 - calls getmode, 500
 - calls member[5], 500
 - calls pname[5], 500
 - calls stringPrefix?, 500
 - uses \$e, 500
 - uses \$insideCompTypeOf, 500
 - defun, 500
- compAdd, 247
 - calls NRTgetLocalIndex, 248
 - calls compCapsule, 248
 - calls compOrCroak, 248
 - calls compSubDomain1, 248
 - calls compTuple2Record, 248
 - calls comp, 248
 - calls nreverse0, 248
 - calls qcar, 248
 - calls qcdr, 248
 - uses /editfile, 248
 - uses \$EmptyMode, 248
 - uses \$NRTaddForm, 248
 - uses \$addFormLhs, 248
 - uses \$addForm, 248
 - uses \$bootStrapMode, 248
 - uses \$functorForm, 248
 - uses \$packagesUsed, 248
 - defun, 247
- compAndDefine
 - calledby compileConstructor1, 153
- compApply
 - calledby comp3, 500
- compApplyModemap
 - calledby getModemap, 234
- compareMode2Arg
 - calledby hasSigInTargetCategory, 290
- compArgumentConditions, 286
 - calledby compDefineCapsuleFunction, 280
 - calls compOrCroak, 286
 - calls msubst, 286
 - local def \$argumentConditionList, 286
 - local ref \$Boolean, 286
 - local ref \$argumentConditionList, 286
 - defun, 286
- compArgumentsAndTryAgain, 517
 - calledby compForm, 507
 - calls compForm1, 517
 - calls comp, 517
 - uses \$EmptyMode, 517
 - defun, 517
- compAtom, 503
 - calledby comp3, 500
 - calls compAtomWithModemap, 503
 - calls compList, 503
 - calls compSymbol, 503
 - calls compVector, 503

- calls convert, 503
- calls get, 503
- calls isSymbol, 503
- calls modeIsAggregateOf, 503
- calls primitiveType, 503
- uses \$Expression, 503
- defun, 503
- compAtomWithModemap
 - calledby compAtom, 503
- compAtSign, 343
 - calledby compLambda, 307
 - calls addDomain, 343
 - calls coerce, 343
 - calls comp, 343
 - defun, 343
- compBoolean, 300
 - calledby compIf, 296
 - calls comp, 300
 - calls getInverseEnvironment, 300
 - calls getSuccessEnvironment, 300
 - defun, 300
- compCapsule, 250
 - calledby compAdd, 248
 - calledby compSubDomain, 331
 - calls addDomain, 250
 - calls bootStrapError, 250
 - calls compCapsuleInner, 250
 - uses \$bootStrapMode, 250
 - uses \$functorForm, 250
 - uses \$insideExpressionIfTrue, 250
 - uses editfile, 250
 - defun, 250
- compCapsuleInner, 250
 - calledby compCapsule, 250
 - calls addInformation, 250
 - calls compCapsuleItems, 251
 - calls mkpf, 251
 - calls processFunctor, 251
 - uses \$addForm, 251
 - uses \$form, 251
 - uses \$functorLocalParameters, 251
 - uses \$getDomainCode, 251
 - uses \$insideCategoryIfTrue, 251
 - uses \$insideCategoryPackageIfTrue, 251
 - uses \$signature, 251
 - defun, 250
- compCapsuleItems, 252
 - calledby compCapsuleInner, 251
 - calls compSingleCapsuleItem, 252
 - local def \$e, 252
 - local def \$myFunctorBody, 252
 - local def \$signatureOfForm, 252
 - local def \$suffix, 252
 - local def \$stop-level, 252
 - local ref \$e, 252
 - local ref \$pred, 252
 - defun, 252
- compCase, 259
 - calls addDomain, 260
 - calls coerce, 260
 - calls compCase1, 260
 - defun, 259
- compCase1, 260
 - calledby compCase, 260
 - calls comp, 260
 - calls getModemapList, 260
 - calls modeEqual, 260
 - calls nreverse0, 260
 - uses \$Boolean, 260
 - uses \$EmptyMode, 260
 - defun, 260
- compCat, 261
 - calls getl, 261
 - defun, 261
- compCategory, 262
 - calls compCategoryItem, 262
 - calls mkExplicitCategoryFunction, 262
 - calls qcar, 262
 - calls qcdr, 262
 - calls resolve, 262
 - calls systemErrorHere, 262
 - local def \$atList, 262
 - local def \$sigList, 262
 - local def \$stop-level, 262
 - local ref \$atList, 262
 - local ref \$sigList, 262
 - defun, 262
- compCategoryItem, 263
 - calledby compCategoryItem, 263
 - calledby compCategory, 262
 - calls compCategoryItem, 263
 - calls mkpf, 263

- calls qcar, 263
- calls qcdr, 263
- local ref \$atList, 263
- local ref \$sigList, 263
- defun, 263
- compCoerce, 343
 - calledby comp3, 500
 - calls addDomain, 343
 - calls coerce, 343
 - calls compCoerce1, 343
 - calls getmode, 343
 - defun, 343
- compCoerce1, 344
 - calledby compCoerce, 343
 - calls coerceByModemap, 344
 - calls coerce, 344
 - calls comp, 344
 - calls mkq, 344
 - calls msubst, 344
 - calls resolve, 344
 - defun, 344
- compColon, 267
 - calledby comp3, 500
 - calledby compColon, 267
 - calledby compMakeDeclaration, 525
 - calls addDomain, 267
 - calls assoc, 267
 - calls compColonInside, 267
 - calls compColon, 267
 - calls eqsubstlist, 267
 - calls genSomeVariable, 267
 - calls getDomainsInScope, 267
 - calls getmode, 267
 - calls isCategoryForm, 267
 - calls isDomainForm, 267
 - calls length, 267
 - calls makeCategoryForm, 267
 - calls member[5], 267
 - calls nreverse0, 267
 - calls put, 267
 - calls systemErrorHere, 267
 - calls take, 267
 - calls unknownTypeError, 267
 - uses \$FormalMapVariableList, 267
 - uses \$bootStrapMode, 267
 - uses \$insideCategoryIfTrue, 267
 - uses \$insideExpressionIfTrue, 267
 - uses \$insideFunctorIfTrue, 267
 - uses \$lhsOfColon, 267
 - uses \$noEnv, 267
 - defun, 267
- compColonInside, 502
 - calledby compColon, 267
 - calls addDomain, 502
 - calls coerce, 502
 - calls comp, 502
 - calls opOf, 502
 - calls stackSemanticError, 502
 - calls stackWarning, 502
 - uses \$EmptyMode, 502
 - uses \$newCompilerUnionFlag, 502
 - defun, 502
- compCons, 270
 - calls compCons1, 270
 - calls compForm, 270
 - defun, 270
- compCons1, 271
 - calledby compCons, 270
 - calls comp, 271
 - calls convert, 271
 - calls qcar, 271
 - calls qcdr, 271
 - uses \$EmptyMode, 271
 - defun, 271
- compConstruct, 272
 - calls compForm, 272
 - calls compList, 272
 - calls compVector, 272
 - calls convert, 272
 - calls getDomainsInScope, 272
 - calls modeIsAggregateOf, 272
 - defun, 272
- compConstructorCategory, 274
 - calls resolve, 274
 - uses \$Category, 274
 - defun, 274
- compDefine, 274
 - calls compDefine1, 274
 - uses \$macroIfTrue, 274
 - uses \$packagesUsed, 274
 - uses \$tripleCache, 274
 - uses \$tripleHits, 274

- defun, 274
- compDefine1, 275
 - calledby compDefine1, 275
 - calledby compDefineCategory1, 137
 - calledby compDefine, 274
 - calls addEmptyCapsuleIfNecessary, 275
 - calls compDefWhereClause, 275
 - calls compDefine1, 275
 - calls compDefineAddSignature, 275
 - calls compDefineCapsuleFunction, 275
 - calls compDefineCategory, 275
 - calls compDefineFunctor, 275
 - calls compInternalFunction, 275
 - calls getAbbreviation, 275
 - calls getSignatureFromMode, 275
 - calls getTargetFromRhs, 275
 - calls giveFormalParametersValues, 275
 - calls isDomainForm, 275
 - calls isMacro, 275
 - calls length, 275
 - calls macroExpand, 275
 - calls stackAndThrow, 275
 - calls strconc, 275
 - uses \$Category, 275
 - uses \$ConstructorNames, 275
 - uses \$EmptyMode, 275
 - uses \$NoValueMode, 275
 - uses \$formalArgList, 275
 - uses \$form, 275
 - uses \$insideCapsuleFunctionIfTrue, 275
 - uses \$insideCategoryIfTrue, 275
 - uses \$insideExpressionIfTrue, 275
 - uses \$insideFunctorIfTrue, 275
 - uses \$insideWhereIfTrue, 275
 - uses \$op, 275
 - uses \$prefix, 275
 - defun, 275
- compDefineAddSignature, 133
 - calledby compDefine1, 275
 - calls assoc, 133
 - calls comp, 133
 - calls getProplist, 133
 - calls hasFullSignature, 133
 - calls lassoc, 133
 - uses \$EmptyMode, 133
 - defun, 133
- compDefineCapsuleFunction, 280
 - calledby compDefine1, 275
 - calls NRTassignCapsuleFunctionSlot, 280
 - calls addArgumentConditions, 280
 - calls addDomain, 280
 - calls addStats, 280
 - calls checkAndDeclare, 280
 - calls compArgumentConditions, 280
 - calls compOrCroak, 280
 - calls compileCases, 280
 - calls formatUnabbreviated, 280
 - calls getArgumentModeOrMoan, 280
 - calls getSignature, 280
 - calls getmode, 280
 - calls get, 280
 - calls giveFormalParametersValues, 280
 - calls hasSigInTargetCategory, 280
 - calls length, 280
 - calls member, 280
 - calls mkq, 280
 - calls profileRecord, 280
 - calls put, 280
 - calls replaceExitEtc, 280
 - calls resolve, 280
 - calls sayBrightly, 280
 - calls stripOffArgumentConditions, 280
 - calls stripOffSubdomainConditions, 280
 - local def \$CapsuleDomainsInScope, 281
 - local def \$CapsuleModemapFrame, 280
 - local def \$argumentConditionList, 280
 - local def \$finalEnv, 280
 - local def \$formalArgList, 281
 - local def \$form, 280
 - local def \$functionLocations, 281
 - local def \$functionStats, 280
 - local def \$initCapsuleErrorCount, 280
 - local def \$insideCapsuleFunctionIfTrue, 280
 - local def \$insideExpressionIfTrue, 281
 - local def \$op, 281
 - local def \$returnMode, 281
 - local def \$signatureOffForm, 281
 - local ref \$DomainsInScope, 280
 - local ref \$compileOnlyCertainItems, 280
 - local ref \$formalArgList, 280
 - local ref \$functionLocations, 280

- local ref \$functionStats, 280
- local ref \$functorStats, 280
- local ref \$op, 280
- local ref \$profileCompiler, 280
- local ref \$returnMode, 280
- local ref \$semanticErrorStack, 280
- local ref \$signatureOfForm, 280
- defun, 280
- compDefineCategory, 169
 - calledby compDefine1, 275
 - calls compDefineCategory1, 169
 - calls compDefineLisplib, 169
 - uses \$domainShell, 169
 - uses \$insideFunctorIfTrue, 169
 - uses \$lisplibCategory, 169
 - uses \$lisplib, 169
 - defun, 169
- compDefineCategory1, 137
 - calledby compDefineCategory, 169
 - calls compDefine1, 137
 - calls compDefineCategory2, 137
 - calls makeCategoryPredicates, 137
 - calls mkCategoryPackage, 137
 - uses \$EmptyMode, 137
 - uses \$bootStrapMode, 137
 - uses \$categoryPredicateList, 137
 - uses \$insideCategoryPackageIfTrue, 137
 - uses \$lisplibCategory, 137
 - defun, 137
- compDefineCategory2, 142
 - calledby compDefineCategory1, 137
 - calls addBinding, 142
 - calls augLisplibModemapsFromCategory, 142
 - calls compMakeDeclaration, 142
 - calls compOrCroak, 142
 - calls compile, 142
 - calls computeAncestorsOf, 142
 - calls constructor?, 142
 - calls evalAndRwriteLispForm, 142
 - calls eval, 142
 - calls getArgumentModeOrMoan, 142
 - calls getParentsFor, 142
 - calls giveFormalParametersValues, 142
 - calls lisplibWrite, 142
 - calls mkConstructor, 142
 - calls mkq, 142
 - calls nequal, 142
 - calls opOf, 142
 - calls optFunctorBody, 142
 - calls removeZeroOne, 142
 - calls sublis, 142
 - calls take, 142
 - local def \$addForm, 143
 - local def \$definition, 142
 - local def \$domainShell, 143
 - local def \$extraParms, 142
 - local def \$formalArgList, 142
 - local def \$form, 142
 - local def \$frontier, 142
 - local def \$functionStats, 142
 - local def \$functorForm, 143
 - local def \$functorStats, 142
 - local def \$getDomainCode, 142
 - local def \$insideCategoryIfTrue, 142
 - local def \$lisplibAbbreviation, 143
 - local def \$lisplibAncestors, 143
 - local def \$lisplibCategory, 143
 - local def \$lisplibForm, 143
 - local def \$lisplibKind, 143
 - local def \$lisplibModemap, 143
 - local def \$lisplibParents, 143
 - local def \$op, 142
 - local def \$stop-level, 142
 - local ref \$FormalMapVariableList, 142
 - local ref \$TriangleVariableList, 142
 - local ref \$definition, 142
 - local ref \$extraParms, 142
 - local ref \$formalArgList, 142
 - local ref \$form, 142
 - local ref \$libFile, 142
 - local ref \$lisplibCategory, 142
 - local ref \$lisplib, 142
 - local ref \$op, 142
 - uses \$prefix, 142
 - defun, 142
- compDefineFunctor, 181
 - calledby compDefine1, 275
 - calls compDefineFunctor1, 181
 - calls compDefineLisplib, 181
 - uses \$domainShell, 181
 - uses \$lisplib, 181

- uses \$profileAlist, 181
- uses \$profileCompiler, 181
- defun, 181
- compDefineFunctor1, 181
 - calledby compDefineFunctor, 181
 - calls NRTgenInitialAttributeAlist, 182
 - calls NRTgetLocalIndex, 182
 - calls NRTgetLookupFunction, 182
 - calls NRTmakeSlot1Info, 182
 - calls augModemapsFromCategoryRep, 182
 - calls augModemapsFromCategory, 182
 - calls augmentLisplibModemapsFromFunctor, 182
 - calls compFunctorBody, 182
 - calls compMakeCategoryObject, 181
 - calls compMakeDeclaration, 182
 - calls compile, 182
 - calls computeAncestorsOf, 182
 - calls constructor?, 182
 - calls disallowNilAttribute, 182
 - calls evalAndRwriteLispForm, 182
 - calls getArgumentModeOrMoan, 181
 - calls getModemap, 181
 - calls getParentsFor, 182
 - calls getdatabase, 182
 - calls giveFormalParametersValues, 181
 - calls isCategoryPackageName, 181, 182
 - calls lisplibWrite, 182
 - calls makeFunctorArgumentParameters, 182
 - calls maxindex, 182
 - calls mkq, 182
 - calls nequal, 182
 - calls pname, 181
 - calls pp, 181
 - calls qcar, 182
 - calls qcdr, 182
 - calls remdup, 182
 - calls removeZeroOne, 182
 - calls reportOnFunctorCompilation, 182
 - calls sayBrightly, 181
 - calls simpBool, 182
 - calls strconc, 181
 - calls sublis, 182
 - uses \$CategoryFrame, 182
 - uses \$CheckVectorList, 182
 - uses \$FormalMapVariableList, 182
 - uses \$LocalDomainAlist, 182
 - uses \$NRTaddForm, 182
 - uses \$NRTaddList, 182
 - uses \$NRTattributeAlist, 182
 - uses \$NRTbase, 182
 - uses \$NRTdeltaLength, 182
 - uses \$NRTdeltaListComp, 182
 - uses \$NRTdeltaList, 182
 - uses \$NRTdomainFormList, 182
 - uses \$NRTloadTimeAlist, 182
 - uses \$NRTslot1Info, 183
 - uses \$NRTslot1PredicateList, 183
 - uses \$QuickCode, 183
 - uses \$Representation, 183
 - uses \$addForm, 183
 - uses \$attributesName, 183
 - uses \$bootStrapMode, 182
 - uses \$byteAddress, 183
 - uses \$byteVec, 183
 - uses \$compileOnlyCertainItems, 183
 - uses \$condAlist, 183
 - uses \$domainShell, 183
 - uses \$form, 183
 - uses \$functionLocations, 183
 - uses \$functionStats, 183
 - uses \$functorForm, 183
 - uses \$functorLocalParameters, 183
 - uses \$functorSpecialCases, 183
 - uses \$functorStats, 183
 - uses \$functorTarget, 183
 - uses \$functorsUsed, 183
 - uses \$genFVar, 183
 - uses \$genSDVar, 183
 - uses \$getDomainCode, 183
 - uses \$goGetList, 183
 - uses \$insideCategoryPackageIfTrue, 183
 - uses \$insideFunctorIfTrue, 183
 - uses \$isOpPackageName, 183
 - uses \$libFile, 183
 - uses \$lisplibAbbreviation, 183
 - uses \$lisplibAncestors, 183
 - uses \$lisplibCategoriesExtended, 183
 - uses \$lisplibCategory, 183
 - uses \$lisplibForm, 183
 - uses \$lisplibFunctionLocations, 184

- uses \$lisplibKind, 183
- uses \$lisplibMissingFunctions, 183
- uses \$lisplibModemap, 183
- uses \$lisplibOperationAlist, 183
- uses \$lisplibParents, 183
- uses \$lisplibSlot1, 183
- uses \$lisplib, 182
- uses \$lookupFunction, 183
- uses \$mutableDomains, 183
- uses \$mutableDomain, 183
- uses \$myFunctorBody, 183
- uses \$op, 183
- uses \$pairlis, 183
- uses \$setelt, 183
- uses \$signature, 184
- uses \$template, 184
- uses \$top-level, 182
- uses \$uncondAlist, 184
- uses \$viewNames, 184
- defun, 181
- compDefineLisplib, 169
 - calledby compDefineCategory, 169
 - calledby compDefineFunctor, 181
 - calls bright, 169
 - calls compileDocumentation, 169
 - calls filep, 170
 - calls fillerSpaces, 169
 - calls finalizeLisplib, 169
 - calls getConstructorAbbreviation, 169
 - calls getdatabase, 170
 - calls lisplibDoRename, 169
 - calls localdatabase, 170
 - calls rpackfile, 170
 - calls rshut, 169
 - calls sayMSG, 169
 - calls unloadOneConstructor, 170
 - calls updateCategoryFrameForCategory, 170
 - calls updateCategoryFrameForConstructor, 170
 - local def \$libFile, 170
 - local def \$lisplibAbbreviation, 170
 - local def \$lisplibAncestors, 170
 - local def \$lisplibCategoriesExtended, 170
 - local def \$lisplibCategory, 170
 - local def \$lisplibForm, 170
 - local def \$lisplibKind, 170
 - local def \$lisplibModemapAlist, 170
 - local def \$lisplibModemap, 170
 - local def \$lisplibOperationAlist, 170
 - local def \$lisplibParents, 170
 - local def \$lisplibPredicates, 170
 - local def \$lisplibSlot1, 170
 - local def \$lisplibSuperDomain, 170
 - local def \$lisplibVariableAlist, 170
 - local def \$lisplib, 170
 - local def \$newConlist, 170
 - local def \$op, 170
 - local ref \$algebraOutputStream, 170
 - local ref \$compileDocumentation, 170
 - local ref \$filep, 170
 - local ref \$lisplibKind, 170
 - local ref \$newConlist, 170
 - local ref \$spadLibFT, 170
 - defun, 169
- compDefWhereClause, 200
 - calledby compDefine1, 275
 - calls assocleft, 200
 - calls assocright, 200
 - calls comp, 200
 - calls concat, 200
 - calls delete, 200
 - calls getmode, 200
 - calls lassoc, 200
 - calls listOffidentifiersIn, 200
 - calls orderByDependency, 200
 - calls pairList, 200
 - calls qcar, 200
 - calls qcdr, 200
 - calls union, 200
 - calls userError, 200
 - uses \$predAlist, 200
 - uses \$sigAlist, 200
 - defun, 200
- compElt, 292
 - calls addDomain, 292
 - calls compForm, 292
 - calls convert, 292
 - calls getDeltaEntry, 292
 - calls getModemapListFromDomain, 292
 - calls isDomainForm, 292
 - calls length, 292

- calls nequal, 292
- calls opOf, 292
- calls stackMessage, 292
- calls stackWarning, 292
- uses \$One, 292
- uses \$Zero, 292
- defun, 292
- compExit, 293
 - calls comp, 293
 - calls modifyModeStack, 293
 - calls stackMessageIfNone, 293
 - uses \$exitModeStack, 293
 - defun, 293
- compExpression, 507
 - calledby comp3, 500
 - calls compForm, 507
 - calls getl, 507
 - uses \$insideExpressionIfTrue, 507
 - defun, 507
- compExpressionList, 512
 - calledby compForm1, 508
 - calls comp, 512
 - calls convert, 512
 - calls nreverse0, 512
 - local ref \$Expression, 512
 - defun, 512
- compForm, 507
 - calledby compConstruct, 272
 - calledby compCons, 270
 - calledby compElt, 292
 - calledby compExpression, 507
 - calls compArgumentsAndTryAgain, 507
 - calls compForm1, 507
 - calls stackMessageIfNone, 507
 - defun, 507
- compForm1, 508
 - calledby compArgumentsAndTryAgain, 517
 - calledby compForm, 507
 - calls addDomain, 508
 - calls augModemapsFromDomain1, 508
 - calls coerceable, 508
 - calls coerce, 508
 - calls compExpressionList, 508
 - calls compForm2, 508
 - calls compOrCroak, 508
 - calls compToApply, 508
- calls comp, 508
- calls getFormModemaps, 508
- calls length, 508
- calls nreverse0, 508
- calls outputComp, 508
- uses \$EmptyMode, 508
- uses \$Expression, 508
- uses \$NumberOfArgsIfInteger, 508
- defun, 508
- compForm2, 513
 - calledby compForm1, 508
 - calls PredImplies, 513
 - calls assoc, 513
 - calls compForm3, 513
 - calls compFormPartiallyBottomUp, 513
 - calls compUniquely, 513
 - calls isSimple, 513
 - calls length, 513
 - calls nreverse0, 513
 - calls sublis, 513
 - calls take, 513
 - uses \$EmptyMode, 513
 - uses \$TriangleVariableList, 513
 - defun, 513
- compForm3, 515
 - calledby compForm2, 513
 - calledby compFormPartiallyBottomUp, 516
 - calls compFormWithModemap, 515
 - local ref \$compUniquelyIfTrue, 515
 - defun, 515
 - throws, 515
- compFormMatch, 516
 - calledby compFormPartiallyBottomUp, 516
 - defun, 516
- compForMode, 307
 - calledby compJoin, 305
 - calls comp, 307
 - local def \$compForModeIfTrue, 307
 - defun, 307
- compFormPartiallyBottomUp, 516
 - calledby compForm2, 513
 - calls compForm3, 516
 - calls compFormMatch, 516
 - defun, 516
- compFormWithModemap
 - calledby compForm3, 515

- compFromIf, 297
 - calledby compIf, 296
 - calls comp, 297
 - defun, 297
- compFunctorBody, 191
 - calledby compDefineFunctor1, 182
 - calls bootStrapError, 191
 - calls compOrCroak, 191
 - uses /editfile, 191
 - uses \$NRTaddForm, 191
 - uses \$bootStrapMode, 191
 - uses \$functorForm, 191
 - defun, 191
- compHas, 294
 - calls chaseInferences, 294
 - calls coerce, 294
 - calls compHasFormat, 294
 - local def \$e, 294
 - local ref \$Boolean, 294
 - local ref \$e, 294
 - defun, 294
- compHasFormat, 295
 - calledby compHas, 294
 - calls comp, 295
 - calls isDomainForm, 295
 - calls length, 295
 - calls mkDomainConstructor, 295
 - calls mkList, 295
 - calls qcar, 295
 - calls qcdr, 295
 - calls sublislis, 295
 - calls take, 295
 - local ref \$EmptyEnvironment, 295
 - local ref \$EmptyMode, 295
 - local ref \$FormalMapVariableList, 295
 - local ref \$e, 295
 - local ref \$form, 295
 - defun, 295
- compIf, 296
 - calls canReturn, 296
 - calls coerce, 296
 - calls compBoolean, 296
 - calls compFromIf, 296
 - calls intersectionEnvironment, 296
 - calls quotify, 296
 - calls resolve, 296
 - uses \$Boolean, 297
 - defun, 296
- compile, 145
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby compileCases, 283
 - calls addStats, 146
 - calls constructMacro, 145
 - calls elapsedTime, 146
 - calls encodeFunctionName, 145
 - calls encodeItem, 145
 - calls getmode, 145
 - calls get, 145
 - calls kar, 145
 - calls member, 145
 - calls modeEqual, 145
 - calls nequal, 145
 - calls optimizeFunctionDef, 145
 - calls printStats, 146
 - calls putInLocalDomainReferences, 145
 - calls qcar, 145
 - calls qcdr, 145
 - calls sayBrightly, 145
 - calls spadCompileOrSetq, 146
 - calls splitEncodedFunctionName, 145
 - calls strconc, 145
 - calls userError, 145
 - local def \$functionStats, 146
 - local def \$savableItems, 146
 - local def \$suffix, 146
 - local ref \$compileOnlyCertainItems, 146
 - local ref \$doNotCompileJustPrint, 146
 - local ref \$e, 146
 - local ref \$functionStats, 146
 - local ref \$functorForm, 146
 - local ref \$insideCapsuleFunctionIfTrue, 146
 - local ref \$lisplibItemsAlreadyThere, 146
 - local ref \$lisplib, 146
 - local ref \$macroIfTrue, 146
 - local ref \$prefix, 146
 - local ref \$saveableItems, 146
 - local ref \$signatureOfForm, 146
 - local ref \$splitUpItemsAlreadyThere, 146
 - local ref \$suffix, 146
 - defun, 145

- compile-lib-file, 530
 - calledby recompile-lib-file-if-necessary, 529
 - defun, 530
- compileCases, 283
 - calledby compDefineCapsuleFunction, 280
 - calls assocleft, 283
 - calls assocright, 283
 - calls compile, 283
 - calls eval, 283
 - calls getSpecialCaseAssoc, 283
 - calls get, 283
 - calls mkpf, 283
 - calls msubst, 283
 - calls outerProduct, 283
 - calls qcar, 283
 - calls qcdr, 283
 - local def \$specialCaseKeyList, 284
 - local ref \$getDomainCode, 283
 - local ref \$insideFunctorIfTrue, 283
 - defun, 283
- compileConstructor, 153
 - calledby spadCompileOrSetq, 152
 - calls clearClams, 153
 - calls compileConstructor1, 153
 - defun, 153
- compileConstructor1, 153
 - calledby compileConstructor, 153
 - calls clearConstructorCache, 153
 - calls compAndDefine, 153
 - calls comp, 153
 - calls getdatabase, 153
 - local def \$clamList, 153
 - local ref \$ConstructorCache, 153
 - local ref \$clamList, 153
 - local ref \$mutableDomain, 153
 - defun, 153
- compiled-function-p
 - calledby subrname, 208
- compileDocumentation, 172
 - calledby compDefineLisplib, 169
 - calls finalizeDocumentation, 172
 - calls lisplibWrite, 172
 - calls make-input-filename, 172
 - calls rdefiostream, 172
 - calls replaceFile, 172
 - calls rpackfile, 172
 - calls rshut, 172
 - local ref \$EmptyMode, 172
 - local ref \$e, 172
 - local ref \$fcopy, 172
 - local ref \$spadLibFT, 172
 - defun, 172
- compileFileQuietly, 530
 - uses *standard-output*, 530
 - uses \$InteractiveMode, 530
 - defun, 530
- compiler, 472
 - calls compileSpad2Cmd, 473
 - calls compileSpadLispCmd, 473
 - calls findfile, 473
 - calls helpSpad2Cmd[5], 473
 - calls mergePathnames[5], 473
 - calls namestring[5], 473
 - calls pathnameType[5], 473
 - calls pathname[5], 473
 - calls selectOptionLC[5], 473
 - calls throwKeyedMsg, 473
 - uses /editfile, 473
 - uses \$newConlist, 473
 - uses \$options, 473
 - defun, 472
- compilerDoit, 478
 - calledby compileSpad2Cmd, 476
 - calledby compilerDoitWithScreenedLisplib, 349
 - calls /RQ,LIB, 478
 - calls /rf[5], 478
 - calls /rq[5], 478
 - calls member[5], 478
 - calls opOf, 478
 - calls sayBrightly, 478
 - uses \$byConstructors, 478
 - uses \$constructorsSeen, 478
 - defun, 478
- compilerDoitWithScreenedLisplib
 - calledby compileSpad2Cmd, 476
 - calls compilerDoit, 349
 - calls embed, 349
 - calls rwrite, 349
 - calls unembed, 349
 - local ref \$libFile, 349
 - local ref \$saveableItems, 349

- compilerMessage
 - calledby augModemapsFromCategoryRep, 243
 - calledby augModemapsFromCategory, 237
- compileSpad2Cmd, 474
 - calledby compiler, 473
 - calls browserAutoloadOnceTrigger, 476
 - calls compilerDoitWithScreenedLisplib, 476
 - calls compilerDoit, 476
 - calls error, 476
 - calls extendLocalLibdb, 476
 - calls namestring[5], 476
 - calls nequal, 476
 - calls object2String, 476
 - calls pathnameType[5], 476
 - calls pathname[5], 476
 - calls sayKeyedMsg[5], 476
 - calls selectOptionLC[5], 476
 - calls spad2AsTranslatorAutoloadOnceTrigger, 476
 - calls spadPrompt, 476
 - calls strconc, 476
 - calls terminateSystemCommand[5], 476
 - calls throwKeyedMsg, 476
 - calls updateSourceFiles[5], 476
 - uses /editfile, 476
 - uses \$InteractiveMode, 476
 - uses \$QuickCode, 476
 - uses \$QuickLet, 476
 - uses \$compileOnlyCertainItems, 476
 - uses \$f, 476
 - uses \$m, 476
 - uses \$newComp, 476
 - uses \$newConlist, 476
 - uses \$options, 476
 - uses \$scanIfTrue, 476
 - uses \$sourceFileTypes, 476
 - defun, 474
- compileSpadLispCmd, 528
 - calledby compiler, 473
 - calls fnameMake[5], 528
 - calls fnameReadable?[5], 528
 - calls localdatabase[5], 528
 - calls namestring[5], 528
 - calls object2String, 528
 - calls pathnameDirectory[5], 528
 - calls pathnameName[5], 528
 - calls pathnameType[5], 528
 - calls pathname[5], 528
 - calls recompile-lib-file-if-necessary, 528
 - calls sayKeyedMsg[5], 528
 - calls selectOptionLC[5], 528
 - calls spadPrompt, 528
 - calls terminateSystemCommand[5], 528
 - calls throwKeyedMsg, 528
 - uses \$options, 528
 - defun, 528
- compileTimeBindingOf, 213
 - calledby optSpecialCall, 212
 - calls bpiname, 213
 - calls keyedSystemError, 213
 - calls moan, 213
 - defun, 213
- compImport, 304
 - calls addDomain, 304
 - uses \$NoValueMode, 304
 - defun, 304
- compInternalFunction, 279
 - calledby compDefine1, 275
 - calls identp, 279
 - calls stackAndThrow, 279
 - defun, 279
- compIs, 304
 - calls coerce, 304
 - calls comp, 304
 - uses \$Boolean, 304
 - uses \$EmptyMode, 304
 - defun, 304
- compIterator
 - calledby compReduce1, 312
 - calledby compRepeatOrCollect, 315
- compJoin, 305
 - calls compForMode, 305
 - calls compJoin, getParms, 305
 - calls convert, 305
 - calls isCategoryForm, 305
 - calls nreverse0, 305
 - calls qcar, 305
 - calls qcdr, 305
 - calls stackSemanticError, 305
 - calls union, 305
 - calls wrapDomainSub, 305

- uses \$Category, 305
 - defun, 305
- compJoin, getParms
 - calledby compJoin, 305
- compLambda, 307
 - calledby compWithMappingMode1, 518
 - calls argsToSig, 307
 - calls compAtSign, 307
 - calls qcar, 307
 - calls qcdr, 307
 - calls stackAndThrow, 307
 - defun, 307
- compLeave, 308
 - calls comp, 308
 - calls modifyModeStack, 308
 - uses \$exitModeStack, 308
 - uses \$leaveLevelStack, 309
 - defun, 308
- compList, 506
 - calledby compAtom, 503
 - calledby compConstruct, 272
 - calls comp, 506
 - defun, 506
- compMacro, 309
 - calls formatUnabbreviated, 309
 - calls macroExpand, 309
 - calls put, 309
 - calls qcar, 309
 - calls sayBrightly, 309
 - uses \$EmptyMode, 309
 - uses \$NoValueMode, 309
 - uses \$macroIfTrue, 309
 - defun, 309
- compMakeCategoryObject, 180
 - calledby compDefineFunctor1, 181
 - calledby getOperationAlist, 242
 - calledby getSlotFromFunctor, 179
 - calls isCategoryForm, 180
 - calls mkEvalableCategoryForm, 180
 - local ref \$Category, 180
 - local ref \$e, 180
 - defun, 180
- compMakeDeclaration, 525
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby compSetq1, 321
 - calledby compSubDomain1, 332
 - calledby compWithMappingMode1, 518
 - calls compColon, 525
 - uses \$insideExpressionIfTrue, 525
 - defun, 525
- compNoStacking, 498
 - calledby comp, 497
 - calls comp2, 498
 - calls compNoStacking1, 498
 - local ref \$compStack, 498
 - uses \$EmptyMode, 498
 - uses \$Representation, 498
 - defun, 498
- compNoStacking1, 498
 - calledby compNoStacking, 498
 - calls comp2, 498
 - calls get, 498
 - local ref \$compStack, 498
 - defun, 498
- compOrCroak, 496
 - calledby compAdd, 248
 - calledby compArgumentConditions, 286
 - calledby compDefineCapsuleFunction, 280
 - calledby compDefineCategory2, 142
 - calledby compForm1, 508
 - calledby compFunctorBody, 191
 - calledby compRepeatOrCollect, 315
 - calledby compSubDomain1, 332
 - calledby compTopLevel, 494
 - calledby doIt, 253
 - calledby getTargetFromRhs, 135
 - calledby makeCategoryForm, 270
 - calledby mkEvalableCategoryForm, 140
 - calls compOrCroak1, 496
 - defun, 496
- compOrCroak1, 496
 - calledby compOrCroak, 496
 - calls compOrCroak1, compactify, 496
 - calls comp, 496
 - calls displayComp, 497
 - calls displaySemanticErrors, 497
 - calls mkErrorExpr, 497
 - calls say, 497
 - calls stackSemanticError, 497
 - calls userError, 497
 - local def \$compStack, 497

- uses \$compErrorMessageStack, 497
 - uses \$exitModeStack, 497
 - uses \$level, 497
 - uses \$scanIfTrue, 497
 - uses \$s, 497
 - catches, 497
 - defun, 496
- compOrCroak1,compactify, 527
 - calledby compOrCroak1,compactify, 527
 - calledby compOrCroak1, 496
 - calls compOrCroak1,compactify, 527
 - calls lassoc, 527
 - defun, 527
- compPretend, 310
 - calls addDomain, 310
 - calls comp, 310
 - calls nequal, 310
 - calls opOf, 310
 - calls stackSemanticError, 310
 - calls stackWarning, 310
 - uses \$EmptyMode, 311
 - uses \$newCompilerUnionFlag, 311
 - defun, 310
- compQuote, 311
 - defun, 311
- compReduce, 312
 - calls compReduce1, 312
 - uses \$formalArgList, 312
 - defun, 312
- compReduce1, 312
 - calledby compReduce, 312
 - calls compIterator, 312
 - calls comp, 312
 - calls getIdentity, 312
 - calls msubst, 312
 - calls nreverse0, 312
 - calls parseTran, 312
 - calls systemError, 312
 - uses \$Boolean, 312
 - uses \$endTestList, 312
 - uses \$e, 312
 - uses \$initList, 312
 - uses \$sideEffectsList, 312
 - uses \$until, 312
 - defun, 312
- compRepeatOrCollect, 314
 - calls coerceExit, 315
 - calls compIterator, 315
 - calls compOrCroak, 315
 - calls comp, 315
 - calls length, 314
 - calls modelIsAggregateOf, 315
 - calls msubst, 315
 - calls stackMessage, 315
 - calls , 315
 - uses \$Boolean, 315
 - uses \$NoValueMode, 315
 - uses \$exitModeStack, 315
 - uses \$formalArgList, 315
 - uses \$leaveLevelStack, 315
 - uses \$until, 315
 - defun, 314
- compReturn, 317
 - calls comp, 317
 - calls modifyModeStack, 317
 - calls nequal, 317
 - calls resolve, 317
 - calls stackSemanticError, 317
 - calls userError, 317
 - uses \$exitModeStack, 317
 - uses \$returnMode, 317
 - defun, 317
- compSeq, 318
 - calls compSeq1, 318
 - uses \$exitModeStack, 318
 - defun, 318
- compSeq1, 318
 - calledby compSeq, 318
 - calls compSeqItem, 318
 - calls mkq, 318
 - calls nreverse0, 318
 - calls replaceExitEtc, 318
 - uses \$NoValueMode, 318
 - uses \$exitModeStack, 318
 - uses \$finalEnv, 318
 - uses \$insideExpressionIfTrue, 318
 - defun, 318
- compSeqItem, 320
 - calledby compSeq1, 318
 - calls comp, 320
 - calls macroExpand, 320
 - defun, 320

- compSetq, 321
 - calledby compSetq1, 321
 - calls compSetq1, 321
 - defun, 321
- compSetq1, 321
 - calledby compSetq, 321
 - calledby setqMultipleExplicit, 324
 - calledby setqMultiple, 322
 - calls compMakeDeclaration, 321
 - calls compSetq, 321
 - calls identp[5], 321
 - calls qcar, 321
 - calls qcdr, 321
 - calls setqMultiple, 321
 - calls setqSetelt, 321
 - calls setqSingle, 321
 - uses \$EmptyMode, 321
 - defun, 321
- compSingleCapsuleItem, 252
 - calledby compCapsuleItems, 252
 - calledby doItIf, 257
 - calledby doIt, 253
 - calls doit, 252
 - calls macroExpandInPlace, 252
 - local ref \$e, 252
 - local ref \$pred, 252
 - defun, 252
- compString, 331
 - calls resolve, 331
 - uses \$StringCategory, 331
 - defun, 331
- compSubDomain, 331
 - calls compCapsule, 331
 - calls compSubDomain1, 331
 - uses \$NRTaddForm, 331
 - uses \$addFormLhs, 331
 - uses \$addForm, 331
 - defun, 331
- compSubDomain1, 332
 - calledby compAdd, 248
 - calledby compSubDomain, 331
 - calls addDomain, 332
 - calls compMakeDeclaration, 332
 - calls compOrCroak, 332
 - calls evalAndRwriteLispForm, 332
 - calls lispize, 332
 - calls stackSemanticError, 332
 - uses \$Boolean, 332
 - uses \$CategoryFrame, 332
 - uses \$EmptyMode, 332
 - uses \$lisplibSuperDomain, 332
 - uses \$op, 332
 - defun, 332
- compSubsetCategory, 333
 - calls comp, 333
 - calls msubst, 333
 - calls put, 333
 - uses \$lhsOfColon, 333
 - defun, 333
- compSuchthat, 334
 - calls comp, 334
 - calls put, 334
 - uses \$Boolean, 334
 - defun, 334
- compSymbol, 505
 - calledby compAtom, 503
 - calls NRTgetLocalIndex, 505
 - calls errorRef, 505
 - calls getmode, 505
 - calls get, 505
 - calls isFunction, 505
 - calls member[5], 505
 - calls stackMessage, 505
 - uses \$Boolean, 505
 - uses \$Expression, 505
 - uses \$FormalMapVariableList, 505
 - uses \$NoValueMode, 505
 - uses \$NoValue, 505
 - uses \$Symbol, 505
 - uses \$compForModeIfTrue, 505
 - uses \$formalArgList, 505
 - uses \$functorLocalParameters, 505
 - defun, 505
- compToApply
 - calledby compForm1, 508
- compTopLevel, 494
 - calledby s-process, 491
 - calls compOrCroak, 494
 - calls newComp, 494
 - uses \$NRTderivedTargetIfTrue, 494
 - uses \$compTimeSum, 494
 - uses \$envHashTable, 494

- uses \$forceAdd, 494
 - uses \$killOptimizeIfTrue, 494
 - uses \$packagesUsed, 494
 - uses \$resolveTimeSum, 494
 - defun, 494
- compTuple2Record, 249
 - calledby compAdd, 248
 - defun, 249
- compTypeOf, 502
 - calledby comp3, 500
 - calls comp3, 502
 - calls eqsubstlist, 502
 - calls get, 502
 - calls put, 502
 - uses \$FormalMapVariableList, 502
 - uses \$insideCompTypeOf, 502
 - defun, 502
- compUniquely, 516
 - calledby compForm2, 513
 - calls comp, 516
 - local def \$compUniquelyIfTrue, 516
 - catches, 516
 - defun, 516
- computeAncestorsOf
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
- compVector, 335
 - calledby compAtom, 503
 - calledby compConstruct, 272
 - calls comp, 335
 - uses \$EmptyVector, 335
 - defun, 335
- compWhere, 336
 - calls addContour, 336
 - calls comp, 336
 - calls deltaContour, 336
 - calls macroExpand, 336
 - uses \$EmptyMode, 336
 - uses \$insideExpressionIfTrue, 336
 - uses \$insideWhereIfTrue, 336
 - defun, 336
- compWithMappingMode, 517
 - calledby comp3, 500
 - calls compWithMappingMode1, 517
 - uses \$formalArgList, 517
 - defun, 517
- compWithMappingMode1, 518
 - calledby compWithMappingMode, 517
 - calls comp-tran, 518
 - calls compLambda, 518
 - calls compMakeDeclaration, 518
 - calls comp, 518
 - calls extendsCategoryForm, 518
 - calls extractCodeAndConstructTriple, 518
 - calls freelist, 518
 - calls get, 518
 - calls hasFormalMapVariable, 518
 - calls isFunctor, 518
 - calls optimizeFunctionDef, 518
 - calls qcar, 518
 - calls qcdr, 518
 - calls stackAndThrow, 518
 - calls take, 518
 - uses \$CategoryFrame, 518
 - uses \$EmptyMode, 518
 - uses \$FormalMapVariableList, 518
 - uses \$QuickCode, 518
 - uses \$formalArgList, 518
 - uses \$formatArgList, 518
 - uses \$funnameTail, 518
 - uses \$funname, 518
 - uses \$killOptimizeIfTrue, 518
 - defun, 518
- concat
 - calledby checkWarning, 461
 - calledby compDefWhereClause, 200
- cond, 222
 - defplist, 222
- cons, 270
 - defplist, 270
- consProplistOf
 - calledby getSuccessEnvironment, 300
 - calledby setqSingle, 326
- construct, 95, 272, 369
 - defplist, 95, 272, 369
- constructMacro, 151
 - calledby compile, 145
 - calls identp, 151
 - calls stackSemanticError, 151
 - defun, 151
- constructor?
 - calledby addDomain, 228

- calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby getAbbreviation, 277
 - calledby isFunctor, 229
- contained
 - calledby evalAndSub, 241
 - calledby parseCategory, 99
 - calledby spadCompileOrSetq, 152
 - calledby substVars, 166
- containsBang, 390
 - calledby apITran, 387
 - calledby containsBang, 390
 - calls containsBang, 390
 - defun, 390
- convert, 504
 - calledby compAtom, 503
 - calledby compCons1, 271
 - calledby compConstruct, 272
 - calledby compElt, 292
 - calledby compExpressionList, 512
 - calledby compJoin, 305
 - calledby convertOrCroak, 320
 - calledby setqMultiple, 323
 - calledby setqSingle, 326
 - calls coerce, 504
 - calls resolve, 504
 - defun, 504
- convertOpAlist2compilerInfo, 112
 - calledby updateCategoryFrameForConstructor, 112
 - defun, 112
- convertOrCroak, 320
 - calledby replaceExitEtc, 319
 - calls convert, 320
 - calls userError, 320
 - defun, 320
- copy
 - calledby modifyModeStack, 525
- copy-token
 - calledby PARSE-TokTail, 416
 - calledby advance-token, 448
- croak
 - calledby drop, 465
- curoutstream
 - usedby s-process, 491
 - usedby spad, 489
- current-char, 449
 - calledby Advance-Char, 539
 - calledby PARSE-FloatBasePart, 423
 - calledby PARSE-FloatBase, 422
 - calledby PARSE-FloatExponent, 423
 - calledby PARSE-Selector, 419
 - calledby PARSE-TokTail, 416
 - calledby match-string, 440
 - calledby skip-blanks, 440
 - uses \$line, 449
 - uses current-line, 449
 - defun, 449
- current-fragment, 533
 - defvar, 533
- current-line, 536
 - usedby PARSE-Category, 410
 - usedby current-char, 449
 - usedby next-char, 450
 - defvar, 536
- current-symbol, 446
 - calledby PARSE-AnyId, 430
 - calledby PARSE-ElseClause, 437
 - calledby PARSE-FloatBase, 422
 - calledby PARSE-FloatExponent, 423
 - calledby PARSE-Infix, 415
 - calledby PARSE-NewExpr, 403
 - calledby PARSE-OpenBrace, 432
 - calledby PARSE-OpenBracket, 432
 - calledby PARSE-Operation, 413
 - calledby PARSE-Prefix, 414
 - calledby PARSE-Primary1, 420
 - calledby PARSE-ReductionOp, 417
 - calledby PARSE-Selector, 419
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-SpecialKeyword, 404
 - calledby PARSE-Suffix, 434
 - calledby PARSE-TokTail, 416
 - calledby PARSE-TokenList, 406
 - calledby isTokenDelimiter, 443
 - calls current-token, 446
 - calls make-symbol-of, 446
 - defun, 446
- current-token, 91, 447
 - calledby PARSE-FloatBasePart, 423
 - calledby PARSE-SpecialKeyword, 404
 - calledby advance-token, 448

- calledby current-symbol, 446
- calledby match-advance-string, 441
- calledby match-current-token, 445
- calledby next-token, 448
- calls try-get-token, 447
- usedby advance-token, 448
- usedby current-token, 447
- uses \$token, 91
- uses current-token, 447
- uses valid-tokens, 447
- defun, 447
- defvar, 91
- curstrm
 - calledby s-process, 490
- dcq
 - calledby preparseReadLine, 85
- decodeScripts, 391
 - calledby decodeScripts, 391
 - calledby getScriptName, 391
 - calls decodeScripts, 391
 - calls qcar, 391
 - calls qcdr, 391
 - calls strconc, 391
 - defun, 391
- deepestExpression, 390
 - calledby deepestExpression, 390
 - calledby hasAplExtension, 389
 - calls deepestExpression, 390
 - defun, 390
- def, 101, 274
 - defplist, 101, 274
- def-process
 - calledby s-process, 491
- def-rename, 493
 - calledby s-process, 490
 - calls def-rename1, 493
 - defun, 493
- def-rename1, 494
 - calledby def-rename1, 494
 - calledby def-rename, 493
 - calls def-rename1, 494
 - defun, 494
- definition-name, 403
 - usedby PARSE-NewExpr, 403
 - defvar, 403
- defmacro
 - Bang, 452
 - line-clear, 536
 - must, 452
 - nth-stack, 464
 - pop-stack-1, 462
 - pop-stack-2, 463
 - pop-stack-3, 463
 - pop-stack-4, 463
 - reduce-stack-clear, 454
 - stack-/empty, 89
 - star, 453
- defplist, 97, 220, 247, 334, 342, 360
 - + - >, 307
 - >, 376
 - <=, 121
 - ==>, 377
 - =>, 373
 - >, 108
 - >=, 106, 107
 - „, 368
 - , 217
 - /, 383
 - :, 100, 266, 366
 - ::, 99, 343, 367
 - :BF:, 361
 - ;;, 381
 - ==, 370
 - add, 358
 - and, 97
 - Block, 362
 - call, 209
 - capsule, 250
 - case, 259
 - catch, 220
 - category, 98, 262, 363
 - collect, 314, 365
 - cond, 222
 - cons, 270
 - construct, 95, 272, 369
 - def, 101, 274
 - dollargreaterequal, 104
 - dollargreaterthan, 104
 - dollarnotequal, 105
 - elt, 291
 - eq, 215

- eqv, 106
- exit, 293
- has, 108, 294
- if, 113, 296, 373
- implies, 116
- import, 303
- In, 375
- in, 117, 374
- inby, 118
- is, 119, 304
- isnt, 119
- Join, 120, 305, 375
- leave, 121, 308
- lessp, 218
- let, 122, 320
- letd, 123
- ListCategory, 273
- Mapping, 261
- mdef, 123, 309
- minus, 216
- mkRecord, 225
- not, 124
- notequal, 125
- or, 125
- pretend, 126, 310, 378
- qsminus, 217
- quote, 311, 379
- Record, 261
- RecordCategory, 273
- recordcopy, 227
- recordelt, 225
- reduce, 312, 379
- repeat, 314, 380
- return, 127, 316
- Scripts, 380
- segment, 127, 128
- seq, 214, 318
- setq, 321
- setrecordelt, 226
- Signature, 382
- spadcall, 219
- String, 330
- SubDomain, 331
- SubsetCategory, 333
- TupleCollect, 384
- Union, 261
- UnionCategory, 273
- vcons, 129
- vector, 335
- VectorCategory, 273
- where, 129, 336, 385
- with, 386
- defstruct
 - line, 535
 - reduction, 92
 - stack, 88
 - token, 90
- defun
 - /RQ,LIB, 479
 - /rf-1, 480
 - action, 453
 - add-parens-and-semis-to-line, 84
 - addArgumentConditions, 285
 - addclose, 464
 - addConstructorModemaps, 234
 - addDomain, 228
 - addEltModemap, 237
 - addEmptyCapsuleIfNecessary, 134
 - addModemap, 245
 - addModemap0, 246
 - addModemap1, 246
 - addModemapKnown, 245
 - addNewDomain, 231
 - addSuffix, 278
 - Advance-Char, 539
 - advance-token, 448
 - alistSize, 278
 - allLASSOCs, 190
 - aplTran, 387
 - aplTran1, 387
 - aplTranList, 389
 - argsToSig, 524
 - assignError, 328
 - AssocBarGensym, 200
 - augLisplibModemapsFromCategory, 155
 - augmentLisplibModemapsFromFunctor, 189
 - augModemapsFromCategory, 237
 - augModemapsFromCategoryRep, 243
 - augModemapsFromDomain, 232
 - augModemapsFromDomain1, 232
 - autoCoerceByModemap, 345
 - blankp, 465

- bootStrapError, 192
- bumperrorcount, 457
- canReturn, 298
- char-eq, 450
- char-ne, 450
- checkAndDeclare, 289
- checkWarning, 461
- coerce, 337
- coerceable, 341
- coerceByModemap, 345
- coerceEasy, 338
- coerceExit, 342
- coerceExtraHard, 340
- coerceHard, 339
- coerceSubset, 338
- comma2Tuple, 368
- comp, 497
- comp2, 499
- comp3, 500
- compAdd, 247
- compArgumentConditions, 286
- compArgumentsAndTryAgain, 517
- compAtom, 503
- compAtSign, 343
- compBoolean, 300
- compCapsule, 250
- compCapsuleInner, 250
- compCapsuleItems, 252
- compCase, 259
- compCase1, 260
- compCat, 261
- compCategory, 262
- compCategoryItem, 263
- compCoerce, 343
- compCoerce1, 344
- compColon, 267
- compColonInside, 502
- compCons, 270
- compCons1, 271
- compConstruct, 272
- compConstructorCategory, 274
- compDefine, 274
- compDefine1, 275
- compDefineAddSignature, 133
- compDefineCapsuleFunction, 280
- compDefineCategory, 169
- compDefineCategory1, 137
- compDefineCategory2, 142
- compDefineFunctor, 181
- compDefineFunctor1, 181
- compDefineLisplib, 169
- compDefWhereClause, 200
- compElt, 292
- compExit, 293
- compExpression, 507
- compExpressionList, 512
- compForm, 507
- compForm1, 508
- compForm2, 513
- compForm3, 515
- compFormMatch, 516
- compForMode, 307
- compFormPartiallyBottomUp, 516
- compFromIf, 297
- compFunctorBody, 191
- compHas, 294
- compHasFormat, 295
- compIf, 296
- compile, 145
- compile-lib-file, 530
- compileCases, 283
- compileConstructor, 153
- compileConstructor1, 153
- compileDocumentation, 172
- compileFileQuietly, 530
- compiler, 472
- compilerDoit, 478
- compileSpad2Cmd, 474
- compileSpadLispCmd, 528
- compileTimeBindingOf, 213
- compImport, 304
- compInternalFunction, 279
- compIs, 304
- compJoin, 305
- compLambda, 307
- compLeave, 308
- compList, 506
- compMacro, 309
- compMakeCategoryObject, 180
- compMakeDeclaration, 525
- compNoStacking, 498
- compNoStacking1, 498

- compOrCroak, 496
- compOrCroak1, 496
- compOrCroak1,compactify, 527
- compPretend, 310
- compQuote, 311
- compReduce, 312
- compReduce1, 312
- compRepeatOrCollect, 314
- compReturn, 317
- compSeq, 318
- compSeq1, 318
- compSeqItem, 320
- compSetq, 321
- compSetq1, 321
- compSingleCapsuleItem, 252
- compString, 331
- compSubDomain, 331
- compSubDomain1, 332
- compSubsetCategory, 333
- compSuchthat, 334
- compSymbol, 505
- compTopLevel, 494
- compTuple2Record, 249
- compTypeOf, 502
- compUniquely, 516
- compVector, 335
- compWhere, 336
- compWithMappingMode, 517
- compWithMappingMode1, 518
- constructMacro, 151
- containsBang, 390
- convert, 504
- convertOpAlist2compilerInfo, 112
- convertOrCroak, 320
- current-char, 449
- current-symbol, 446
- current-token, 447
- decodeScripts, 391
- deepestExpression, 390
- def-rename, 493
- def-rename1, 494
- disallowNilAttribute, 191
- displayMissingFunctions, 193
- displayPreCompilationErrors, 456
- doIt, 253
- doItIf, 257
- dollarTran, 451
- domainMember, 236
- drop, 465
- eltModemapFilter, 511
- encodeFunctionName, 148
- encodeItem, 150
- EqualBarGensym, 224
- errhuh, 395
- escape-keywords, 443
- escaped, 465
- evalAndRwriteLispForm, 168
- evalAndSub, 241
- extractCodeAndConstructTriple, 523
- finalizeLisplib, 174
- fincomblock, 466
- fixUpPredicate, 160
- flattenSignatureList, 158
- floatexpid, 451
- formal2Pattern, 190
- freelist, 526
- genDomainOps, 198
- genDomainView, 197
- genDomainViewList, 196
- genDomainViewList0, 196
- get-a-line, 540
- get-token, 449
- getAbbreviation, 277
- getArgumentMode, 291
- getArgumentModeOrMoan, 154
- getCaps, 150
- getCategoryOpsAndAtts, 177
- getConstructorOpsAndAtts, 176
- getDomainsInScope, 230
- getFormModemaps, 510
- getFunctorOpsAndAtts, 179
- getInverseEnvironment, 301
- getModemap, 234
- getModemapList, 235
- getModemapListFromDomain, 236
- getOperationAlist, 242
- getScriptName, 391
- getSignature, 288
- getSignatureFromMode, 278
- getSlotFromCategoryForm, 177
- getSlotFromFunctor, 179
- getSpecialCaseAssoc, 285

- getSuccessEnvironment, 300
- getTargetFromRhs, 135
- getToken, 444
- getUnionMode, 303
- getUniqueModemap, 235
- getUniqueSignature, 235
- giveFormalParametersValues, 135
- hackforis, 394
- hackforis1, 395
- hasAplExtension, 389
- hasFormalMapVariable, 524
- hasFullSignature, 134
- hasSigInTargetCategory, 290
- hasType, 341
- indent-pos, 466
- infixtok, 467
- initial-substring, 539
- initial-substring-p, 442
- initialize-preparse, 73
- initializeLisplib, 173
- insertModemap, 239
- interactiveModemapForm, 158
- is-console, 467
- isDomainConstructorForm, 330
- isDomainForm, 329
- isDomainSubst, 164
- isFunctor, 229
- isListConstructor, 103
- isMacro, 259
- isSuperDomain, 231
- isTokenDelimiter, 443
- isUnionMode, 303
- killColons, 383
- line-advance-char, 537
- line-at-end-p, 536
- line-current-segment, 538
- line-new-line, 538
- line-next-char, 537
- line-past-end-p, 537
- line-print, 536
- lispize, 333
- lisplibDoRename, 172
- lisplibWrite, 180
- loadIfNecessary, 111
- loadLibIfNecessary, 111
- macroExpand, 136
- macroExpandInPlace, 136
- macroExpandList, 137
- make-symbol-of, 446
- makeCategoryForm, 270
- makeCategoryPredicates, 138
- makeFunctorArgumentParameters, 194
- makeSimplePredicateOrNil, 458
- match-advance-string, 441
- match-current-token, 445
- match-next-token, 446
- match-string, 440
- match-token, 446
- maxSuperType, 329
- mergeModemap, 239
- mergeSignatureAndLocalVarAlists, 180
- meta-syntax-error, 451
- mkAbbrev, 277
- mkAlistOfExplicitCategoryOps, 156
- mkCategoryPackage, 139
- mkConstructor, 168
- mkDatabasePred, 191
- mkEvalableCategoryForm, 140
- mkExplicitCategoryFunction, 265
- mkList, 296
- mkNewModemapList, 238
- mkOpVec, 199
- mkRepetitionAssoc, 149
- mkUnion, 347
- modeEqual, 348
- modeEqualSubst, 348
- modemapPattern, 167
- modifyModeStack, 525
- moveORsOutside, 165
- mustInstantiate, 266
- ncINTERPFILE, 527
- new2OldLisp, 458
- next-char, 449
- next-line, 538
- next-tab-loc, 467
- next-token, 448
- nonblankloc, 468
- opt-, 218
- optCall, 209
- optCallEval, 213
- optCallSpecially, 211
- optCatch, 220

- optCond, 222
- optCONDtail, 206
- optEQ, 216
- optIF2COND, 207
- optimize, 205
- optimizeFunctionDef, 203
- optional, 453
- optLESSP, 218
- optMINUS, 216
- optMkRecord, 225
- optPackageCall, 210
- optPredicateIfTrue, 207
- optQSMINUS, 217
- optRECORDCOPY, 227
- optRECORDELT, 225
- optSEQ, 214
- optSETRECORDELT, 226
- optSPADCALL, 219
- optSpecialCall, 212
- optSuchthat, 220
- optXLAMCond, 206
- orderByDependency, 202
- orderPredicateItems, 161
- orderPredTran, 162
- outputComp, 328
- PARSE-AnyId, 430
- PARSE-Application, 418
- parse-argument-designator, 460
- PARSE-Category, 409
- PARSE-Command, 404
- PARSE-CommandTail, 407
- PARSE-Conditional, 437
- PARSE-Data, 428
- PARSE-ElseClause, 437
- PARSE-Enclosure, 424
- PARSE-Exit, 435
- PARSE-Expr, 412
- PARSE-Expression, 411
- PARSE-Float, 421
- PARSE-FloatBase, 422
- PARSE-FloatBasePart, 422
- PARSE-FloatExponent, 423
- PARSE-FloatTok, 439
- PARSE-Form, 417
- PARSE-FormalParameter, 425
- PARSE-FormalParameterTok, 425
- PARSE-getSemanticForm, 414
- PARSE-GlyphTok, 430
- parse-identifier, 459
- PARSE-Import, 411
- PARSE-Infix, 415
- PARSE-InfixWith, 409
- PARSE-IntegerTok, 424
- PARSE-Iterator, 433
- PARSE-IteratorTail, 433
- parse-keyword, 460
- PARSE-Label, 419
- PARSE-LabelExpr, 438
- PARSE-Leave, 436
- PARSE-LedPart, 412
- PARSE-leftBindingPowerOf, 413
- PARSE-Loop, 438
- PARSE-Name, 427
- PARSE-NBGlyphTok, 429
- PARSE-NewExpr, 403
- PARSE-NudPart, 412
- parse-number, 460
- PARSE-OpenBrace, 432
- PARSE-OpenBracket, 432
- PARSE-Operation, 413
- PARSE-Option, 408
- PARSE-Prefix, 414
- PARSE-Primary, 420
- PARSE-Primary1, 420
- PARSE-PrimaryNoFloat, 420
- PARSE-PrimaryOrQM, 407
- PARSE-Quad, 425
- PARSE-Qualification, 416
- PARSE-Reduction, 417
- PARSE-ReductionOp, 417
- PARSE-Return, 435
- PARSE-rightBindingPowerOf, 414
- PARSE-ScriptItem, 427
- PARSE-Scripts, 426
- PARSE-Seg, 436
- PARSE-Selector, 419
- PARSE-SemiColon, 435
- PARSE-Sequence, 431
- PARSE-Sequence1, 431
- PARSE-Sexpr, 428
- PARSE-Sexpr1, 428
- parse-spadstring, 458

- PARSE-SpecialCommand, 405
- PARSE-SpecialKeyword, 404
- PARSE-Statement, 408
- PARSE-String, 425
- parse-string, 459
- PARSE-Suffix, 434
- PARSE-TokenCommandTail, 405
- PARSE-TokenList, 406
- PARSE-TokenOption, 406
- PARSE-TokTail, 416
- PARSE-VarForm, 426
- PARSE-With, 409
- parseAnd, 97
- parseAtom, 94
- parseAtSign, 98
- parseCategory, 99
- parseCoerce, 100
- parseColon, 100
- parseConstruct, 95
- parseDEF, 101
- parseDollarGreaterEqual, 104
- parseDollarGreaterThan, 104
- parseDollarLessEqual, 105
- parseDollarNotEqual, 105
- parseDropAssertions, 99
- parseEquivalence, 106
- parseExit, 107
- parseGreaterEqual, 107
- parseGreaterThan, 108
- parseHas, 108
- parseHasRhs, 110
- parseIf, 114
- parseIf,ifTran, 114
- parseImplies, 116
- parseIn, 117
- parseInBy, 118
- parseIs, 119
- parseIsnt, 120
- parseJoin, 120
- parseLeave, 121
- parseLessEqual, 122
- parseLET, 122
- parseLETD, 123
- parseLhs, 102
- parseMDEF, 123
- parseNot, 124
- parseNotEqual, 125
- parseOr, 125
- parsepiles, 84
- parsePretend, 126
- parseprint, 468
- parseReturn, 127
- parseSegment, 128
- parseSeq, 128
- parseTran, 93
- parseTranCheckForRecord, 457
- parseTranList, 95
- parseTransform, 93
- parseType, 98
- parseVCONS, 129
- parseWhere, 129
- Pop-Reduction, 464
- postAdd, 358
- postAtom, 353
- postAtSign, 361
- postBigFloat, 362
- postBlock, 362
- postBlockItem, 360
- postBlockItemList, 359
- postCapsule, 359
- postCategory, 363
- postcheck, 355
- postCollect, 365
- postCollect,finish, 364
- postColon, 367
- postColonColon, 367
- postComma, 368
- postConstruct, 369
- postDef, 370
- postDefArgs, 372
- postError, 356
- postExit, 373
- postFlatten, 368
- postFlattenLeft, 381
- postForm, 356
- postIf, 373
- postIn, 375
- postin, 374
- postInSeq, 374
- postIteratorList, 366
- postJoin, 376
- postMakeCons, 364

- postMapping, 376
- postMDef, 377
- postOp, 353
- postPretend, 378
- postQUOTE, 379
- postReduce, 379
- postRepeat, 380
- postScripts, 381
- postScriptsForm, 354
- postSemiColon, 381
- postSignature, 382
- postSlash, 383
- postTran, 352
- postTranList, 354
- postTranScripts, 354
- postTranSegment, 370
- postTransform, 351
- postTransformCheck, 355
- postTuple, 384
- postTupleCollect, 385
- postType, 361
- postWhere, 385
- postWith, 386
- preparse, 76
- preparse-echo, 88
- preparse1, 81
- preparseReadLine, 85
- preparseReadLine1, 87
- primitiveType, 504
- print-defun, 493
- print-package, 461
- processFunctor, 251
- push-reduction, 454
- putDomainsInScope, 230
- putInLocalDomainReferences, 154
- quote-if-string, 442
- read-a-line, 533
- recompile-lib-file-if-necessary, 529
- removeSuperfluousMapping, 383
- replaceExitEtc, 319
- replaceVars, 159
- reportOnFunctorCompilation, 192
- resolve, 347
- rwriteLispForm, 168
- s-process, 490
- setDefOp, 386
- seteltModemapFilter, 512
- setqMultiple, 322
- setqMultipleExplicit, 324
- setqSetelt, 326
- setqSingle, 326
- signatureTran, 161
- skip-blanks, 440
- skip-ifblock, 86
- skip-to-endif, 468
- spad, 489
- spad-fixed-arg, 529
- spadCompileOrSetq, 151
- splitEncodedFunctionName, 149
- stack-clear, 89
- stack-load, 89
- stack-pop, 90
- stack-push, 89
- storeblanks, 539
- stripOffArgumentConditions, 287
- stripOffSubdomainConditions, 287
- subrname, 208
- substituteCategoryArguments, 233
- substNames, 243
- substVars, 166
- token-install, 92
- token-lookahead-type, 441
- token-print, 92
- transformOperationAlist, 177
- transIs, 102
- transIs1, 102
- translabel, 455
- translabel1, 455
- TruthP, 241
- try-get-token, 447
- tuple2List, 461
- uncons, 322
- underscore, 444
- unget-tokens, 444
- unknownTypeError, 229
- unTuple, 395
- updateCategoryFrameForCategory, 113
- updateCategoryFrameForConstructor, 112
- wrapDomainSub, 266
- writeLib1, 174
- defvar
 - \$BasicPredicates, 207

- \$EmptyMode, 131
- \$FormalMapVariableList, 242
- \$NoValueMode, 131
- \$byConstructors, 531
- \$comblocklist, 465
- \$constructorsSeen, 531
- \$defstack, 393
- \$echolinestack, 72
- \$index, 72
- \$is-eqlist, 394
- \$is-gensymlist, 394
- \$is-spill-list, 393
- \$is-spill, 393
- \$linelist, 72
- \$preparse-last-line, 72
- \$vl, 394
- current-fragment, 533
- current-line, 536
- current-token, 91
- definition-name, 403
- initial-gensym, 394
- lablasoc, 403
- meta-error-handler, 450
- next-token, 91
- nonblank, 91
- ParseMode, 403
- prior-token, 90
- reduce-stack, 454
- tmptok, 402
- tok, 402
- valid-tokens, 91
- XTokenReader, 449
- delete
 - calledby compDefWhereClause, 200
 - calledby getInverseEnvironment, 302
 - calledby orderPredTran, 162
 - calledby putDomainsInScope, 230
- deltaContour
 - calledby compWhere, 336
- digitp[5]
 - called by PARSE-FloatBasePart, 423
 - called by PARSE-FloatBase, 422
 - called by floatexpid, 451
- disallowNilAttribute, 191
 - calledby compDefineFunctor1, 182
 - defun, 191
- displayComp
 - calledby compOrCroak1, 497
- displayMissingFunctions, 193
 - calledby reportOnFunctorCompilation, 192
 - calls bright, 193
 - calls formatUnabbreviatedSig, 193
 - calls getmode, 193
 - calls member, 193
 - calls sayBrightly, 193
 - uses \$CheckVectorList, 193
 - uses \$env, 193
 - uses \$formalArgList, 193
 - defun, 193
- displayPreCompilationErrors, 456
 - calledby s-process, 490
 - calls length, 456
 - calls nequal, 456
 - calls remdup, 456
 - calls sayBrightly, 456
 - calls sayMath, 456
 - local ref \$InteractiveMode, 456
 - local ref \$postStack, 456
 - local ref \$stopOp, 456
 - defun, 456
- displaySemanticErrors
 - calledby compOrCroak1, 497
 - calledby reportOnFunctorCompilation, 192
 - calledby s-process, 491
- displayWarnings
 - calledby reportOnFunctorCompilation, 192
- doIt, 253
 - calledby doIt, 253
 - calls NRTgetLocalIndexClear, 253
 - calls NRTgetLocalIndex, 253
 - calls bright, 253
 - calls cannotDo, 253
 - calls compOrCroak, 253
 - calls compSingleCapsuleItem, 253
 - calls doItIf, 253
 - calls doIt, 253
 - calls formatUnabbreviated, 253
 - calls get, 253
 - calls insert, 253
 - calls isDomainForm, 253
 - calls isMacro, 253
 - calls kar, 253

- calls lastnode, 253
- calls member, 253
- calls opOf, 253
- calls put, 253
- calls qcar, 253
- calls qcdr, 253
- calls sayBrightly, 253
- calls stackSemanticError, 253
- calls stackWarning, 253
- calls sublis, 253
- local def \$LocalDomainAlist, 254
- local def \$Representation, 254
- local def \$e, 254
- local def \$functorLocalParameters, 254
- local def \$functorsUsed, 254
- local def \$genno, 254
- local def \$packagesUsed, 254
- local ref \$EmptyMode, 253
- local ref \$LocalDomainAlist, 253
- local ref \$NRTopt, 253
- local ref \$NonMentionableDomainNames, 253
- local ref \$QuickCode, 254
- local ref \$Representation, 253
- local ref \$e, 253
- local ref \$functorLocalParameters, 253
- local ref \$functorsUsed, 253
- local ref \$packagesUsed, 253
- local ref \$predl, 253
- local ref \$signatureOfForm, 254
- defun, 253
- doit
 - calledby compSingleCapsuleItem, 252
- doItIf, 257
 - calledby doIt, 253
 - calls compSingleCapsuleItem, 257
 - calls comp, 257
 - calls getSuccessEnvironment, 257
 - calls localExtras, 257
 - calls rplaca, 257
 - calls rplacd, 257
 - calls userError, 257
 - local def \$e, 257
 - local def \$functorLocalParameters, 257
 - local ref \$Boolean, 257
 - local ref \$e, 257
- local ref \$functorLocalParameters, 257
- local ref \$getDomainCode, 257
- local ref \$predl, 257
- defun, 257
- dollargreaterequal, 104
 - defplist, 104
- dollargreaterthan, 104
 - defplist, 104
- dollarnotequal, 105
 - defplist, 105
- dollarTran, 451
 - calledby PARSE-Qualification, 416
 - uses \$InteractiveMode, 451
 - defun, 451
- domainMember, 236
 - calledby addDomain, 228
 - calls modeEqual, 236
 - defun, 236
- doSystemCommand[5]
 - called by preparse1, 81
- drop, 465
 - calledby add-parens-and-semis-to-line, 84
 - calledby drop, 465
 - calls croak, 465
 - calls drop, 465
 - calls take, 465
 - defun, 465
- Echo-Meta
 - usedby preparse-echo, 88
- echo-meta
 - usedby /rf-1, 480
 - usedby spad, 489
- echo-meta[5]
 - called by /RQ,LIB, 479
- editfile
 - usedby compCapsule, 250
- elapsedTime
 - calledby compile, 146
- elemn
 - calledby PARSE-Operation, 413
 - calledby PARSE-leftBindingPowerOf, 414
 - calledby PARSE-rightBindingPowerOf, 414
- elt, 291
 - defplist, 291
- eltModemapFilter, 511

- calledby getFormModemaps, 510
- calls isConstantId, 511
- calls qcar, 511
- calls qcdr, 511
- calls stackMessage, 511
- defun, 511
- embed
 - calledby compilerDoitWithScreenedLispliberase 349
- encodeFunctionName, 148
 - calledby compile, 145
 - calls encodeItem, 148
 - calls getAbbreviation, 148
 - calls internl, 148
 - calls length, 148
 - calls mkRepetitionAssoc, 148
 - calls msubst, 148
 - calls stringimage, 148
 - local def \$lisplibSignatureAlist, 148
 - local ref \$lisplibSignatureAlist, 148
 - local ref \$lisplib, 148
 - defun, 148
- encodeItem, 150
 - calledby compile, 145
 - calledby encodeFunctionName, 148
 - calls getCaps, 150
 - calls identp, 150
 - calls pname, 150
 - calls qcar, 150
 - calls stringimage, 150
 - defun, 150
- eq, 215
 - defplist, 215
- eqcar
 - calledby PARSE-OpenBrace, 432
 - calledby PARSE-OpenBracket, 432
 - calledby getToken, 444
 - calledby hackforis1, 395
- eqsubstlist
 - calledby compColon, 267
 - calledby compTypeOf, 502
 - calledby getSignatureFromMode, 279
 - calledby isDomainConstructorForm, 330
 - calledby substNames, 243
- EqualBarGensym, 224
 - calledby AssocBarGensym, 200
- calledby optCond, 222
- calls gensymp, 224
- local def \$GensymAssoc, 224
- local ref \$GensymAssoc, 224
- defun, 224
- eqv, 106
 - defplist, 106
- calledby initializeLisplib, 173
- errhuh, 395
 - calls systemError, 395
 - defun, 395
- error
 - calledby compileSpad2Cmd, 476
 - calledby processFunctor, 251
- errorRef
 - calledby compSymbol, 505
- errors
 - usedby initializeLisplib, 173
- Escape-Character
 - usedby token-lookahead-type, 441
- escape-keywords, 443
 - calledby quote-if-string, 442
 - local ref \$keywords, 443
 - defun, 443
- escaped, 465
 - calledby preparse1, 81
 - defun, 465
- eval
 - calledby coerceSubset, 338
 - calledby compDefineCategory2, 142
 - calledby compileCases, 283
 - calledby evalAndRwriteLispForm, 168
 - calledby getSlotFromCategoryForm, 177
 - calledby optCallEval, 213
- evalAndRwriteLispForm, 168
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby compSubDomain1, 332
 - calls eval, 168
 - calls rwriteLispForm, 168
 - defun, 168
- evalAndSub, 241
 - calledby augModemapsFromCategoryRep, 243
 - calledby augModemapsFromCategory, 237

- calls contained, 241
- calls getOperationAlist, 241
- calls get, 241
- calls isCategory, 241
- calls put, 241
- calls substNames, 241
- local def \$lhsOfColon, 241
- defun, 241
- exit, 293
 - defplist, 293
- expand-tabs
 - calledby prepareReadLine1, 87
- extendLocalLibdb
 - calledby compileSpad2Cmd, 476
- extendsCategoryForm
 - calledby coerceHard, 339
 - calledby compWithMappingModel1, 518
- extractCodeAndConstructTriple, 523
 - calledby compWithMappingModel1, 518
 - defun, 523
- FactoredForm
 - calledby optCallEval, 213
- File-Closed
 - usedby read-a-line, 533
- file-closed
 - usedby spad, 489
- filep
 - calledby compDefineLisplib, 170
- fillerSpaces
 - calledby compDefineLisplib, 169
- finalizeDocumentation
 - calledby compileDocumentation, 172
 - calledby finalizeLisplib, 174
- finalizeLisplib, 174
 - calledby compDefineLisplib, 169
 - calls NRTgenInitialAttributeAlist, 174
 - calls finalizeDocumentation, 174
 - calls getConstructorOpsAndAtts, 174
 - calls lisplibWrite, 174
 - calls makeprop, 174
 - calls mergeSignatureAndLocalVarAlists, 174
 - calls namestring, 174
 - calls profileWrite, 174
 - calls removeZeroOne, 174
 - calls sayMSG, 174
 - local def \$NRTslot1PredicateList, 175
 - local def \$lisplibCategory, 175
 - local def \$pairlis, 175
 - local ref \$/editfile, 174
 - local ref \$FormalMapVariableList, 175
 - local ref \$libFile, 174
 - local ref \$lisplibAbbreviation, 175
 - local ref \$lisplibAncestors, 175
 - local ref \$lisplibAttributes, 175
 - local ref \$lisplibCategory, 174
 - local ref \$lisplibForm, 174, 175
 - local ref \$lisplibKind, 174
 - local ref \$lisplibModemapAlist, 175
 - local ref \$lisplibModemap, 174, 175
 - local ref \$lisplibParents, 175
 - local ref \$lisplibPredicates, 175
 - local ref \$lisplibSignatureAlist, 175
 - local ref \$lisplibSlot1, 175
 - local ref \$lisplibSuperDomain, 175
 - local ref \$lisplibVariableAlist, 175
 - local ref \$profileCompiler, 175
 - local ref \$spadLibFT, 175
 - defun, 174
- fincomblock, 466
 - calledby prepare1, 81
 - calls prepare-echo, 466
 - uses \$EchoLineStack, 466
 - uses \$comblocklist, 466
 - defun, 466
- findfile
 - calledby compiler, 473
- fixUpPredicate, 160
 - calledby interactiveModemapForm, 158
 - calls length, 160
 - calls moveORsOutside, 160
 - calls orderPredicateItems, 160
 - calls qcar, 160
 - calls qcdr, 160
 - defun, 160
- flattenSignatureList, 158
 - calledby flattenSignatureList, 158
 - calledby mkAlistOfExplicitCategoryOps, 156
 - calls flattenSignatureList, 158
 - calls qcar, 158

- calls qcdr, 158
- defun, 158
- floatexpid, 451
 - calledby PARSE-FloatExponent, 423
 - calls collect, 451
 - calls digitp[5], 451
 - calls identp[5], 451
 - calls maxindex, 451
 - calls pname[5], 451
 - calls spadreduce, 451
 - calls step, 451
 - defun, 451
- fnameMake[5]
 - called by compileSpadLispCmd, 528
- fnameReadable?[5]
 - called by compileSpadLispCmd, 528
- formal2Pattern, 190
 - calledby augmentLisplibModemapsFrom-Functor, 189
 - calls pairList, 190
 - calls sublis, 190
 - local ref \$PatternVariableList, 190
 - defun, 190
- formatUnabbreviated
 - calledby compDefineCapsuleFunction, 280
 - calledby compMacro, 309
 - calledby doIt, 253
- formatUnabbreviatedSig
 - calledby displayMissingFunctions, 193
- fp-output-stream
 - calledby is-console, 467
- freelist, 526
 - calledby compWithMappingModel, 518
 - calledby freelist, 526
 - calls assq[5], 526
 - calls freelist, 526
 - calls getmode, 526
 - calls identp[5], 526
 - calls unionq, 526
 - defun, 526
- function
 - calledby optSpecialCall, 212
- functionp
 - calledby loadLibIfNecessary, 111
- genDeltaEntry
 - calledby coerceByModemap, 345
- genDomainOps, 198
 - calledby genDomainView, 197
 - calls addModemap, 198
 - calls getOperationAlist, 198
 - calls mkDomainConstructor, 198
 - calls mkq, 198
 - calls substNames, 198
 - uses \$ConditionalOperators, 198
 - uses \$e, 198
 - uses \$getDomainCode, 198
 - defun, 198
- genDomainView, 197
 - calledby genDomainViewList, 196
 - calls augModemapsFromCategory, 197
 - calls genDomainOps, 197
 - calls member, 197
 - calls mkDomainConstructor, 197
 - calls qcar, 197
 - calls qcdr, 197
 - uses \$e, 197
 - uses \$getDomainCode, 197
 - defun, 197
- genDomainViewList, 196
 - calledby genDomainViewList, 196
 - calls genDomainViewList, 196
 - calls genDomainView, 196
 - calls isCategoryForm, 196
 - calls qcdr, 196
 - uses \$EmptyEnvironment, 196
 - defun, 196
- genDomainViewList0, 196
 - calledby makeFunctorArgumentParameters, 194
 - calls getDomainViewList, 196
 - defun, 196
- genSomeVariable
 - calledby compColon, 267
 - calledby setqMultiple, 323
- gensymp
 - calledby EqualBarGensym, 224
- genvar
 - calledby hasAplExtension, 389
- genVariable
 - calledby setqMultipleExplicit, 324
 - calledby setqMultiple, 322

- get
 - calledby autoCoerceByModemap, 346
 - calledby coerceHard, 339
 - calledby coerceSubset, 338
 - calledby compAtom, 503
 - calledby compDefineCapsuleFunction, 280
 - calledby compNoStacking1, 498
 - calledby compSymbol, 505
 - calledby compTypeOf, 502
 - calledby compWithMappingModel, 518
 - calledby compileCases, 283
 - calledby compile, 145
 - calledby doIt, 253
 - calledby evalAndSub, 241
 - calledby getArgumentMode, 291
 - calledby getDomainsInScope, 230
 - calledby getFormModemaps, 510
 - calledby getInverseEnvironment, 301
 - calledby getModemapListFromDomain, 236
 - calledby getModemapList, 236
 - calledby getModemap, 234
 - calledby getSignature, 288
 - calledby getSuccessEnvironment, 300
 - calledby giveFormalParametersValues, 135
 - calledby hasFullSignature, 134
 - calledby hasType, 341
 - calledby isFunctor, 229
 - calledby isMacro, 259
 - calledby isSuperDomain, 231
 - calledby isUnionMode, 303
 - calledby maxSuperType, 329
 - calledby mkEvalableCategoryForm, 141
 - calledby optCallSpecially, 211
 - calledby outputComp, 328
 - calledby parseHasRhs, 110
 - calledby setqSingle, 326
- get-a-line, 540
 - calledby initialize-prepare, 73
 - calledby prepareReadLine1, 87
 - calls is-console, 540
 - calls mkprompt[5], 540
 - calls read-a-line, 540
 - defun, 540
- get-internal-run-time
 - calledby s-process, 491
- get-token, 449
 - calledby try-get-token, 447
 - calls XTokenReader, 449
 - uses XTokenReader, 449
 - defun, 449
- getAbbreviation, 277
 - calledby compDefine1, 275
 - calledby encodeFunctionName, 148
 - calls assq, 277
 - calls constructor?, 277
 - calls mkAbbrev, 277
 - calls rplac, 277
 - local def \$abbreviationTable, 277
 - local ref \$abbreviationTable, 277
 - defun, 277
- getArgumentMode, 291
 - calledby checkAndDeclare, 289
 - calledby getArgumentModeOrMoan, 154
 - calledby hasSigInTargetCategory, 290
 - calls get, 291
 - defun, 291
- getArgumentModeOrMoan, 154
 - calledby compDefineCapsuleFunction, 280
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 181
 - calls getArgumentMode, 154
 - calls stackSemanticError, 155
 - defun, 154
- getCaps, 150
 - calledby encodeItem, 150
 - calls l-case, 150
 - calls maxindex, 150
 - calls strconc, 150
 - calls stringimage, 150
 - defun, 150
- getCategoryOpsAndAtts, 177
 - calledby getConstructorOpsAndAtts, 176
 - calls getSlotFromCategoryForm, 177
 - calls transformOperationAlist, 177
 - defun, 177
- getConstructorAbbreviation
 - calledby compDefineLisplib, 169
- getConstructorOpsAndAtts, 176
 - calledby finalizeLisplib, 174
 - calls getCategoryOpsAndAtts, 176
 - calls getFunctorOpsAndAtts, 176
 - defun, 176

- getdatabase
 - calledby augModemapsFromDomain, 232
 - calledby compDefineFunctor1, 182
 - calledby compDefineLisplib, 170
 - calledby compileConstructor1, 153
 - calledby getOperationAlist, 242
 - calledby isFunctor, 229
 - calledby loadLibIfNecessary, 111
 - calledby macroExpandList, 137
 - calledby mkCategoryPackage, 139
 - calledby mkEvaluableCategoryForm, 140
 - calledby parseHas, 108
 - calledby updateCategoryFrameForCategory, 113
 - calledby updateCategoryFrameForConstructor, 112
- getDeltaEntry
 - calledby compElt, 292
- getDomainsInScope, 230
 - calledby addDomain, 228
 - calledby augModemapsFromDomain, 232
 - calledby comp3, 500
 - calledby compColon, 267
 - calledby compConstruct, 272
 - calledby putDomainsInScope, 230
 - calls get, 230
 - local ref \$CapsuleDomainsInScope, 230
 - local ref \$insideCapsuleFunctionIfTrue, 230
 - defun, 230
- getDomainViewList
 - calledby genDomainViewList0, 196
- getFormModemaps, 510
 - calledby compForm1, 508
 - calledby getFormModemaps, 510
 - calls eltModemapFilter, 510
 - calls getFormModemaps, 510
 - calls get, 510
 - calls last, 510
 - calls length, 510
 - calls nequal, 510
 - calls nreverse0, 510
 - calls qcar, 510
 - calls qcdr, 510
 - calls stackMessage, 510
- local ref \$insideCategoryPackageIfTrue, 510
- defun, 510
- getFunctorOpsAndAtts, 179
 - calledby getConstructorOpsAndAtts, 176
 - calls getSlotFromFunctor, 179
 - calls transformOperationAlist, 179
 - defun, 179
- getIdentity
 - calledby compReduce1, 312
- getInverseEnvironment, 301
 - calledby compBoolean, 300
 - calls delete, 302
 - calls getUnionMode, 302
 - calls get, 301
 - calls identp, 301
 - calls isDomainForm, 301
 - calls member, 301
 - calls mkpf, 301
 - calls put, 301
 - calls qcar, 301
 - calls qcdr, 301
 - local ref \$EmptyEnvironment, 302
 - defun, 301
- getl
 - calledby PARSE-Operation, 413
 - calledby PARSE-ReductionOp, 417
 - calledby PARSE-leftBindingPowerOf, 413
 - calledby PARSE-rightBindingPowerOf, 414
 - calledby addConstructorModemaps, 234
 - calledby augModemapsFromDomain1, 232
 - calledby compCat, 261
 - calledby compExpression, 507
 - calledby loadLibIfNecessary, 111
 - calledby mustInstantiate, 266
 - calledby optSpecialCall, 212
 - calledby optimize, 205
 - calledby parseTran, 93
- getmode
 - calledby addDomain, 228
 - calledby augModemapsFromDomain1, 233
 - calledby coerceHard, 339
 - calledby comp3, 500
 - calledby compCoerce, 343
 - calledby compColon, 267
 - calledby compDefWhereClause, 200

- calledby compDefineCapsuleFunction, 280
- calledby compSymbol, 505
- calledby compile, 145
- calledby displayMissingFunctions, 193
- calledby freelist, 526
- calledby getSignatureFromMode, 278
- calledby getSignature, 288
- calledby getUnionMode, 303
- calledby isUnionMode, 303
- calledby setqSingle, 326
- getModemap, 234
 - calledby compDefineFunctor1, 181
 - calls compApplyModemap, 234
 - calls get, 234
 - calls sublis, 234
 - defun, 234
- getModemapList, 235
 - calledby autoCoerceByModemap, 345
 - calledby compCase1, 260
 - calledby getUniqueModemap, 235
 - calls getModemapListFromDomain, 236
 - calls get, 236
 - calls nreverse0, 236
 - calls qcar, 235
 - calls qcdr, 236
 - defun, 235
- getModemapListFromDomain, 236
 - calledby compElt, 292
 - calledby getModemapList, 236
 - calls get, 236
 - defun, 236
- getmodeOrMapping
 - calledby augModemapsFromDomain1, 233
- getOperationAlist, 242
 - calledby evalAndSub, 241
 - calledby genDomainOps, 198
 - calls compMakeCategoryObject, 242
 - calls getdatabase, 242
 - calls isFunctor, 242
 - calls stackMessage, 242
 - calls systemError, 242
 - uses \$domainShell, 242
 - uses \$e, 242
 - uses \$functorForm, 242
 - uses \$insideFunctorIfTrue, 242
 - defun, 242
- getOperationAlistFromLisplib
 - calledby mkOpVec, 199
- getParentsFor
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
- getPrincipalView
 - calledby mkOpVec, 199
- getProplist
 - calledby addModemap1, 246
 - calledby compDefineAddSignature, 133
 - calledby getSuccessEnvironment, 300
 - calledby loadLibIfNecessary, 111
- getProplist[5]
 - called by setqSingle, 326
- getScriptName, 391
 - calledby postScriptsForm, 354
 - calledby postScripts, 381
 - calls decodeScripts, 391
 - calls identp[5], 391
 - calls internl, 391
 - calls pname[5], 391
 - calls postError, 391
 - defun, 391
- getSignature, 288
 - calledby compDefineCapsuleFunction, 280
 - calls SourceLevelSubsume, 288
 - calls getmode, 288
 - calls get, 288
 - calls knownInfo, 288
 - calls length, 288
 - calls printSignature, 288
 - calls qcar, 288
 - calls qcdr, 288
 - calls remdup, 288
 - calls say, 288
 - calls stackSemanticError, 288
 - local ref \$e, 288
 - defun, 288
- getSignatureFromMode, 278
 - calledby compDefine1, 275
 - calledby hasSigInTargetCategory, 290
 - calls eqsubstlist, 279
 - calls getmode, 278
 - calls length, 278
 - calls nequal, 278
 - calls opOf, 278

- calls qcar, 278
- calls qcdr, 278
- calls stackAndThrow, 279
- calls take, 279
- local ref \$FormalMapVariableList, 279
- defun, 278
- getSlotFromCategoryForm, 177
 - calledby getCategoryOpsAndAtts, 177
 - calls eval, 177
 - calls systemErrorHere, 177
 - calls take, 177
 - local ref \$FormalMapVariableList, 177
 - defun, 177
- getSlotFromFunctor, 179
 - calledby getFunctorOpsAndAtts, 179
 - calls compMakeCategoryObject, 179
 - calls systemErrorHere, 179
 - local ref \$e, 179
 - local ref \$lisplibOperationAlist, 179
 - defun, 179
- getSpecialCaseAssoc, 285
 - calledby compileCases, 283
 - local ref \$functorForm, 285
 - local ref \$functorSpecialCases, 285
 - defun, 285
- getSuccessEnvironment, 300
 - calledby compBoolean, 300
 - calledby doItIf, 257
 - calls addBinding, 300
 - calls comp, 300
 - calls consProplistOf, 300
 - calls getProplist, 300
 - calls get, 300
 - calls identp, 300
 - calls isDomainForm, 300
 - calls put, 300
 - calls qcar, 300
 - calls qcdr, 300
 - calls removeEnv, 300
 - local ref \$EmptyEnvironment, 300
 - local ref \$EmptyMode, 300
 - defun, 300
- getTargetFromRhs, 135
 - calledby compDefine1, 275
 - calledby getTargetFromRhs, 135
 - calls compOrCroak, 135
 - calls getTargetFromRhs, 135
 - calls stackSemanticError, 135
 - defun, 135
- getToken, 444
 - calledby PARSE-OpenBrace, 432
 - calledby PARSE-OpenBracket, 432
 - calls eqcar, 444
 - defun, 444
- getUnionMode, 303
 - calledby getInverseEnvironment, 302
 - calls getmode, 303
 - calls isUnionMode, 303
 - defun, 303
- getUniqueModemap, 235
 - calledby getUniqueSignature, 235
 - calls getModemapList, 235
 - calls qslessp, 235
 - calls stackWarning, 235
 - defun, 235
- getUniqueSignature, 235
 - calls getUniqueModemap, 235
 - defun, 235
- giveFormalParametersValues, 135
 - calledby compDefine1, 275
 - calledby compDefineCapsuleFunction, 280
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 181
 - calls get, 135
 - calls put, 135
 - defun, 135
- hackforis, 394
 - calls hackforis1, 394
 - defun, 394
- hackforis1, 395
 - calledby hackforis, 394
 - calls eqcar, 395
 - calls kar, 395
 - defun, 395
- has, 108, 294
 - defplist, 108, 294
- hasAplExtension, 389
 - calledby aplTran1, 387
 - calls aplTran1, 389
 - calls deepestExpression, 389
 - calls genvar, 389

- calls msubst, 389
- calls nreverse0, 389
- defun, 389
- hasFormalMapVariable, 524
 - calledby compWithMappingModel, 518
 - calls ScanOrPairVec[5], 524
 - local def \$formalMapVariables, 524
 - defun, 524
- hasFullSignature, 134
 - calledby compDefineAddSignature, 133
 - calls get, 134
 - defun, 134
- hasSigInTargetCategory, 290
 - calledby compDefineCapsuleFunction, 280
 - calls bright, 290
 - calls compareMode2Arg, 290
 - calls getArgumentMode, 290
 - calls getSignatureFromMode, 290
 - calls length, 290
 - calls remdup, 290
 - calls stackWarning, 290
 - local ref \$domainShell, 290
 - defun, 290
- hasType, 341
 - calledby coerceExtraHard, 340
 - calls get, 341
 - defun, 341
- helpSpad2Cmd[5]
 - called by compiler, 473
- identp
 - calledby addDomain, 228
 - calledby compInternalFunction, 279
 - calledby constructMacro, 151
 - calledby encodeItem, 150
 - calledby getInverseEnvironment, 301
 - calledby getSuccessEnvironment, 300
 - calledby isFunctor, 229
 - calledby mkExplicitCategoryFunction, 265
 - calledby subname, 208
- identp[5]
 - called by PARSE-FloatExponent, 423
 - called by compSetq1, 321
 - called by floatexpid, 451
 - called by freelist, 526
 - called by getScriptName, 391
- called by postTransform, 351
- called by setqSingle, 326
- if, 113, 296, 373
 - defplist, 113, 296, 373
- ifcar
 - calledby preparsed, 76
- implies, 116
 - defplist, 116
- import, 303
 - defplist, 303
- In, 375
 - defplist, 375
- in, 117, 374
 - defplist, 117, 374
- inby, 118
 - defplist, 118
- incExitLevel
 - calledby parseIf,ifTran, 114
- indent-pos, 466
 - calledby preparsed1, 81
 - defun, 466
- infixtok, 467
 - calledby add-parens-and-semis-to-line, 84
 - calls string2id-n, 467
 - defun, 467
- init-boot/spad-reader[5]
 - called by spad, 489
- initial-gensym, 394
 - defvar, 394
- initial-substring, 539
 - calledby preparsedReadLine, 85
 - calledby skip-ifblock, 86
 - calledby skip-to-endif, 468
 - calls mismatch, 539
 - defun, 539
- initial-substring-p, 442
 - calledby match-string, 440
 - calls string-not-greaterp, 442
 - defun, 442
- initialize-preparse, 73
 - calledby spad, 489
 - calls get-a-line, 73
 - uses \$echolinestack, 73
 - uses \$index, 73
 - uses \$linelist, 73
 - uses \$preparse-last-line, 73

- defun, 73
- initializeLisplib, 173
 - calls LAM,FILEACTQ, 173
 - calls addoptions, 173
 - calls erase, 173
 - calls pathnameTypeId, 173
 - calls writeLib1, 173
 - local def \$libFile, 173
 - local def \$lisplibAbbreviation, 173
 - local def \$lisplibAncestors, 173
 - local def \$lisplibForm, 173
 - local def \$lisplibKind, 173
 - local def \$lisplibModemapAlist, 173
 - local def \$lisplibModemap, 173
 - local def \$lisplibOpAlist, 173
 - local def \$lisplibOperationAlist, 173
 - local def \$lisplibSignatureAlist, 173
 - local def \$lisplibSuperDomain, 173
 - local def \$lisplibVariableAlist, 173
 - local ref \$erase, 173
 - local ref \$libFile, 173
 - uses /editfile, 173
 - uses /major-version, 173
 - uses errors, 173
 - defun, 173
- insert
 - calledby comp2, 499
 - calledby doIt, 253
- insertAlist
 - calledby transformOperationAlist, 178
- insertModemap, 239
 - calledby mkNewModemapList, 238
 - defun, 239
- insertWOC
 - calledby orderPredTran, 162
- Integer
 - calledby optCallEval, 213
- interactiveModemapForm, 158
 - calledby augLisplibModemapsFromCategory, 155
 - calledby augmentLisplibModemapsFromFunctor, 189
 - calls fixUpPredicate, 158
 - calls modemapPattern, 158
 - calls nequal, 158
 - calls qcar, 158
 - calls qcdr, 158
 - calls replaceVars, 158
 - calls substVars, 158
 - local ref \$FormalMapVariableList, 158
 - local ref \$PatternVariableList, 158
 - defun, 158
- internl
 - calledby encodeFunctionName, 148
 - calledby getScriptName, 391
 - calledby postForm, 356
 - calledby substituteCategoryArguments, 233
- intersection
 - calledby orderByDependency, 202
- intersectionEnvironment
 - calledby compIf, 296
 - calledby replaceExitEtc, 319
- intersectionq
 - calledby orderPredTran, 162
- ioclear
 - calledby spad, 489
- is, 119, 304
 - defplist, 119, 304
- is-console, 467
 - calledby get-a-line, 540
 - calledby preparse1, 81
 - calledby print-defun, 493
 - calls fp-output-stream, 467
 - uses *terminal-io*, 467
 - defun, 467
- isAlmostSimple
 - calledby makeSimplePredicateOrNil, 458
- isCategory
 - calledby augModemapsFromCategoryRep, 243
 - calledby evalAndSub, 241
- isCategoryForm
 - calledby addDomain, 228
 - calledby augLisplibModemapsFromCategory, 155
 - calledby coerceHard, 339
 - calledby compColon, 267
 - calledby compJoin, 305
 - calledby compMakeCategoryObject, 180
 - calledby genDomainViewList, 196
 - calledby isDomainConstructorForm, 330
 - calledby isDomainForm, 329

- calledby makeCategoryForm, 270
- calledby makeFunctorArgumentParameters, 194
- calledby mkAlistOfExplicitCategoryOps, 156
- calledby mkDatabasePred, 191
- calledby signatureTran, 161
- isCategoryPackageName
 - calledby compDefineFunctor1, 181, 182
 - calledby substNames, 243
- isConstantId
 - calledby eltModemapFilter, 511
 - calledby seteltModemapFilter, 512
- isDomainConstructorForm, 330
 - calledby isDomainForm, 329
 - calls eqsubstlist, 330
 - calls isCategoryForm, 330
 - calls qcar, 330
 - calls qcdr, 330
 - local ref \$FormalMapVariableList, 330
 - defun, 330
- isDomainForm, 329
 - calledby comp2, 499
 - calledby compColon, 267
 - calledby compDefine1, 275
 - calledby compElt, 292
 - calledby compHasFormat, 295
 - calledby doIt, 253
 - calledby getInverseEnvironment, 301
 - calledby getSuccessEnvironment, 300
 - calledby setqSingle, 326
 - calls isCategoryForm, 329
 - calls isDomainConstructorForm, 329
 - calls isFunctor, 329
 - calls kar, 329
 - calls qcar, 329
 - calls qcdr, 329
 - local ref \$SpecialDomainNames, 329
 - defun, 329
- isDomainInScope
 - calledby setqSingle, 326
- isDomainSubst, 164
 - calledby orderPredTran, 162
 - defun, 164
- isFunction
 - calledby compSymbol, 505
- isFunctor, 229
 - calledby addDomain, 228
 - calledby comp2, 499
 - calledby compWithMappingMode1, 518
 - calledby getOperationAlist, 242
 - calledby isDomainForm, 329
 - calls constructor?, 229
 - calls getdatabase, 229
 - calls get, 229
 - calls identp, 229
 - calls opOf, 229
 - calls updateCategoryFrameForCategory, 229
 - calls updateCategoryFrameForConstructor, 229
 - local ref \$CategoryFrame, 229
 - local ref \$InteractiveMode, 229
 - defun, 229
- isListConstructor, 103
 - calledby transIs, 102
 - calls member, 103
 - defun, 103
- isLiteral
 - calledby addDomain, 228
- isMacro, 259
 - calledby compDefine1, 275
 - calledby doIt, 253
 - calls get, 259
 - calls qcar, 259
 - calls qcdr, 259
 - defun, 259
- isnt, 119
 - defplist, 119
- isSimple
 - calledby compForm2, 513
 - calledby makeSimplePredicateOrNil, 458
- isSomeDomainVariable
 - calledby coerce, 337
- isSubset
 - calledby coerceByModemap, 345
 - calledby coerceSubset, 338
 - calledby isSuperDomain, 231
- isSuperDomain, 231
 - calledby mergeModemap, 239
 - calls get, 231
 - calls isSubset, 231

- calls lassoc, 231
- calls opOf, 231
- defun, 231
- isSymbol
 - calledby compAtom, 503
- isTokenDelimiter, 443
 - calledby PARSE-TokenList, 406
 - calls current-symbol, 443
 - defun, 443
- isUnionMode, 303
 - calledby coerceExtraHard, 340
 - calledby getUnionMode, 303
 - calls getmode, 303
 - calls get, 303
 - defun, 303
- Join, 120, 305, 375
 - defplist, 120, 305, 375
- JoinInner
 - calledby mkCategoryPackage, 139
- kar
 - calledby addEmptyCapsuleIfNecessary, 134
 - calledby augModemapsFromDomain1, 232
 - calledby augModemapsFromDomain, 232
 - calledby compile, 145
 - calledby doIt, 253
 - calledby hackforis1, 395
 - calledby isDomainForm, 329
 - calledby optCallSpecially, 211
- keyedSystemError
 - calledby coerce, 337
 - calledby compileTimeBindingOf, 213
 - calledby mkAlistOfExplicitCategoryOps, 156
 - calledby optRECORDELT, 225
 - calledby optSETRECORDELT, 226
 - calledby optSpecialCall, 212
 - calledby transformOperationAlist, 178
- killColons, 383
 - calledby killColons, 383
 - calledby postSignature, 382
 - calls killColons, 383
 - defun, 383
- knownInfo
 - calledby addModemap, 245
 - calledby getSignature, 288
- l-case
 - calledby getCaps, 150
- labasoc
 - usedby PARSE-Data, 428
- lablasoc, 403
 - defvar, 403
- LAM,EVALANDFILEACTQ
 - calledby spadCompileOrSetq, 152
- LAM,FILEACTQ
 - calledby initializeLisplib, 173
- lassoc
 - calledby addModemap1, 246
 - calledby augLisplibModemapsFromCategory, 155
 - calledby coerceSubset, 338
 - calledby compDefWhereClause, 200
 - calledby compDefineAddSignature, 133
 - calledby compOrCroak1,compactify, 527
 - calledby isSuperDomain, 231
 - calledby loadLibIfNecessary, 111
 - calledby mergeSignatureAndLocalVarAlists, 180
 - calledby optCallSpecially, 211
 - calledby translabe11, 455
- lassq
 - calledby transformOperationAlist, 178
- last
 - calledby getFormModemaps, 510
 - calledby parseSeq, 128
 - calledby setqMultipleExplicit, 324
- lastnode
 - calledby doIt, 253
- leave, 121, 308
 - defplist, 121, 308
- length
 - calledby compColon, 267
 - calledby compDefine1, 275
 - calledby compDefineCapsuleFunction, 280
 - calledby compElt, 292
 - calledby compForm1, 508
 - calledby compForm2, 513
 - calledby compHasFormat, 295
 - calledby compRepeatOrCollect, 314
 - calledby displayPreCompilationErrors, 456

- calledby encodeFunctionName, 148
- calledby fixUpPredicate, 160
- calledby getFormModemaps, 510
- calledby getSignatureFromMode, 278
- calledby getSignature, 288
- calledby hasSigInTargetCategory, 290
- calledby mkOpVec, 199
- calledby modeEqualSubst, 348
- calledby optMkRecord, 225
- calledby postScriptsForm, 354
- calledby setqMultiple, 323
- lessp, 218
 - defplist, 218
- let, 122, 320
 - defplist, 122, 320
- letd, 123
 - defplist, 123
- line, 535
 - usedby match-string, 440
 - usedby spad, 489
 - defstruct, 535
- Line-Advance-Char
 - calledby Advance-Char, 539
- line-advance-char, 537
 - uses \$line, 537
 - defun, 537
- Line-At-End-P
 - calledby Advance-Char, 539
- line-at-end-p, 536
 - calledby next-char, 450
 - uses \$line, 536
 - defun, 536
- line-clear, 536
 - uses \$line, 536
 - defmacro, 536
- line-current-char
 - calledby match-advance-string, 441
- line-current-index
 - calledby match-advance-string, 441
- line-current-segment, 538
 - calledby unget-tokens, 444
 - defun, 538
- Line-New-Line
 - calledby read-a-line, 533
- line-new-line, 538
 - calledby unget-tokens, 445
 - uses \$line, 538
 - defun, 538
- line-next-char, 537
 - calledby next-char, 450
 - uses \$line, 537
 - defun, 537
- line-number
 - calledby PARSE-Category, 410
 - calledby unget-tokens, 445
- line-past-end-p, 537
 - calledby match-advance-string, 441
 - calledby match-string, 440
 - uses \$line, 537
 - defun, 537
- line-print, 536
 - local ref \$out-stream, 536
 - uses \$line, 536, 538
 - defun, 536
- lispize, 333
 - calledby compSubDomain1, 332
 - calls optimize, 333
 - defun, 333
- lisplibDoRename, 172
 - calledby compDefineLisplib, 169
 - calls replaceFile, 172
 - local ref \$spadLibFT, 172
 - defun, 172
- lisplibWrite, 180
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby compileDocumentation, 172
 - calledby finalizeLisplib, 174
 - calls rwrite128, 180
 - local ref \$lisplib, 181
 - defun, 180
- List
 - calledby optCallEval, 213
- ListCategory, 273
 - defplist, 273
- listOfIdentifiersIn
 - calledby compDefWhereClause, 200
- listOfPatternIds
 - calledby augmentLisplibModemapsFrom-Functor, 189
 - calledby orderPredTran, 162
- listOrVectorElementNode

- calledby augModemapsFromDomain, 232
- loadIfNecessary, 111
 - calledby parseHasRhs, 110
 - calls loadLibIfNecessary, 111
 - defun, 111
- loadLib
 - calledby loadLibIfNecessary, 111
- loadLibIfNecessary, 111
 - calledby loadIfNecessary, 111
 - calledby loadLibIfNecessary, 111
 - calls functionp, 111
 - calls getPropList, 111
 - calls getdatabase, 111
 - calls getl, 111
 - calls lassoc, 111
 - calls loadLibIfNecessary, 111
 - calls loadLib, 111
 - calls macrop, 111
 - calls throwKeyedMsg, 111
 - calls updateCategoryFrameForCategory, 111
 - calls updateCategoryFrameForConstructor, 111
 - local ref \$CategoryFrame, 111
 - local ref \$InteractiveMode, 111
 - defun, 111
- localdatabase
 - calledby compDefineLisplib, 170
- localdatabase[5]
 - called by compileSpadLispCmd, 528
- localExtras
 - calledby doItIf, 257
- lt
 - calledby PARSE-Operation, 413
- macroExpand, 136
 - calledby compDefine1, 275
 - calledby compMacro, 309
 - calledby compSeqItem, 320
 - calledby compWhere, 336
 - calledby macroExpandInPlace, 136
 - calledby macroExpandList, 137
 - calledby macroExpand, 136
 - calls macroExpandList, 136
 - calls macroExpand, 136
 - defun, 136
- macroExpandInPlace, 136
 - calledby compSingleCapsuleItem, 252
 - calls macroExpand, 136
 - defun, 136
- macroExpandList, 137
 - calledby macroExpand, 136
 - calls getdatabase, 137
 - calls macroExpand, 137
 - defun, 137
- macrop
 - calledby loadLibIfNecessary, 111
- make-float
 - calledby PARSE-Float, 421
- make-full-cvec
 - calledby preparse1, 81
- make-input-filename
 - calledby compileDocumentation, 172
- make-reduction
 - calledby push-reduction, 454
- make-symbol-of, 446
 - calledby PARSE-Expression, 411
 - calledby current-symbol, 446
 - uses \$token, 446
 - defun, 446
- makeCategoryForm, 270
 - calledby compColon, 267
 - calls compOrCroak, 270
 - calls isCategoryForm, 270
 - local ref \$EmptyMode, 270
 - defun, 270
- makeCategoryPredicates, 138
 - calledby compDefineCategory1, 137
 - uses \$FormalMapVariableList, 138
 - uses \$TriangleVariableList, 138
 - uses \$mvl, 138
 - uses \$tv1, 138
 - defun, 138
- makeFunctorArgumentParameters, 194
 - calledby compDefineFunctor1, 182
 - calls assq, 194
 - calls genDomainViewList0, 194
 - calls isCategoryForm, 194
 - calls msubst, 194
 - calls qcar, 194
 - calls qcdr, 194
 - calls union, 194

- uses \$ConditionalOperators, 194
 - uses \$alternateViewList, 194
 - uses \$forceAdd, 194
 - defun, 194
- makeInitialModemapFrame[5]
 - called by spad, 489
- makeInputFilename[5]
 - called by /rf-1, 480
- makeLiteral
 - calledby addEltModemap, 237
- makeNonAtomic
 - calledby parseHas, 109
- makeprop
 - calledby finalizeLisplib, 174
- makeSimplePredicateOrNil, 458
 - calledby parseIf.ifTran, 114
 - calls isAlmostSimple, 458
 - calls isSimple, 458
 - calls wrapSEQExit, 458
 - defun, 458
- mapInto
 - calledby parseSeq, 128
 - calledby parseWhere, 129
- Mapping, 261
 - defplist, 261
- match-advance-string, 441
 - calledby PARSE-Category, 409
 - calledby PARSE-Command, 404
 - calledby PARSE-Conditional, 437
 - calledby PARSE-Enclosure, 424
 - calledby PARSE-Exit, 435
 - calledby PARSE-FloatBasePart, 422
 - calledby PARSE-FloatExponent, 423
 - calledby PARSE-Form, 417
 - calledby PARSE-Import, 411
 - calledby PARSE-IteratorTail, 433
 - calledby PARSE-Iterator, 433
 - calledby PARSE-Label, 419
 - calledby PARSE-Leave, 436
 - calledby PARSE-Loop, 438
 - calledby PARSE-Option, 408
 - calledby PARSE-Primary1, 421
 - calledby PARSE-PrimaryOrQM, 407
 - calledby PARSE-Quad, 425
 - calledby PARSE-Qualification, 416
 - calledby PARSE-Return, 435
 - calledby PARSE-ScriptItem, 427
 - calledby PARSE-Scripts, 426
 - calledby PARSE-Selector, 419
 - calledby PARSE-SemiColon, 435
 - calledby PARSE-Sequence, 431
 - calledby PARSE-Sexpr1, 428
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-Statement, 408
 - calledby PARSE-TokenOption, 406
 - calledby PARSE-With, 409
 - calls current-token, 441
 - calls line-current-char, 441
 - calls line-current-index, 441
 - calls line-past-end-p, 441
 - calls match-string, 441
 - calls quote-if-string, 441
 - uses \$line, 441
 - uses \$token, 441
 - defun, 441
- match-current-token, 445
 - calledby PARSE-GlyphTok, 430
 - calledby PARSE-NBGlyphTok, 429
 - calledby PARSE-Operation, 413
 - calledby PARSE-SpecialKeyword, 404
 - calledby parse-argument-designator, 460
 - calledby parse-identifier, 459
 - calledby parse-keyword, 460
 - calledby parse-number, 460
 - calledby parse-spadstring, 458
 - calledby parse-string, 459
 - calls current-token, 445
 - calls match-token, 445
 - defun, 445
- match-next-token, 446
 - calledby PARSE-ReductionOp, 417
 - calls match-token, 446
 - calls next-token, 446
 - defun, 446
- match-string, 440
 - calledby PARSE-AnyId, 430
 - calledby PARSE-NewExpr, 403
 - calledby PARSE-Primary1, 421
 - calledby match-advance-string, 441
 - calls current-char, 440
 - calls initial-substring-p, 440
 - calls line-past-end-p, 440

- calls skip-blanks, 440
- calls subseq, 440
- calls unget-tokens, 440
- uses \$line, 440
- uses line, 440
- defun, 440
- match-token, 446
 - calledby match-current-token, 445
 - calledby match-next-token, 446
 - calls token-symbol, 446
 - calls token-type, 446
 - defun, 446
- Matrix
 - calledby optCallEval, 213
- maxindex
 - calledby compDefineFunctor1, 182
 - calledby floatexpid, 451
 - calledby getCaps, 150
 - calledby preparse1, 81
 - calledby preparseReadLine1, 87
 - calledby translable1, 455
- maxSuperType, 329
 - calledby coerceSubset, 338
 - calledby maxSuperType, 329
 - calledby setqSingle, 326
 - calls get, 329
 - calls maxSuperType, 329
 - defun, 329
- mbpip
 - calledby subname, 208
- mdef, 123, 309
 - defplist, 123, 309
- member
 - calledby addDomain, 228
 - calledby augLisplibModemapsFromCategory, 155
 - calledby augModemapsFromDomain, 232
 - calledby augmentLisplibModemapsFromFunctor, 189
 - calledby autoCoerceByModemap, 345
 - calledby coerceExtraHard, 340
 - calledby compDefineCapsuleFunction, 280
 - calledby compile, 145
 - calledby displayMissingFunctions, 193
 - calledby doIt, 253
 - calledby genDomainView, 197
 - calledby getInverseEnvironment, 301
 - calledby isListConstructor, 103
 - calledby mkNewModemapList, 238
 - calledby orderByDependency, 202
 - calledby orderPredTran, 162
 - calledby parseHasRhs, 110
 - calledby parseHas, 109
 - calledby putDomainsInScope, 231
 - calledby transformOperationAlist, 178
- member[5]
 - called by comp3, 500
 - called by compColon, 267
 - called by compSymbol, 505
 - called by compilerDoit, 478
- mergeModemap, 239
 - calledby mkNewModemapList, 238
 - calls TruthP, 239
 - calls isSuperDomain, 239
 - local ref \$forceAdd, 239
 - defun, 239
- mergePathnames[5]
 - called by compiler, 473
- mergeSignatureAndLocalVarAlists, 180
 - calledby finalizeLisplib, 174
 - calls lassoc, 180
 - defun, 180
- meta-error-handler, 450
 - calledby meta-syntax-error, 451
 - usedby meta-syntax-error, 451
 - defvar, 450
- meta-syntax-error, 451
 - calledby must, 452
 - calls meta-error-handler, 451
 - uses meta-error-handler, 451
 - defun, 451
- minus, 216
 - defplist, 216
- mismatch
 - calledby initial-substring, 539
- mkAbbrev, 277
 - calledby getAbbreviation, 277
 - calls addSuffix, 277
 - calls alistSize, 277
 - defun, 277
- mkAlistOfExplicitCategoryOps, 156

- calledby augLisplibModemapsFromCategory, 155
- calledby augmentLisplibModemapsFromFunctor, 189
- calledby mkAlistOfExplicitCategoryOps, 156
- calls assocleft, 156
- calls flattenSignatureList, 156
- calls isCategoryForm, 156
- calls keyedSystemError, 156
- calls mkAlistOfExplicitCategoryOps, 156
- calls nreverse0, 156
- calls qcar, 156
- calls qcdr, 156
- calls remdup, 156
- calls union, 156
- local ref \$e, 156
- defun, 156
- mkCategoryPackage, 139
 - calledby compDefineCategory1, 137
 - calls JoinInner, 139
 - calls abbreviationsSpad2Cmd, 139
 - calls assoc, 139
 - calls getdatabase, 139
 - calls msubst, 139
 - calls pname, 139
 - calls strconc, 139
 - calls sublis, 139
 - uses \$FormalMapVariableList, 139
 - uses \$categoryPredicateList, 139
 - uses \$e, 139
 - uses \$options, 139
 - defun, 139
- mkConstructor, 168
 - calledby compDefineCategory2, 142
 - calledby mkConstructor, 168
 - calls mkConstructor, 168
 - defun, 168
- mkDatabasePred, 191
 - calledby augmentLisplibModemapsFromFunctor, 189
 - calls isCategoryForm, 191
 - local ref \$e, 191
 - defun, 191
- mkDomainConstructor
 - calledby bootStrapError, 192
 - calledby compHasFormat, 295
 - calledby genDomainOps, 198
 - calledby genDomainView, 197
- mkErrorExpr
 - calledby compOrCroak1, 497
- mkEvalableCategoryForm, 140
 - calledby compMakeCategoryObject, 180
 - calledby mkEvalableCategoryForm, 140
 - calls compOrCroak, 140
 - calls getdatabase, 140
 - calls get, 141
 - calls mkEvalableCategoryForm, 140
 - calls mkq, 141
 - calls qcar, 140
 - calls qcdr, 140
 - local def \$e, 141
 - local ref \$CategoryFrame, 141
 - local ref \$CategoryNames, 141
 - local ref \$Category, 141
 - local ref \$EmptyMode, 141
 - local ref \$e, 141
 - defun, 140
- mkExplicitCategoryFunction, 265
 - calledby compCategory, 262
 - calls identp, 265
 - calls mkq, 265
 - calls mustInstantiate, 265
 - calls nequal, 265
 - calls remdup, 265
 - calls union, 265
 - calls wrapDomainSub, 265
 - defun, 265
- mkList, 296
 - calledby compHasFormat, 295
 - defun, 296
- mkNewModemapList, 238
 - calledby addModemap1, 246
 - calls assoc, 238
 - calls insertModemap, 238
 - calls member, 238
 - calls mergeModemap, 238
 - calls nequal, 238
 - calls nreverse0, 238
 - calls qcar, 238
 - calls qcdr, 238
 - local ref \$InteractiveMode, 238

- local ref \$forceAdd, 238
 - defun, 238
- mkOpVec, 199
 - calls AssocBarGensym, 199
 - calls assoc, 199
 - calls assq, 199
 - calls getOperationAlistFromLisplib, 199
 - calls getPrincipalView, 199
 - calls length, 199
 - calls msubst, 199
 - calls opOf, 199
 - calls qcar, 199
 - calls qcdr, 199
 - calls sublis, 199
 - uses Undef, 199
 - uses \$FormalMapVariableList, 199
 - defun, 199
- mkpf
 - calledby augLisplibModemapsFromCategory, 155
 - calledby augmentLisplibModemapsFromFunctor, 189
 - calledby compCapsuleInner, 251
 - calledby compCategoryItem, 263
 - calledby compileCases, 283
 - calledby getInverseEnvironment, 301
 - calledby stripOffSubdomainConditions, 287
- mkprogn
 - calledby setqMultiple, 323
- mkprompt[5]
 - called by get-a-line, 540
- mkq
 - calledby addArgumentConditions, 285
 - calledby bootStrapError, 192
 - calledby compCoerce1, 344
 - calledby compDefineCapsuleFunction, 280
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby compSeq1, 318
 - calledby genDomainOps, 198
 - calledby mkEvalableCategoryForm, 141
 - calledby mkExplicitCategoryFunction, 265
 - calledby optSpecialCall, 212
 - calledby spadCompileOrSetq, 152
- mkRecord, 225
 - defplist, 225
- mkRepetitionAssoc, 149
 - calledby encodeFunctionName, 148
 - calls qcar, 149
 - calls qcdr, 149
 - defun, 149
- mkUnion, 347
 - calledby resolve, 347
 - calls qcar, 347
 - calls qcdr, 347
 - calls union, 347
 - local ref \$Rep, 347
 - defun, 347
- moan
 - calledby compileTimeBindingOf, 213
 - calledby parseExit, 107
 - calledby parseReturn, 127
- modeEqual, 348
 - calledby autoCoerceByModemap, 345
 - calledby checkAndDeclare, 289
 - calledby coerceByModemap, 345
 - calledby coerceHard, 339
 - calledby compCase1, 260
 - calledby compile, 145
 - calledby domainMember, 236
 - calledby modeEqualSubst, 348
 - calledby resolve, 347
 - defun, 348
- modeEqualSubst, 348
 - calledby coerceEasy, 338
 - calledby modeEqualSubst, 348
 - calls length, 348
 - calls modeEqualSubst, 348
 - calls modeEqual, 348
 - defun, 348
- modeIsAggregateOf
 - calledby compAtom, 503
 - calledby compConstruct, 272
 - calledby compRepeatOrCollect, 315
- modemapPattern, 167
 - calledby interactiveModemapForm, 158
 - calls qcar, 167
 - calls qcdr, 167
 - calls rassoc, 167
 - local ref \$PatternVariableList, 167
 - defun, 167
- modifyModeStack, 525

- calledby compExit, 293
- calledby compLeave, 308
- calledby compReturn, 317
- calls copy, 525
- calls resolve, 525
- calls say, 525
- calls setelt, 525
- uses \$exitModeStack, 525
- uses \$reportExitModeStack, 525
- defun, 525
- moveORsOutside, 165
 - calledby fixUpPredicate, 160
 - calledby moveORsOutside, 165
 - calls moveORsOutside, 165
 - defun, 165
- msubst
 - calledby addConstructorModemaps, 234
 - calledby addModemap1, 246
 - calledby augModemapsFromCategoryRep, 243
 - calledby augmentLisplibModemapsFrom-Functor, 189
 - calledby coerceSubset, 338
 - calledby coerce, 337
 - calledby compArgumentConditions, 286
 - calledby compCoerce1, 344
 - calledby compReduce1, 312
 - calledby compRepeatOrCollect, 315
 - calledby compSubsetCategory, 333
 - calledby compileCases, 283
 - calledby encodeFunctionName, 148
 - calledby hasAplExtension, 389
 - calledby makeFunctorArgumentParameters, 194
 - calledby mkCategoryPackage, 139
 - calledby mkOpVec, 199
 - calledby parseDollarGreaterEqual, 104
 - calledby parseDollarGreaterThan, 104
 - calledby parseDollarLessEqual, 105
 - calledby parseDollarNotEqual, 106
 - calledby parseNotEqual, 125
 - calledby parseTransform, 93
 - calledby parseType, 98
 - calledby replaceVars, 159
 - calledby stripOffArgumentConditions, 287
 - calledby substVars, 166
 - calledby substituteCategoryArguments, 233
- must, 452
 - calledby PARSE-Category, 409
 - calledby PARSE-Command, 404
 - calledby PARSE-Conditional, 437
 - calledby PARSE-Enclosure, 424
 - calledby PARSE-Exit, 435
 - calledby PARSE-FloatBasePart, 423
 - calledby PARSE-FloatBase, 422
 - calledby PARSE-FloatExponent, 423
 - calledby PARSE-Float, 421
 - calledby PARSE-Form, 417
 - calledby PARSE-Import, 411
 - calledby PARSE-Infix, 415
 - calledby PARSE-Iterator, 433
 - calledby PARSE-LabelExpr, 438
 - calledby PARSE-Label, 419
 - calledby PARSE-Leave, 436
 - calledby PARSE-Loop, 438
 - calledby PARSE-NewExpr, 403
 - calledby PARSE-Option, 408
 - calledby PARSE-Prefix, 415
 - calledby PARSE-Primary1, 420
 - calledby PARSE-Qualification, 416
 - calledby PARSE-Reduction, 417
 - calledby PARSE-Return, 435
 - calledby PARSE-ScriptItem, 427
 - calledby PARSE-Scripts, 426
 - calledby PARSE-Selector, 419
 - calledby PARSE-SemiColon, 435
 - calledby PARSE-Sequence, 431
 - calledby PARSE-Sexpr1, 428
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-Statement, 408
 - calledby PARSE-TokenOption, 406
 - calledby PARSE-With, 409
 - calls meta-syntax-error, 452
 - defmacro, 452
- mustInstantiate, 266
 - calledby mkExplicitCategoryFunction, 265
 - calls get1, 266
 - calls qcar, 266
 - local ref \$DummyFunctorNames, 266
 - defun, 266
- namestring

- calledby bootStrapError, 192
- calledby finalizeLisplib, 174
- namestring[5]
 - called by compileSpad2Cmd, 476
 - called by compileSpadLispCmd, 528
 - called by compiler, 473
- ncINTERPFILE, 527
 - calledby /rf-1, 480
 - calls SpadInterpretStream[5], 527
 - uses \$EchoLines, 527
 - uses \$ReadingFile, 527
 - defun, 527
- nequal
 - calledby comp2, 499
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby compElt, 292
 - calledby compPretend, 310
 - calledby compReturn, 317
 - calledby compileSpad2Cmd, 476
 - calledby compile, 145
 - calledby displayPreCompilationErrors, 456
 - calledby getFormModemaps, 510
 - calledby getSignatureFromMode, 278
 - calledby interactiveModemapForm, 158
 - calledby mkExplicitCategoryFunction, 265
 - calledby mkNewModemapList, 238
 - calledby postDef, 371
 - calledby postError, 356
 - calledby resolve, 347
 - calledby setqMultipleExplicit, 324
 - calledby setqSingle, 326
- new2OldLisp, 458
 - calledby s-process, 490
 - calls new2OldTran, 458
 - calls postTransform, 458
 - defun, 458
- new2OldTran
 - calledby new2OldLisp, 458
- newComp
 - calledby compTopLevel, 494
- next-char, 449
 - calledby PARSE-FloatBase, 422
 - calls line-at-end-p, 450
 - calls line-next-char, 450
 - uses current-line, 450
- defun, 449
- next-line, 538
 - calledby Advance-Char, 539
 - local ref \$in-stream, 538
 - local ref \$line-handler, 538
 - defun, 538
- next-tab-loc, 467
 - defun, 467
- next-token, 91, 448
 - calledby match-next-token, 446
 - calls current-token, 448
 - calls try-get-token, 448
 - usedby next-token, 448
 - uses \$token, 91
 - uses next-token, 448
 - uses valid-tokens, 448
 - defun, 448
 - defvar, 91
- nonblank, 91
 - defvar, 91
- nonblankloc, 468
 - calledby add-parens-and-semis-to-line, 84
 - calls blankp, 468
 - defun, 468
- normalizeStatAndStringify
 - calledby reportOnFunctorCompilation, 192
- not, 124
 - defplist, 124
- notequal, 125
 - defplist, 125
- nreverse0
 - calledby aplTran1, 387
 - calledby compAdd, 248
 - calledby compCase1, 260
 - calledby compColon, 267
 - calledby compExpressionList, 512
 - calledby compForm1, 508
 - calledby compForm2, 513
 - calledby compJoin, 305
 - calledby compReduce1, 312
 - calledby compSeq1, 318
 - calledby getFormModemaps, 510
 - calledby getModemapList, 236
 - calledby hasAplExtension, 389
 - calledby mkAlistOfExplicitCategoryOps, 156

- calledby mkNewModemapList, 238
- calledby outputComp, 328
- calledby parseHas, 109
- calledby postCategory, 363
- calledby postDef, 371
- calledby postIf, 373
- calledby postMDef, 377
- calledby setqMultiple, 322
- calledby substNames, 243
- calledby transIs1, 102
- NRTassignCapsuleFunctionSlot
 - calledby compDefineCapsuleFunction, 280
- NRTassocIndex
 - calledby setqSingle, 326
- NRTgenInitialAttributeAlist
 - calledby compDefineFunctor1, 182
 - calledby finalizeLisplib, 174
- NRTgetLocalIndex
 - calledby compAdd, 248
 - calledby compDefineFunctor1, 182
 - calledby compSymbol, 505
 - calledby doIt, 253
- NRTgetLocalIndexClear
 - calledby doIt, 253
- NRTgetLookupFunction
 - calledby compDefineFunctor1, 182
- NRTmakeSlot1Info
 - calledby compDefineFunctor1, 182
- NRTputInTail
 - calledby putInLocalDomainReferences, 154
- nsubst
 - calledby substVars, 166
- nth-stack, 464
 - calledby PARSE-Category, 410
 - calledby PARSE-Sexpr1, 428
 - calls reduction-value, 464
 - calls stack-store, 464
 - defmacro, 464
- object2String
 - calledby compileSpad2Cmd, 476
 - calledby compileSpadLispCmd, 528
- opFf
 - calledby parseDEF, 101
- opOf
 - calledby augModemapsFromDomain, 232
 - calledby coerceSubset, 338
 - calledby comp2, 499
 - calledby compColonInside, 502
 - calledby compDefineCategory2, 142
 - calledby compElt, 292
 - calledby compPretend, 310
 - calledby compilerDoit, 478
 - calledby doIt, 253
 - calledby getSignatureFromMode, 278
 - calledby isFunctor, 229
 - calledby isSuperDomain, 231
 - calledby mkOpVec, 199
 - calledby optCallSpecially, 211
 - calledby parseHas, 108
 - calledby parseLET, 122
 - calledby parseMDEF, 123
- opt-, 218
 - defun, 218
- optCall, 209
 - calledby optSPADCALL, 219
 - calls optCallSpecially, 209
 - calls optPackageCall, 209
 - calls optimize, 209
 - calls rplac, 209
 - calls systemErrorHere, 209
 - local ref \$QuickCode, 209
 - local ref \$bootStrapMode, 209
 - defun, 209
- optCallEval, 213
 - calledby optSpecialCall, 212
 - calls FactoredForm, 213
 - calls Integer, 213
 - calls List, 213
 - calls Matrix, 213
 - calls PrimitivveArray, 213
 - calls Vector, 213
 - calls eval, 213
 - calls qcar, 213
 - defun, 213
- optCallSpecially, 211
 - calledby optCall, 209
 - calls get, 211
 - calls kar, 211
 - calls lassoc, 211
 - calls opOf, 211
 - calls optSpecialCall, 211

- local ref \$e, 211
 - local ref \$getDomainCode, 211
 - local ref \$optimizableConstructorNames, 211
 - local ref \$specialCaseKeyList, 211
 - defun, 211
- optCatch, 220
 - calls optimize, 220
 - calls qcar, 220
 - calls qcdr, 220
 - calls rplac, 220
 - local ref \$InteractiveMode, 220
 - defun, 220
- optCond, 222
 - calls EqualBarGensym, 222
 - calls TruthP, 222
 - calls qcar, 222
 - calls qcdr, 222
 - calls rplacd, 222
 - calls rplac, 222
 - defun, 222
- optCONDtail, 206
 - calledby optCONDtail, 206
 - calledby optXLAMCond, 206
 - calls optCONDtail, 206
 - local ref \$true, 206
 - defun, 206
- optEQ, 216
 - defun, 216
- optFunctorBody
 - calledby compDefineCategory2, 142
- optIF2COND, 207
 - calledby optIF2COND, 207
 - calledby optimize, 205
 - calls optIF2COND, 207
 - local ref \$true, 208
 - defun, 207
- optimize, 205
 - calledby lispize, 333
 - calledby optCall, 209
 - calledby optCatch, 220
 - calledby optSpecialCall, 212
 - calledby optimizeFunctionDef, 203
 - calledby optimize, 205
 - calls getl, 205
 - calls optIF2COND, 205
 - calls optimize, 205
 - calls prettyprint, 205
 - calls qcar, 205
 - calls qcdr, 205
 - calls rplac, 205
 - calls say, 205
 - calls subname, 205
 - defun, 205
- optimizeFunctionDef, 203
 - calledby compWithMappingMode1, 518
 - calledby compile, 145
 - calls bright, 204
 - calls optimize, 203
 - calls pp, 203
 - calls qcar, 203
 - calls qcdr, 203
 - calls rplac, 203
 - calls sayBrightlyI, 203
 - local ref \$reportOptimization, 204
 - defun, 203
- optional, 453
 - calledby PARSE-Application, 418
 - calledby PARSE-Category, 409
 - calledby PARSE-CommandTail, 407
 - calledby PARSE-Conditional, 437
 - calledby PARSE-Expr, 412
 - calledby PARSE-Form, 417
 - calledby PARSE-Import, 411
 - calledby PARSE-Infix, 415
 - calledby PARSE-IteratorTail, 433
 - calledby PARSE-Iterator, 433
 - calledby PARSE-Prefix, 415
 - calledby PARSE-Primary1, 420
 - calledby PARSE-PrimaryNoFloat, 420
 - calledby PARSE-ScriptItem, 427
 - calledby PARSE-Seg, 436
 - calledby PARSE-Sequence1, 431
 - calledby PARSE-Sexpr1, 428
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-Statement, 408
 - calledby PARSE-Suffix, 434
 - calledby PARSE-TokenCommandTail, 406
 - calledby PARSE-VarForm, 426
 - defun, 453
- optionlist
 - usedby spad, 489

- optLESSP, 218
 - defun, 218
- optMINUS, 216
 - defun, 216
- optMkRecord, 225
 - calls length, 225
 - defun, 225
- optPackageCall, 210
 - calledby optCall, 209
 - calls rplaca, 210
 - calls rplacd, 210
 - defun, 210
- optPredicateIfTrue, 207
 - calledby optXLAMCond, 206
 - local ref \$BasicPredicates, 207
 - defun, 207
- optQSMINUS, 217
 - defun, 217
- optRECORDCOPY, 227
 - defun, 227
- optRECORDELT, 225
 - calls keyedSystemError, 225
 - defun, 225
- optSEQ, 214
 - defun, 214
- optSETRECORDELT, 226
 - calls keyedSystemError, 226
 - defun, 226
- optSPADCALL, 219
 - calls optCall, 219
 - local ref \$InteractiveMode, 219
 - defun, 219
- optSpecialCall, 212
 - calledby optCallSpecially, 211
 - calls compileTimeBindingOf, 212
 - calls function, 212
 - calls getl, 212
 - calls keyedSystemError, 212
 - calls mkq, 212
 - calls optCallEval, 212
 - calls optimize, 212
 - calls rplaca, 212
 - calls rplacw, 212
 - calls rplac, 212
 - local ref \$QuickCode, 212
 - local ref \$Undef, 212
 - defun, 212
- optSuchthat, 220
 - defun, 220
- optXLAMCond, 206
 - calledby optXLAMCond, 206
 - calls optCONDtail, 206
 - calls optPredicateIfTrue, 206
 - calls optXLAMCond, 206
 - calls qcar, 206
 - calls qcdr, 206
 - calls rplac, 206
 - defun, 206
- or, 125
 - defplist, 125
- orderByDependency, 202
 - calledby compDefWhereClause, 200
 - calls intersection, 202
 - calls member, 202
 - calls remdup, 202
 - calls say, 202
 - calls userError, 202
 - defun, 202
- orderPredicateItems, 161
 - calledby fixUpPredicate, 160
 - calls orderPredTran, 161
 - calls qcar, 161
 - calls qcdr, 161
 - calls signatureTran, 161
 - defun, 161
- orderPredTran, 162
 - calledby orderPredicateItems, 161
 - calls delete, 162
 - calls insertWOC, 162
 - calls intersectionq, 162
 - calls isDomainSubst, 162
 - calls listOfPatternIds, 162
 - calls member, 162
 - calls qcar, 162
 - calls qcdr, 162
 - calls setdifference, 162
 - calls unionq, 162
 - defun, 162
- outerProduct
 - calledby compileCases, 283
- outputComp, 328
 - calledby compForm1, 508

- calledby outputComp, 328
- calledby setqSingle, 326
- calls comp, 328
- calls get, 328
- calls nreverse0, 328
- calls outputComp, 328
- calls qcar, 328
- calls qcdr, 328
- local ref \$Expression, 328
- defun, 328
- pack
 - calledby quote-if-string, 442
- pairList
 - calledby compDefWhereClause, 200
 - calledby formal2Pattern, 190
- PARSE-AnyId, 430
 - calledby PARSE-Sexpr1, 428
 - calls action, 430
 - calls advance-token, 430
 - calls current-symbol, 430
 - calls match-string, 430
 - calls parse-identifier, 430
 - calls parse-keyword, 430
 - calls push-reduction, 430
 - defun, 430
- PARSE-Application, 418
 - calledby PARSE-Application, 418
 - calledby PARSE-Category, 410
 - calledby PARSE-Form, 418
 - calls PARSE-Application, 418
 - calls PARSE-Primary, 418
 - calls PARSE-Selector, 418
 - calls optional, 418
 - calls pop-stack-1, 418
 - calls pop-stack-2, 418
 - calls push-reduction, 418
 - calls star, 418
 - defun, 418
- parse-argument-designator, 460
 - calledby PARSE-FormalParameterTok, 425
 - calls advance-token, 461
 - calls match-current-token, 460
 - calls push-reduction, 460
 - calls token-symbol, 460
 - defun, 460
- PARSE-Category, 409
 - calledby PARSE-Category, 409
 - calls PARSE-Application, 410
 - calls PARSE-Category, 409
 - calls PARSE-Expression, 409
 - calls action, 410
 - calls bang, 409
 - calls line-number, 410
 - calls match-advance-string, 409
 - calls must, 409
 - calls nth-stack, 410
 - calls optional, 409
 - calls pop-stack-1, 410
 - calls pop-stack-2, 409
 - calls pop-stack-3, 409
 - calls push-reduction, 409
 - calls recordAttributeDocumentation, 410
 - calls recordSignatureDocumentation, 410
 - calls star, 410
 - uses current-line, 410
 - defun, 409
- PARSE-Command, 404
 - calls PARSE-SpecialCommand, 404
 - calls PARSE-SpecialKeyword, 404
 - calls match-advance-string, 404
 - calls must, 404
 - calls push-reduction, 404
 - defun, 404
- PARSE-CommandTail, 407
 - calledby PARSE-CommandTail, 407
 - calledby PARSE-SpecialCommand, 405
 - calls PARSE-CommandTail, 407
 - calls PARSE-Option, 407
 - calls action, 407
 - calls bang, 407
 - calls optional, 407
 - calls pop-stack-1, 407
 - calls pop-stack-2, 407
 - calls push-reduction, 407
 - calls star, 407
 - calls systemCommand[5], 407
 - defun, 407
- PARSE-Conditional, 437
 - calledby PARSE-ElseClause, 437
 - calls PARSE-ElseClause, 437
 - calls PARSE-Expression, 437

- calls bang, 437
- calls match-advance-string, 437
- calls must, 437
- calls optional, 437
- calls pop-stack-1, 437
- calls pop-stack-2, 437
- calls pop-stack-3, 437
- calls push-reduction, 437
- defun, 437
- PARSE-Data, 428
 - calledby PARSE-Primary1, 421
 - calls PARSE-Sexpr, 428
 - calls action, 428
 - calls pop-stack-1, 428
 - calls push-reduction, 428
 - calls translable, 428
 - uses labasoc, 428
 - defun, 428
- PARSE-ElseClause, 437
 - calledby PARSE-Conditional, 437
 - calls PARSE-Conditional, 437
 - calls PARSE-Expression, 438
 - calls current-symbol, 437
 - defun, 437
- PARSE-Enclosure, 424
 - calledby PARSE-Primary1, 421
 - calls PARSE-Expr, 424
 - calls match-advance-string, 424
 - calls must, 424
 - calls pop-stack-1, 424
 - calls push-reduction, 424
 - defun, 424
- PARSE-Exit, 435
 - calls PARSE-Expression, 435
 - calls match-advance-string, 435
 - calls must, 435
 - calls pop-stack-1, 436
 - calls push-reduction, 435
 - defun, 435
- PARSE-Expr, 412
 - calledby PARSE-Enclosure, 424
 - calledby PARSE-Expression, 411
 - calledby PARSE-Import, 411
 - calledby PARSE-Iterator, 433
 - calledby PARSE-LabelExpr, 438
 - calledby PARSE-Loop, 438
 - calledby PARSE-Primary1, 421
 - calledby PARSE-Reduction, 417
 - calledby PARSE-ScriptItem, 427
 - calledby PARSE-SemiColon, 435
 - calledby PARSE-Statement, 408
 - calls PARSE-LedPart, 412
 - calls PARSE-NudPart, 412
 - calls optional, 412
 - calls pop-stack-1, 412
 - calls push-reduction, 412
 - calls star, 412
 - defun, 412
- PARSE-Expression, 411
 - calledby PARSE-Category, 409
 - calledby PARSE-Conditional, 437
 - calledby PARSE-ElseClause, 438
 - calledby PARSE-Exit, 435
 - calledby PARSE-Infix, 415
 - calledby PARSE-Iterator, 433
 - calledby PARSE-Leave, 436
 - calledby PARSE-Prefix, 415
 - calledby PARSE-Return, 435
 - calledby PARSE-Seg, 436
 - calledby PARSE-Sequence1, 431
 - calledby PARSE-SpecialCommand, 405
 - calls PARSE-Expr, 411
 - calls PARSE-rightBindingPowerOf, 411
 - calls make-symbol-of, 411
 - calls pop-stack-1, 411
 - calls push-reduction, 411
 - uses ParseMode, 411
 - uses prior-token, 411
 - defun, 411
- PARSE-Float, 421
 - calledby PARSE-Primary, 420
 - calledby PARSE-Selector, 419
 - calls PARSE-FloatBase, 421
 - calls PARSE-FloatExponent, 421
 - calls make-float, 421
 - calls must, 421
 - calls pop-stack-1, 421
 - calls pop-stack-2, 421
 - calls pop-stack-3, 421
 - calls pop-stack-4, 421
 - calls push-reduction, 421
 - defun, 421

- PARSE-FloatBase, 422
 - calledby PARSE-Float, 421
 - calls PARSE-FloatBasePart, 422
 - calls PARSE-IntegerTok, 422
 - calls char-eq, 422
 - calls char-ne, 422
 - calls current-char, 422
 - calls current-symbol, 422
 - calls digitp[5], 422
 - calls must, 422
 - calls next-char, 422
 - calls push-reduction, 422
 - defun, 422
- PARSE-FloatBasePart, 422
 - calledby PARSE-FloatBase, 422
 - calls PARSE-IntegerTok, 423
 - calls current-char, 423
 - calls current-token, 423
 - calls digitp[5], 423
 - calls match-advance-string, 422
 - calls must, 423
 - calls push-reduction, 423
 - calls token-nonblank, 423
 - defun, 422
- PARSE-FloatExponent, 423
 - calledby PARSE-Float, 421
 - calls PARSE-IntegerTok, 423
 - calls action, 423
 - calls advance-token, 423
 - calls current-char, 423
 - calls current-symbol, 423
 - calls floatexpid, 423
 - calls identp[5], 423
 - calls match-advance-string, 423
 - calls must, 423
 - calls push-reduction, 423
 - defun, 423
- PARSE-FloatTok, 439
 - calls bfp-, 439
 - calls parse-number, 439
 - calls pop-stack-1, 439
 - calls push-reduction, 439
 - local ref \$boot, 439
 - defun, 439
- PARSE-Form, 417
 - calledby PARSE-NudPart, 412
 - calls PARSE-Application, 418
 - calls bang, 417
 - calls match-advance-string, 417
 - calls must, 417
 - calls optional, 417
 - calls pop-stack-1, 418
 - calls push-reduction, 417
 - defun, 417
- PARSE-FormalParameter, 425
 - calledby PARSE-Primary1, 421
 - calls PARSE-FormalParameterTok, 425
 - defun, 425
- PARSE-FormalParameterTok, 425
 - calledby PARSE-FormalParameter, 425
 - calls parse-argument-designator, 425
 - defun, 425
- PARSE-getSemanticForm, 414
 - calledby PARSE-Operation, 413
 - calls PARSE-Infix, 414
 - calls PARSE-Prefix, 414
 - defun, 414
- PARSE-GlyphTok, 430
 - calledby PARSE-Quad, 425
 - calledby PARSE-Seg, 436
 - calledby PARSE-Sexpr1, 429
 - calls action, 430
 - calls advance-token, 430
 - calls match-current-token, 430
 - uses tok, 430
 - defun, 430
- parse-identifier, 459
 - calledby PARSE-AnyId, 430
 - calledby PARSE-Name, 427
 - calls advance-token, 459
 - calls match-current-token, 459
 - calls push-reduction, 459
 - calls token-symbol, 459
 - defun, 459
- PARSE-Import, 411
 - calls PARSE-Expr, 411
 - calls bang, 411
 - calls match-advance-string, 411
 - calls must, 411
 - calls optional, 411
 - calls pop-stack-1, 411
 - calls pop-stack-2, 411

- calls push-reduction, 411
- calls star, 411
- defun, 411
- PARSE-Infix, 415
 - calledby PARSE-getSemanticForm, 414
 - calls PARSE-Expression, 415
 - calls PARSE-TokTail, 415
 - calls action, 415
 - calls advance-token, 415
 - calls current-symbol, 415
 - calls must, 415
 - calls optional, 415
 - calls pop-stack-1, 415
 - calls pop-stack-2, 415
 - calls push-reduction, 415
 - defun, 415
- PARSE-InfixWith, 409
 - calls PARSE-With, 409
 - calls pop-stack-1, 409
 - calls pop-stack-2, 409
 - calls push-reduction, 409
 - defun, 409
- PARSE-IntegerTok, 424
 - calledby PARSE-FloatBasePart, 423
 - calledby PARSE-FloatBase, 422
 - calledby PARSE-FloatExponent, 423
 - calledby PARSE-Primary1, 421
 - calledby PARSE-Sexpr1, 428
 - calls parse-number, 424
 - defun, 424
- PARSE-Iterator, 433
 - calledby PARSE-IteratorTail, 433
 - calledby PARSE-Loop, 438
 - calls PARSE-Expression, 433
 - calls PARSE-Expr, 433
 - calls PARSE-Primary, 433
 - calls match-advance-string, 433
 - calls must, 433
 - calls optional, 433
 - calls pop-stack-1, 433
 - calls pop-stack-2, 433
 - calls pop-stack-3, 433
 - defun, 433
- PARSE-IteratorTail, 433
 - calledby PARSE-Sequence1, 431
 - calls PARSE-Iterator, 433
- calls bang, 433
- calls match-advance-string, 433
- calls optional, 433
- calls star, 433
- defun, 433
- parse-keyword, 460
 - calledby PARSE-AnyId, 430
 - calls advance-token, 460
 - calls match-current-token, 460
 - calls push-reduction, 460
 - calls token-symbol, 460
 - defun, 460
- PARSE-Label, 419
 - calledby PARSE-LabelExpr, 438
 - calledby PARSE-Leave, 436
 - calls PARSE-Name, 419
 - calls match-advance-string, 419
 - calls must, 419
 - defun, 419
- PARSE-LabelExpr, 438
 - calls PARSE-Expr, 438
 - calls PARSE-Label, 438
 - calls must, 438
 - calls pop-stack-1, 438
 - calls pop-stack-2, 438
 - calls push-reduction, 438
 - defun, 438
- PARSE-Leave, 436
 - calls PARSE-Expression, 436
 - calls PARSE-Label, 436
 - calls match-advance-string, 436
 - calls must, 436
 - calls pop-stack-1, 436
 - calls push-reduction, 436
 - defun, 436
- PARSE-LedPart, 412
 - calledby PARSE-Expr, 412
 - calls PARSE-Operation, 412
 - calls pop-stack-1, 412
 - calls push-reduction, 412
 - defun, 412
- PARSE-leftBindingPowerOf, 413
 - calledby PARSE-Operation, 413
 - calls elemn, 414
 - calls get1, 413
 - defun, 413

- PARSE-Loop, 438
 - calls PARSE-Expr, 438
 - calls PARSE-Iterator, 438
 - calls match-advance-string, 438
 - calls must, 438
 - calls pop-stack-1, 438
 - calls pop-stack-2, 438
 - calls push-reduction, 438
 - calls star, 438
 - defun, 438
- PARSE-Name, 427
 - calledby PARSE-Label, 419
 - calledby PARSE-VarForm, 426
 - calls parse-identifier, 427
 - calls pop-stack-1, 427
 - calls push-reduction, 427
 - defun, 427
- PARSE-NBGlyphTok, 429
 - calledby PARSE-Sexpr1, 428
 - calls action, 429
 - calls advance-token, 429
 - calls match-current-token, 429
 - uses tok, 429
 - defun, 429
- PARSE-NewExpr, 403
 - calledby spad, 489
 - calls PARSE-Statement, 403
 - calls action, 403
 - calls current-symbol, 403
 - calls match-string, 403
 - calls must, 403
 - calls processSynonyms[5], 403
 - uses definition-name, 403
 - defun, 403
- PARSE-NudPart, 412
 - calledby PARSE-Expr, 412
 - calls PARSE-Form, 412
 - calls PARSE-Operation, 412
 - calls PARSE-Reduction, 412
 - calls pop-stack-1, 413
 - calls push-reduction, 412
 - uses rbp, 413
 - defun, 412
- parse-number, 460
 - calledby PARSE-FloatTok, 439
 - calledby PARSE-IntegerTok, 424
 - calls advance-token, 460
 - calls match-current-token, 460
 - calls push-reduction, 460
 - calls token-symbol, 460
 - defun, 460
- PARSE-OpenBrace, 432
 - calledby PARSE-Sequence, 431
 - calls action, 432
 - calls advance-token, 432
 - calls current-symbol, 432
 - calls eqcar, 432
 - calls getToken, 432
 - calls push-reduction, 432
 - defun, 432
- PARSE-OpenBracket, 432
 - calledby PARSE-Sequence, 431
 - calls action, 432
 - calls advance-token, 432
 - calls current-symbol, 432
 - calls eqcar, 432
 - calls getToken, 432
 - calls push-reduction, 432
 - defun, 432
- PARSE-Operation, 413
 - calledby PARSE-LedPart, 412
 - calledby PARSE-NudPart, 412
 - calls PARSE-getSemanticForm, 413
 - calls PARSE-leftBindingPowerOf, 413
 - calls PARSE-rightBindingPowerOf, 413
 - calls action, 413
 - calls current-symbol, 413
 - calls elemn, 413
 - calls getl, 413
 - calls lt, 413
 - calls match-current-token, 413
 - uses ParseMode, 413
 - uses rbp, 413
 - uses tmptok, 413
 - defun, 413
- PARSE-Option, 408
 - calledby PARSE-CommandTail, 407
 - calls PARSE-PrimaryOrQM, 408
 - calls match-advance-string, 408
 - calls must, 408
 - calls star, 408
 - defun, 408

- PARSE-Prefix, 414
 - calledby PARSE-getSemanticForm, 414
 - calls PARSE-Expression, 415
 - calls PARSE-TokTail, 415
 - calls action, 414
 - calls advance-token, 415
 - calls current-symbol, 414
 - calls must, 415
 - calls optional, 415
 - calls pop-stack-1, 415
 - calls pop-stack-2, 415
 - calls push-reduction, 414, 415
 - defun, 414
- PARSE-Primary, 420
 - calledby PARSE-Application, 418
 - calledby PARSE-Iterator, 433
 - calledby PARSE-PrimaryOrQM, 407
 - calledby PARSE-Selector, 419
 - calls PARSE-Float, 420
 - calls PARSE-PrimaryNoFloat, 420
 - defun, 420
- PARSE-Primary1, 420
 - calledby PARSE-Primary1, 420
 - calledby PARSE-PrimaryNoFloat, 420
 - calledby PARSE-Qualification, 416
 - calls PARSE-Data, 421
 - calls PARSE-Enclosure, 421
 - calls PARSE-Expr, 421
 - calls PARSE-FormalParameter, 421
 - calls PARSE-IntegerTok, 421
 - calls PARSE-Primary1, 420
 - calls PARSE-Quad, 421
 - calls PARSE-Sequence, 421
 - calls PARSE-String, 421
 - calls PARSE-VarForm, 420
 - calls current-symbol, 420
 - calls match-advance-string, 421
 - calls match-string, 421
 - calls must, 420
 - calls optional, 420
 - calls pop-stack-1, 420
 - calls pop-stack-2, 420
 - calls push-reduction, 420
 - local ref \$boot, 421
 - defun, 420
- PARSE-PrimaryNoFloat, 420
 - calledby PARSE-Primary, 420
 - calledby PARSE-Selector, 419
 - calls PARSE-Primary1, 420
 - calls PARSE-TokTail, 420
 - calls optional, 420
 - defun, 420
- PARSE-PrimaryOrQM, 407
 - calledby PARSE-Option, 408
 - calledby PARSE-PrimaryOrQM, 407
 - calledby PARSE-SpecialCommand, 405
 - calls PARSE-PrimaryOrQM, 407
 - calls PARSE-Primary, 407
 - calls match-advance-string, 407
 - calls push-reduction, 407
 - defun, 407
- PARSE-Quad, 425
 - calledby PARSE-Primary1, 421
 - calls PARSE-GlyphTok, 425
 - calls match-advance-string, 425
 - calls push-reduction, 425
 - uses \$boot, 425
 - defun, 425
- PARSE-Qualification, 416
 - calledby PARSE-TokTail, 416
 - calls PARSE-Primary1, 416
 - calls dollarTran, 416
 - calls match-advance-string, 416
 - calls must, 416
 - calls pop-stack-1, 416
 - calls push-reduction, 416
 - defun, 416
- PARSE-Reduction, 417
 - calledby PARSE-NudPart, 412
 - calls PARSE-Expr, 417
 - calls PARSE-ReductionOp, 417
 - calls must, 417
 - calls pop-stack-1, 417
 - calls pop-stack-2, 417
 - calls push-reduction, 417
 - defun, 417
- PARSE-ReductionOp, 417
 - calledby PARSE-Reduction, 417
 - calls action, 417
 - calls advance-token, 417
 - calls current-symbol, 417
 - calls getl, 417

- calls match-next-token, 417
- defun, 417
- PARSE-Return, 435
 - calls PARSE-Expression, 435
 - calls match-advance-string, 435
 - calls must, 435
 - calls pop-stack-1, 435
 - calls push-reduction, 435
 - defun, 435
- PARSE-rightBindingPowerOf, 414
 - calledby PARSE-Expression, 411
 - calledby PARSE-Operation, 413
 - calls elemn, 414
 - calls getl, 414
 - defun, 414
- PARSE-ScriptItem, 427
 - calledby PARSE-ScriptItem, 427
 - calledby PARSE-Scripts, 426
 - calls PARSE-Expr, 427
 - calls PARSE-ScriptItem, 427
 - calls match-advance-string, 427
 - calls must, 427
 - calls optional, 427
 - calls pop-stack-1, 427
 - calls pop-stack-2, 427
 - calls push-reduction, 427
 - calls star, 427
 - defun, 427
- PARSE-Scripts, 426
 - calledby PARSE-VarForm, 426
 - calls PARSE-ScriptItem, 426
 - calls match-advance-string, 426
 - calls must, 426
 - defun, 426
- PARSE-Seg, 436
 - calls PARSE-Expression, 436
 - calls PARSE-GlyphTok, 436
 - calls bang, 436
 - calls optional, 436
 - calls pop-stack-1, 437
 - calls pop-stack-2, 437
 - calls push-reduction, 436
 - defun, 436
- PARSE-Selector, 419
 - calledby PARSE-Application, 418
 - calls PARSE-Float, 419
 - calls PARSE-PrimaryNoFloat, 419
 - calls PARSE-Primary, 419
 - calls char-ne, 419
 - calls current-char, 419
 - calls current-symbol, 419
 - calls match-advance-string, 419
 - calls must, 419
 - calls pop-stack-1, 419
 - calls pop-stack-2, 419
 - calls push-reduction, 419
 - uses \$boot, 419
 - defun, 419
- PARSE-SemiColon, 435
 - calls PARSE-Expr, 435
 - calls match-advance-string, 435
 - calls must, 435
 - calls pop-stack-1, 435
 - calls pop-stack-2, 435
 - calls push-reduction, 435
 - defun, 435
- PARSE-Sequence, 431
 - calledby PARSE-Primary1, 421
 - calls PARSE-OpenBrace, 431
 - calls PARSE-OpenBracket, 431
 - calls PARSE-Sequence1, 431
 - calls match-advance-string, 431
 - calls must, 431
 - calls pop-stack-1, 431
 - calls push-reduction, 431
 - defun, 431
- PARSE-Sequence1, 431
 - calledby PARSE-Sequence, 431
 - calls PARSE-Expression, 431
 - calls PARSE-IteratorTail, 431
 - calls optional, 431
 - calls pop-stack-1, 431
 - calls pop-stack-2, 431
 - calls push-reduction, 431
 - defun, 431
- PARSE-Sexpr, 428
 - calledby PARSE-Data, 428
 - calls PARSE-Sexpr1, 428
 - defun, 428
- PARSE-Sexpr1, 428
 - calledby PARSE-Sexpr1, 428
 - calledby PARSE-Sexpr, 428

- calls PARSE-AnyId, 428
- calls PARSE-GlyphTok, 429
- calls PARSE-IntegerTok, 428
- calls PARSE-NBGlyphTok, 428
- calls PARSE-Sexpr1, 428
- calls PARSE-String, 429
- calls action, 428
- calls bang, 429
- calls match-advance-string, 428
- calls must, 428
- calls nth-stack, 428
- calls optional, 428
- calls pop-stack-1, 429
- calls pop-stack-2, 428
- calls push-reduction, 428
- calls star, 429
- defun, 428
- parse-spadstring, 458
 - calledby PARSE-String, 425
 - calls advance-token, 458
 - calls match-current-token, 458
 - calls push-reduction, 458
 - calls token-symbol, 458
 - defun, 458
- PARSE-SpecialCommand, 405
 - calledby PARSE-Command, 404
 - calledby PARSE-SpecialCommand, 405
 - calls PARSE-CommandTail, 405
 - calls PARSE-Expression, 405
 - calls PARSE-PrimaryOrQM, 405
 - calls PARSE-SpecialCommand, 405
 - calls PARSE-TokenCommandTail, 405
 - calls PARSE-TokenList, 405
 - calls action, 405
 - calls bang, 405
 - calls current-symbol, 405
 - calls match-advance-string, 405
 - calls must, 405
 - calls optional, 405
 - calls pop-stack-1, 405
 - calls push-reduction, 405
 - calls star, 405
 - local ref \$noParseCommands, 405
 - local ref \$tokenCommands, 405
 - defun, 405
- PARSE-SpecialKeyWord, 404
 - calledby PARSE-Command, 404
 - calls action, 404
 - calls current-symbol, 404
 - calls current-token, 404
 - calls match-current-token, 404
 - calls token-symbol, 404
 - calls unAbbreviateKeyword[5], 404
 - defun, 404
- PARSE-Statement, 408
 - calledby PARSE-NewExpr, 403
 - calls PARSE-Expr, 408
 - calls match-advance-string, 408
 - calls must, 408
 - calls optional, 408
 - calls pop-stack-1, 408
 - calls pop-stack-2, 408
 - calls push-reduction, 408
 - calls star, 408
 - defun, 408
- PARSE-String, 425
 - calledby PARSE-Primary1, 421
 - calledby PARSE-Sexpr1, 429
 - calls parse-spadstring, 425
 - defun, 425
- parse-string, 459
 - calls advance-token, 459
 - calls match-current-token, 459
 - calls push-reduction, 459
 - calls token-symbol, 459
 - defun, 459
- PARSE-Suffix, 434
 - calls PARSE-TokTail, 434
 - calls action, 434
 - calls advance-token, 434
 - calls current-symbol, 434
 - calls optional, 434
 - calls pop-stack-1, 434
 - calls push-reduction, 434
 - defun, 434
- PARSE-TokenCommandTail, 405
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-TokenCommandTail, 406
 - calls PARSE-TokenCommandTail, 406
 - calls PARSE-TokenOption, 406
 - calls action, 406
 - calls atEndOfLine, 406

- calls bang, 405
- calls optional, 406
- calls pop-stack-1, 406
- calls pop-stack-2, 406
- calls push-reduction, 406
- calls star, 406
- calls systemCommand[5], 406
- defun, 405
- PARSE-TokenList, 406
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-TokenOption, 406
 - calls action, 406
 - calls advance-token, 406
 - calls current-symbol, 406
 - calls isTokenDelimiter, 406
 - calls push-reduction, 406
 - calls star, 406
 - defun, 406
- PARSE-TokenOption, 406
 - calledby PARSE-TokenCommandTail, 406
 - calls PARSE-TokenList, 406
 - calls match-advance-string, 406
 - calls must, 406
 - defun, 406
- PARSE-TokTail, 416
 - calledby PARSE-Infix, 415
 - calledby PARSE-Prefix, 415
 - calledby PARSE-PrimaryNoFloat, 420
 - calledby PARSE-Suffix, 434
 - calls PARSE-Qualification, 416
 - calls action, 416
 - calls char-eq, 416
 - calls copy-token, 416
 - calls current-char, 416
 - calls current-symbol, 416
 - uses \$boot, 416
 - defun, 416
- PARSE-VarForm, 426
 - calledby PARSE-Primary1, 420
 - calls PARSE-Name, 426
 - calls PARSE-Scripts, 426
 - calls optional, 426
 - calls pop-stack-1, 426
 - calls pop-stack-2, 426
 - calls push-reduction, 426
 - defun, 426
- PARSE-With, 409
 - calledby PARSE-InfixWith, 409
 - calls match-advance-string, 409
 - calls must, 409
 - calls pop-stack-1, 409
 - calls push-reduction, 409
 - defun, 409
- parseAnd, 97
 - calledby parseAnd, 97
 - calls parseAnd, 97
 - calls parseIf, 97
 - calls parseTranList, 97
 - calls parseTran, 97
 - uses \$InteractiveMode, 97
 - defun, 97
- parseAtom, 94
 - calledby parseTran, 93
 - calls parseLeave, 94
 - uses \$NoValue, 94
 - defun, 94
- parseAtSign, 98
 - calls parseTran, 98
 - calls parseType, 98
 - uses \$InteractiveMode, 98
 - defun, 98
- parseCategory, 99
 - calls contained, 99
 - calls parseDropAssertions, 99
 - calls parseTranList, 99
 - defun, 99
- parseCoerce, 100
 - calls parseTran, 100
 - calls parseType, 100
 - uses \$InteractiveMode, 100
 - defun, 100
- parseColon, 100
 - calls parseTran, 100
 - calls parseType, 100
 - local ref \$insideConstructIfTrue, 100
 - uses \$InteractiveMode, 100
 - defun, 100
- parseConstruct, 95
 - calledby parseTran, 93
 - calls parseTranList, 95
 - uses \$insideConstructIfTrue, 95
 - defun, 95

- parseDEF, 101
 - calls opFf, 101
 - calls parseLhs, 101
 - calls parseTranCheckForRecord, 101
 - calls parseTranList, 101
 - calls setDefOp, 101
 - uses \$lhs, 101
 - defun, 101
- parseDollarGreaterEqual, 104
 - calls msubst, 104
 - calls parseTran, 104
 - uses \$op, 104
 - defun, 104
- parseDollarGreaterThan, 104
 - calls msubst, 104
 - calls parseTran, 104
 - uses \$op, 104
 - defun, 104
- parseDollarLessEqual, 105
 - calls msubst, 105
 - calls parseTran, 105
 - uses \$op, 105
 - defun, 105
- parseDollarNotEqual, 105
 - calls msubst, 106
 - calls parseTran, 105
 - uses \$op, 106
 - defun, 105
- parseDropAssertions, 99
 - calledby parseCategory, 99
 - calledby parseDropAssertions, 99
 - calls parseDropAssertions, 99
 - defun, 99
- parseEquivalence, 106
 - calls parseIf, 106
 - defun, 106
- parseExit, 107
 - calls moan, 107
 - calls parseTran, 107
 - defun, 107
- parseGreaterEqual, 107
 - calls parseTran, 107
 - uses \$op, 107
 - defun, 107
- parseGreaterThan, 108
 - calls parseTran, 108
- uses \$op, 108
 - defun, 108
- parseHas, 108
 - calls getdatabase, 108
 - calls makeNonAtomic, 109
 - calls member, 109
 - calls nreverse0, 109
 - calls opOf, 108
 - calls parseHasRhs, 109
 - calls parseType, 109
 - calls qcar, 108
 - calls qcdr, 108
 - calls unabbrevAndLoad, 108
 - uses \$CategoryFrame, 109
 - uses \$InteractiveMode, 109
 - defun, 108
- parseHasRhs, 110
 - calledby parseHas, 109
 - calls abbreviation?, 110
 - calls get, 110
 - calls loadIfNecessary, 110
 - calls member, 110
 - calls qcar, 110
 - calls qcdr, 110
 - calls unabbrevAndLoad, 110
 - uses \$CategoryFrame, 110
 - defun, 110
- parseIf, 114
 - calledby parseAnd, 97
 - calledby parseEquivalence, 106
 - calledby parseImplies, 116
 - calledby parseOr, 126
 - calls parseIf,ifTran, 114
 - calls parseTran, 114
 - defun, 114
- parseIf,ifTran, 114
 - calledby parseIf,ifTran, 114
 - calledby parseIf, 114
 - calls incExitLevel, 114
 - calls makeSimplePredicateOrNil, 114
 - calls parseIf,ifTran, 114
 - calls parseTran, 114
 - uses \$InteractiveMode, 114
 - defun, 114
- parseImplies, 116
 - calls parseIf, 116

- defun, 116
- parseIn, 117
 - calledby parseInBy, 118
 - calls parseTran, 117
 - calls postError, 117
 - defun, 117
- parseInBy, 118
 - calls bright, 118
 - calls parseIn, 118
 - calls parseTran, 118
 - calls postError, 118
 - defun, 118
- parseIs, 119
 - calls parseTran, 119
 - calls transIs, 119
 - defun, 119
- parseIsnt, 120
 - calls parseTran, 120
 - calls transIs, 120
 - defun, 120
- parseJoin, 120
 - calls parseTranList, 120
 - defun, 120
- parseLeave, 121
 - calledby parseAtom, 94
 - calls parseTran, 121
 - defun, 121
- parseLessEqual, 122
 - calls parseTran, 122
 - uses \$op, 122
 - defun, 122
- parseLET, 122
 - calls opOf, 122
 - calls parseTranCheckForRecord, 122
 - calls parseTran, 122
 - calls transIs, 122
 - defun, 122
- parseLETD, 123
 - calls parseTran, 123
 - calls parseType, 123
 - defun, 123
- parseLhs, 102
 - calledby parseDEF, 101
 - calls parseTran, 102
 - calls transIs, 102
 - defun, 102
- parseMDEF, 123
 - calls opOf, 123
 - calls parseTranCheckForRecord, 123
 - calls parseTranList, 123
 - calls parseTran, 123
 - uses \$lhs, 123
 - defun, 123
- ParseMode, 403
 - usedby PARSE-Expression, 411
 - usedby PARSE-Operation, 413
 - defvar, 403
- parseNot, 124
 - calls parseTran, 124
 - uses \$InteractiveMode, 124
 - defun, 124
- parseNotEqual, 125
 - calls msubst, 125
 - calls parseTran, 125
 - uses \$op, 125
 - defun, 125
- parseOr, 125
 - calledby parseOr, 126
 - calls parseIf, 126
 - calls parseOr, 126
 - calls parseTranList, 126
 - calls parseTran, 125
 - defun, 125
- parsepiles, 84
 - calledby preparse1, 81
 - calls add-parens-and-semis-to-line, 84
 - defun, 84
- parsePretend, 126
 - calls parseTran, 126
 - calls parseType, 126
 - defun, 126
- parseprint, 468
 - calledby preparse, 76
 - defun, 468
- parseReturn, 127
 - calls moan, 127
 - calls parseTran, 127
 - defun, 127
- parseSegment, 128
 - calls parseTran, 128
 - defun, 128
- parseSeq, 128

- calls last, 128
- calls mapInto, 128
- calls postError, 128
- calls transSeq, 128
- defun, 128
- parseTran, 93
 - calledby compReduce1, 312
 - calledby parseAnd, 97
 - calledby parseAtSign, 98
 - calledby parseCoerce, 100
 - calledby parseColon, 100
 - calledby parseDollarGreaterEqual, 104
 - calledby parseDollarGreaterThan, 104
 - calledby parseDollarLessEqual, 105
 - calledby parseDollarNotEqual, 105
 - calledby parseExit, 107
 - calledby parseGreaterEqual, 107
 - calledby parseGreaterThan, 108
 - calledby parseIf.ifTran, 114
 - calledby parseIf, 114
 - calledby parseInBy, 118
 - calledby parseIn, 117
 - calledby parseIsnt, 120
 - calledby parseIs, 119
 - calledby parseLETD, 123
 - calledby parseLET, 122
 - calledby parseLeave, 121
 - calledby parseLessEqual, 122
 - calledby parseLhs, 102
 - calledby parseMDEF, 123
 - calledby parseNotEqual, 125
 - calledby parseNot, 124
 - calledby parseOr, 125
 - calledby parsePretend, 126
 - calledby parseReturn, 127
 - calledby parseSegment, 128
 - calledby parseTranCheckForRecord, 457
 - calledby parseTranList, 95
 - calledby parseTransform, 93
 - calledby parseTran, 93
 - calledby parseType, 98
 - calls getl, 93
 - calls parseAtom, 93
 - calls parseConstruct, 93
 - calls parseTranList, 93
 - calls parseTran, 93
 - uses \$op, 93
 - defun, 93
- parseTranCheckForRecord, 457
 - calledby parseDEF, 101
 - calledby parseLET, 122
 - calledby parseMDEF, 123
 - calls parseTran, 457
 - calls postError, 457
 - calls qcar, 457
 - calls qcdr, 457
 - defun, 457
- parseTranList, 95
 - calledby parseAnd, 97
 - calledby parseCategory, 99
 - calledby parseConstruct, 95
 - calledby parseDEF, 101
 - calledby parseJoin, 120
 - calledby parseMDEF, 123
 - calledby parseOr, 126
 - calledby parseTranList, 95
 - calledby parseTran, 93
 - calledby parseVCONS, 129
 - calls parseTranList, 95
 - calls parseTran, 95
 - defun, 95
- parseTransform, 93
 - calledby s-process, 490
 - calls msubst, 93
 - calls parseTran, 93
 - uses \$defOp, 93
 - defun, 93
- parseType, 98
 - calledby parseAtSign, 98
 - calledby parseCoerce, 100
 - calledby parseColon, 100
 - calledby parseHas, 109
 - calledby parseLETD, 123
 - calledby parsePretend, 126
 - calls msubst, 98
 - calls parseTran, 98
 - defun, 98
- parseVCONS, 129
 - calls parseTranList, 129
 - defun, 129
- parseWhere, 129
 - calls mapInto, 129

- defun, 129
- pathname[5]
 - called by compileSpad2Cmd, 476
 - called by compileSpadLispCmd, 528
 - called by compiler, 473
- pathnameDirectory[5]
 - called by compileSpadLispCmd, 528
- pathnameName[5]
 - called by compileSpadLispCmd, 528
- pathnameType[5]
 - called by compileSpad2Cmd, 476
 - called by compileSpadLispCmd, 528
 - called by compiler, 473
- pathnameTypeId
 - called by initializeLisplib, 173
- pmatch
 - called by coerceable, 341
- pname
 - called by compDefineFunctor1, 181
 - called by encodeItem, 150
 - called by mkCategoryPackage, 139
- pname[5]
 - called by comp3, 500
 - called by floatexpid, 451
 - called by getScriptName, 391
- Pop-Reduction, 464
 - called by pop-stack-1, 462
 - called by pop-stack-2, 463
 - called by pop-stack-3, 463
 - called by pop-stack-4, 463
 - calls stack-pop, 464
 - defun, 464
- pop-stack-1, 462
 - called by PARSE-Application, 418
 - called by PARSE-Category, 410
 - called by PARSE-CommandTail, 407
 - called by PARSE-Conditional, 437
 - called by PARSE-Data, 428
 - called by PARSE-Enclosure, 424
 - called by PARSE-Exit, 436
 - called by PARSE-Expression, 411
 - called by PARSE-Expr, 412
 - called by PARSE-FloatTok, 439
 - called by PARSE-Float, 421
 - called by PARSE-Form, 418
 - called by PARSE-Import, 411
 - called by PARSE-InfixWith, 409
 - called by PARSE-Infix, 415
 - called by PARSE-Iterator, 433
 - called by PARSE-LabelExpr, 438
 - called by PARSE-Leave, 436
 - called by PARSE-LedPart, 412
 - called by PARSE-Loop, 438
 - called by PARSE-Name, 427
 - called by PARSE-NudPart, 413
 - called by PARSE-Prefix, 415
 - called by PARSE-Primary1, 420
 - called by PARSE-Qualification, 416
 - called by PARSE-Reduction, 417
 - called by PARSE-Return, 435
 - called by PARSE-ScriptItem, 427
 - called by PARSE-Seg, 437
 - called by PARSE-Selector, 419
 - called by PARSE-SemiColon, 435
 - called by PARSE-Sequence1, 431
 - called by PARSE-Sequence, 431
 - called by PARSE-Sexpr1, 429
 - called by PARSE-SpecialCommand, 405
 - called by PARSE-Statement, 408
 - called by PARSE-Suffix, 434
 - called by PARSE-TokenCommandTail, 406
 - called by PARSE-VarForm, 426
 - called by PARSE-With, 409
 - called by spad, 489
 - called by star, 453
 - calls Pop-Reduction, 462
 - calls reduction-value, 462
 - defmacro, 462
- pop-stack-2, 463
 - called by PARSE-Application, 418
 - called by PARSE-Category, 409
 - called by PARSE-CommandTail, 407
 - called by PARSE-Conditional, 437
 - called by PARSE-Float, 421
 - called by PARSE-Import, 411
 - called by PARSE-InfixWith, 409
 - called by PARSE-Infix, 415
 - called by PARSE-Iterator, 433
 - called by PARSE-LabelExpr, 438
 - called by PARSE-Loop, 438
 - called by PARSE-Prefix, 415
 - called by PARSE-Primary1, 420

- calledby PARSE-Reduction, 417
- calledby PARSE-ScriptItem, 427
- calledby PARSE-Seg, 437
- calledby PARSE-Selector, 419
- calledby PARSE-SemiColon, 435
- calledby PARSE-Sequence1, 431
- calledby PARSE-Sexpr1, 428
- calledby PARSE-Statement, 408
- calledby PARSE-TokenCommandTail, 406
- calledby PARSE-VarForm, 426
- calls Pop-Reduction, 463
- calls reduction-value, 463
- calls stack-push, 463
- defmacro, 463
- pop-stack-3, 463
 - calledby PARSE-Category, 409
 - calledby PARSE-Conditional, 437
 - calledby PARSE-Float, 421
 - calledby PARSE-Iterator, 433
 - calls Pop-Reduction, 463
 - calls reduction-value, 463
 - calls stack-push, 463
 - defmacro, 463
- pop-stack-4, 463
 - calledby PARSE-Float, 421
 - calls Pop-Reduction, 463
 - calls reduction-value, 463
 - calls stack-push, 463
 - defmacro, 463
- postAdd, 358
 - calls postCapsule, 358
 - calls postTran, 358
 - defun, 358
- postAtom, 353
 - calledby postTran, 352
 - uses \$boot, 353
 - defun, 353
- postAtSign, 361
 - calls postTran, 361
 - calls postType, 361
 - defun, 361
- postBigFloat, 362
 - calls postTran, 362
 - uses \$InteractiveMode, 362
 - uses \$boot, 362
 - defun, 362
- postBlock, 362
 - calledby postSemiColon, 381
 - calls postBlockItemList, 362
 - calls postTran, 362
 - defun, 362
- postBlockItem, 360
 - calledby postBlockItemList, 359
 - calledby postCapsule, 359
 - calls postTran, 360
 - defun, 360
- postBlockItemList, 359
 - calledby postBlock, 362
 - calledby postCapsule, 359
 - calls postBlockItem, 359
 - defun, 359
- postCapsule, 359
 - calledby postAdd, 358
 - calls checkWarning, 359
 - calls postBlockItemList, 359
 - calls postBlockItem, 359
 - calls postFlatten, 359
 - defun, 359
- postCategory, 363
 - calls nreverse0, 363
 - calls postTran, 363
 - uses \$insidePostCategoryIfTrue, 363
 - defun, 363
- postcheck, 355
 - calledby postTransformCheck, 355
 - calledby postcheck, 355
 - calls postcheck, 355
 - calls setDefOp, 355
 - defun, 355
- postCollect, 365
 - calledby postCollect, 365
 - calledby postTupleCollect, 385
 - calls postCollect,finish, 365
 - calls postCollect, 365
 - calls postIteratorList, 365
 - calls postTran, 365
 - defun, 365
- postCollect,finish, 364
 - calledby postCollect, 365
 - calls postMakeCons, 364
 - calls postTranList, 364
 - calls qcar, 364

- calls qcdr, 364
- calls tuple2List, 364
- defun, 364
- postColon, 367
 - calls postTran, 367
 - calls postType, 367
 - defun, 367
- postColonColon, 367
 - calls postForm, 367
 - uses \$boot, 367
 - defun, 367
- postComma, 368
 - calls comma2Tuple, 368
 - calls postTuple, 368
 - defun, 368
- postConstruct, 369
 - calls comma2Tuple, 369
 - calls postMakeCons, 369
 - calls postTranList, 369
 - calls postTranSegment, 369
 - calls postTran, 369
 - calls tuple2List, 369
 - defun, 369
- postDef, 370
 - calls nequal, 371
 - calls nreverse0, 371
 - calls postDefArgs, 371
 - calls postMDef, 370
 - calls postTran, 371
 - calls recordHeaderDocumentation, 371
 - uses \$InteractiveMode, 371
 - uses \$boot, 371
 - uses \$docList, 371
 - uses \$headerDocumentation, 371
 - uses \$maxSignatureLineNumber, 371
 - defun, 370
- postDefArgs, 372
 - calledby postDefArgs, 372
 - calledby postDef, 371
 - calls postDefArgs, 372
 - calls postError, 372
 - defun, 372
- postError, 356
 - calledby checkWarning, 461
 - calledby getScriptName, 391
 - calledby parseInBy, 118
 - calledby parseIn, 117
 - calledby parseSeq, 128
 - calledby parseTranCheckForRecord, 457
 - calledby postDefArgs, 372
 - calledby postForm, 356
 - calls bumperrorcount, 356
 - calls nequal, 356
 - uses \$InteractiveMode, 356
 - uses \$defOp, 356
 - uses \$postStack, 356
 - defun, 356
- postExit, 373
 - calls postTran, 373
 - defun, 373
- postFlatten, 368
 - calledby comma2Tuple, 368
 - calledby postCapsule, 359
 - calledby postFlatten, 368
 - calls postFlatten, 368
 - defun, 368
- postFlattenLeft, 381
 - calledby postFlattenLeft, 381
 - calledby postSemiColon, 381
 - calls postFlattenLeft, 381
 - defun, 381
- postForm, 356
 - calledby postColonColon, 367
 - calledby postTran, 352
 - calls bright, 356
 - calls internl, 356
 - calls postError, 356
 - calls postTranList, 356
 - calls postTran, 356
 - uses \$boot, 356
 - defun, 356
- postIf, 373
 - calls nreverse0, 373
 - calls postTran, 373
 - uses \$boot, 373
 - defun, 373
- postIn, 375
 - calls postInSeq, 375
 - calls postTran, 375
 - calls systemErrorHere, 375
 - defun, 375
- postin, 374

- calls postInSeq, 374
 - calls postTran, 374
 - calls systemErrorHere, 374
 - defun, 374
- postInSeq, 374
 - calledby postIn, 375
 - calledby postIteratorList, 366
 - calledby postin, 374
 - calls postTranSegment, 374
 - calls postTran, 374
 - calls tuple2List, 374
 - defun, 374
- postIteratorList, 366
 - calledby postCollect, 365
 - calledby postIteratorList, 366
 - calledby postRepeat, 380
 - calls postInSeq, 366
 - calls postIteratorList, 366
 - calls postTran, 366
 - defun, 366
- postJoin, 376
 - calls postTranList, 376
 - calls postTran, 376
 - defun, 376
- postMakeCons, 364
 - calledby postCollect,finish, 364
 - calledby postConstruct, 369
 - calledby postMakeCons, 364
 - calls postMakeCons, 364
 - calls postTran, 364
 - defun, 364
- postMapping, 376
 - calls postTran, 376
 - calls unTuple, 376
 - defun, 376
- postMDef, 377
 - calledby postDef, 370
 - calls nreverse0, 377
 - calls postTran, 377
 - calls throwkeyedmsg, 377
 - uses \$InteractiveMode, 377
 - uses \$boot, 377
 - defun, 377
- postOp, 353
 - calledby postTran, 352
 - defun, 353
- postPretend, 378
 - calls postTran, 378
 - calls postType, 378
 - defun, 378
- postQUOTE, 379
 - defun, 379
- postReduce, 379
 - calledby postReduce, 379
 - calls postReduce, 379
 - calls postTran, 379
 - uses \$InteractiveMode, 379
 - defun, 379
- postRepeat, 380
 - calls postIteratorList, 380
 - calls postTran, 380
 - defun, 380
- postScripts, 381
 - calls getScriptName, 381
 - calls postTranScripts, 381
 - defun, 381
- postScriptsForm, 354
 - calledby postTran, 352
 - calls getScriptName, 354
 - calls length, 354
 - calls postTranScripts, 354
 - defun, 354
- postSemiColon, 381
 - calls postBlock, 381
 - calls postFlattenLeft, 381
 - defun, 381
- postSignature, 382
 - calls killColons, 382
 - calls postType, 382
 - calls removeSuperfluousMapping, 382
 - defun, 382
- postSlash, 383
 - calls postTran, 383
 - defun, 383
- postTran, 352
 - calledby postAdd, 358
 - calledby postAtSign, 361
 - calledby postBigFloat, 362
 - calledby postBlockItem, 360
 - calledby postBlock, 362
 - calledby postCategory, 363
 - calledby postCollect, 365

- calledby postColon, 367
- calledby postConstruct, 369
- calledby postDef, 371
- calledby postExit, 373
- calledby postForm, 356
- calledby postIf, 373
- calledby postInSeq, 374
- calledby postIn, 375
- calledby postIteratorList, 366
- calledby postJoin, 376
- calledby postMDef, 377
- calledby postMakeCons, 364
- calledby postMapping, 376
- calledby postPretend, 378
- calledby postReduce, 379
- calledby postRepeat, 380
- calledby postSlash, 383
- calledby postTranList, 354
- calledby postTranScripts, 354
- calledby postTranSegment, 370
- calledby postTransform, 351
- calledby postTran, 352
- calledby postType, 361
- calledby postWhere, 385
- calledby postWith, 386
- calledby postin, 374
- calledby tuple2List, 462
- calls postAtom, 352
- calls postForm, 352
- calls postOp, 352
- calls postScriptsForm, 352
- calls postTranList, 352
- calls postTran, 352
- calls qcar, 352
- calls qcdr, 352
- calls unTuple, 352
- defun, 352
- postTranList, 354
 - calledby postCollect,finish, 364
 - calledby postConstruct, 369
 - calledby postForm, 356
 - calledby postJoin, 376
 - calledby postTran, 352
 - calledby postTuple, 384
 - calledby postWhere, 385
 - calls postTran, 354
- defun, 354
- postTranScripts, 354
 - calledby postScriptsForm, 354
 - calledby postScripts, 381
 - calledby postTranScripts, 354
 - calls postTranScripts, 354
 - calls postTran, 354
 - defun, 354
- postTranSegment, 370
 - calledby postConstruct, 369
 - calledby postInSeq, 374
 - calledby tuple2List, 461
 - calls postTran, 370
 - defun, 370
- postTransform, 351
 - calledby new2OldLisp, 458
 - calledby s-process, 490
 - calls aplTran, 351
 - calls identp[5], 351
 - calls postTransformCheck, 351
 - calls postTran, 351
 - defun, 351
- postTransformCheck, 355
 - calledby postTransform, 351
 - calls postcheck, 355
 - uses \$defOp, 355
 - defun, 355
- postTuple, 384
 - calledby postComma, 368
 - calls postTranList, 384
 - defun, 384
- postTupleCollect, 385
 - calls postCollect, 385
 - defun, 385
- postType, 361
 - calledby postAtSign, 361
 - calledby postColon, 367
 - calledby postPretend, 378
 - calledby postSignature, 382
 - calls postTran, 361
 - calls unTuple, 361
 - defun, 361
- postWhere, 385
 - calls postTranList, 385
 - calls postTran, 385
 - defun, 385

- postWith, 386
 - calls postTran, 386
 - uses \$insidePostCategoryIfTrue, 386
 - defun, 386
- pp
 - calledby compDefineFunctor1, 181
 - calledby optimizeFunctionDef, 203
- PredImplies
 - calledby compForm2, 513
- prepare, 71, 76
 - calledby prepare, 76
 - calledby spad, 489
 - calls ifcar, 76
 - calls parseprint, 76
 - calls prepare1, 76
 - calls prepare, 76
 - uses \$comblocklist, 77
 - uses \$constructorLineNumber, 77
 - uses \$docList, 77
 - uses \$headerDocumentation, 77
 - uses \$index, 77
 - uses \$maxSignatureLineNumber, 77
 - uses \$prepare-last-line, 77
 - uses \$prepareReportIfTrue, 77
 - uses \$skipme, 77
 - defun, 76
- prepare-echo, 88
 - calledby fincomblock, 466
 - calledby prepare1, 81
 - uses Echo-Meta, 88
 - uses \$EchoLineStack, 88
 - defun, 88
- prepare1, 81
 - calledby prepare, 76
 - calls doSystemCommand[5], 81
 - calls escaped, 81
 - calls fincomblock, 81
 - calls indent-pos, 81
 - calls is-console, 81
 - calls make-full-cvec, 81
 - calls maxindex, 81
 - calls parsepiles, 81
 - calls prepare-echo, 81
 - calls prepareReadLine, 81
 - calls strposl[5], 81
 - local def \$index, 81
 - local def \$prepare-last-line, 81
 - local def \$skipme, 81
 - local ref \$byConstructors, 81
 - local ref \$constructorsSeen, 81
 - local ref \$echolinestack, 81
 - local ref \$in-stream, 81
 - local ref \$index, 81
 - local ref \$linelist, 81
 - local ref \$prepare-last-line, 81
 - catches, 81
 - defun, 81
- prepareReadLine, 85
 - calledby prepare1, 81
 - calledby prepareReadLine, 85
 - calledby skip-to-endif, 468
 - calls dcq, 85
 - calls initial-substring, 85
 - calls prepareReadLine1, 85
 - calls prepareReadLine, 85
 - calls skip-to-endif, 85
 - calls storeblanks, 85
 - calls string2BootTree, 85
 - local ref \$*eof*, 85
 - defun, 85
- prepareReadLine1, 87
 - calledby prepareReadLine1, 87
 - calledby prepareReadLine, 85
 - calledby skip-ifblock, 86
 - calledby skip-to-endif, 468
 - calls expand-tabs, 87
 - calls get-a-line, 87
 - calls maxindex, 87
 - calls prepareReadLine1, 87
 - calls strconc, 87
 - uses \$EchoLineStack, 87
 - uses \$index, 87
 - uses \$linelist, 87
 - uses \$prepare-last-line, 87
 - defun, 87
- pretend, 126, 310, 378
 - defplist, 126, 310, 378
- prettyprint
 - calledby optimize, 205
 - calledby s-process, 490
- PrimitivArray
 - calledby optCallEval, 213

- primitiveType, 504
 - calledby compAtom, 503
 - uses \$DoubleFloat, 504
 - uses \$EmptyMode, 505
 - uses \$NegativeInteger, 505
 - uses \$NonNegativeInteger, 505
 - uses \$PositiveInteger, 505
 - uses \$String, 505
 - defun, 504
- print-defun, 493
 - calls is-console, 493
 - calls print-full, 493
 - uses \$PrettyPrint, 493
 - uses vmlisp::optionlist, 493
 - defun, 493
- print-full
 - calledby print-defun, 493
- print-package, 461
 - local ref \$out-stream, 461
 - defun, 461
- printSignature
 - calledby getSignature, 288
- printStats
 - calledby compile, 146
- prior-token, 90
 - usedby PARSE-Expression, 411
 - uses \$token, 90
 - defvar, 90
- processFunctor, 251
 - calledby compCapsuleInner, 251
 - calls buildFunctor, 251
 - calls error, 251
 - defun, 251
- processInteractive[5]
 - called by s-process, 491
- processSynonyms[5]
 - called by PARSE-NewExpr, 403
- profileRecord
 - calledby compDefineCapsuleFunction, 280
 - calledby setqSingle, 326
- profileWrite
 - calledby finalizeLisplib, 174
- push-reduction, 454
 - calledby PARSE-AnyId, 430
 - calledby PARSE-Application, 418
 - calledby PARSE-Category, 409
 - calledby PARSE-CommandTail, 407
 - calledby PARSE-Command, 404
 - calledby PARSE-Conditional, 437
 - calledby PARSE-Data, 428
 - calledby PARSE-Enclosure, 424
 - calledby PARSE-Exit, 435
 - calledby PARSE-Expression, 411
 - calledby PARSE-Expr, 412
 - calledby PARSE-FloatBasePart, 423
 - calledby PARSE-FloatBase, 422
 - calledby PARSE-FloatExponent, 423
 - calledby PARSE-FloatTok, 439
 - calledby PARSE-Float, 421
 - calledby PARSE-Form, 417
 - calledby PARSE-Import, 411
 - calledby PARSE-InfixWith, 409
 - calledby PARSE-Infix, 415
 - calledby PARSE-LabelExpr, 438
 - calledby PARSE-Leave, 436
 - calledby PARSE-LedPart, 412
 - calledby PARSE-Loop, 438
 - calledby PARSE-Name, 427
 - calledby PARSE-NudPart, 412
 - calledby PARSE-OpenBrace, 432
 - calledby PARSE-OpenBracket, 432
 - calledby PARSE-Prefix, 414, 415
 - calledby PARSE-Primary1, 420
 - calledby PARSE-PrimaryOrQM, 407
 - calledby PARSE-Quad, 425
 - calledby PARSE-Qualification, 416
 - calledby PARSE-Reduction, 417
 - calledby PARSE-Return, 435
 - calledby PARSE-ScriptItem, 427
 - calledby PARSE-Seg, 436
 - calledby PARSE-Selector, 419
 - calledby PARSE-SemiColon, 435
 - calledby PARSE-Sequence1, 431
 - calledby PARSE-Sequence, 431
 - calledby PARSE-Sexpr1, 428
 - calledby PARSE-SpecialCommand, 405
 - calledby PARSE-Statement, 408
 - calledby PARSE-Suffix, 434
 - calledby PARSE-TokenCommandTail, 406
 - calledby PARSE-TokenList, 406
 - calledby PARSE-VarForm, 426
 - calledby PARSE-With, 409

- calledby parse-argument-designator, 460
- calledby parse-identifier, 459
- calledby parse-keyword, 460
- calledby parse-number, 460
- calledby parse-spadstring, 458
- calledby parse-string, 459
- calledby star, 453
- calls make-reduction, 454
- calls stack-push, 454
- uses reduce-stack, 454
- defun, 454
- put
 - calledby checkAndDeclare, 289
 - calledby compColon, 267
 - calledby compDefineCapsuleFunction, 280
 - calledby compMacro, 309
 - calledby compSubsetCategory, 333
 - calledby compSuchthat, 334
 - calledby compTypeOf, 502
 - calledby doIt, 253
 - calledby evalAndSub, 241
 - calledby getInverseEnvironment, 301
 - calledby getSuccessEnvironment, 300
 - calledby giveFormalParametersValues, 135
 - calledby putDomainsInScope, 230
 - calledby setqMultiple, 323
 - calledby updateCategoryFrameForCategory, 113
 - calledby updateCategoryFrameForConstructor, 112
- putDomainsInScope, 230
 - calledby addConstructorModemaps, 234
 - calledby augModemapsFromCategoryRep, 243
 - calledby augModemapsFromCategory, 237
 - calls delete, 230
 - calls getDomainsInScope, 230
 - calls member, 231
 - calls put, 230
 - calls say, 230
 - local def \$CapsuleDomainsInScope, 231
 - local ref \$insideCapsuleFunctionIfTrue, 231
 - defun, 230
- putInLocalDomainReferences, 154
 - calledby compile, 145
 - calls NRTputInTail, 154
 - local def \$elt, 154
 - local ref \$QuickCode, 154
 - defun, 154
- qcar
 - calledby TruthP, 241
 - calledby addArgumentConditions, 285
 - calledby addConstructorModemaps, 234
 - calledby addEltModemap, 237
 - calledby addModemap0, 246
 - calledby autoCoerceByModemap, 345
 - calledby canReturn, 298
 - calledby coerceByModemap, 345
 - calledby coerceExtraHard, 340
 - calledby compAdd, 248
 - calledby compCategoryItem, 263
 - calledby compCategory, 262
 - calledby compCons1, 271
 - calledby compDefWhereClause, 200
 - calledby compDefineFunctor1, 182
 - calledby compHasFormat, 295
 - calledby compJoin, 305
 - calledby compLambda, 307
 - calledby compMacro, 309
 - calledby compSetq1, 321
 - calledby compWithMappingMode1, 518
 - calledby compileCases, 283
 - calledby compile, 145
 - calledby decodeScripts, 391
 - calledby doIt, 253
 - calledby eltModemapFilter, 511
 - calledby encodeItem, 150
 - calledby fixUpPredicate, 160
 - calledby flattenSignatureList, 158
 - calledby genDomainView, 197
 - calledby getFormModemaps, 510
 - calledby getInverseEnvironment, 301
 - calledby getModemapList, 235
 - calledby getSignatureFromMode, 278
 - calledby getSignature, 288
 - calledby getSuccessEnvironment, 300
 - calledby interactiveModemapForm, 158
 - calledby isDomainConstructorForm, 330
 - calledby isDomainForm, 329
 - calledby isMacro, 259

- calledby makeFunctorArgumentParameters, 194
- calledby mkAlistOfExplicitCategoryOps, 156
- calledby mkEvalableCategoryForm, 140
- calledby mkNewModemapList, 238
- calledby mkOpVec, 199
- calledby mkRepititionAssoc, 149
- calledby mkUnion, 347
- calledby modemapPattern, 167
- calledby mustInstantiate, 266
- calledby optCallEval, 213
- calledby optCatch, 220
- calledby optCond, 222
- calledby optXLAMCond, 206
- calledby optimizeFunctionDef, 203
- calledby optimize, 205
- calledby orderPredTran, 162
- calledby orderPredicateItems, 161
- calledby outputComp, 328
- calledby parseHasRhs, 110
- calledby parseHas, 108
- calledby parseTranCheckForRecord, 457
- calledby postCollect,finish, 364
- calledby postTran, 352
- calledby replaceExitEtc, 319
- calledby setqMultiple, 322
- calledby spadCompileOrSetq, 151
- calledby stripOffArgumentConditions, 287
- calledby stripOffSubdomainConditions, 287
- calledby transIs1, 102
- calledby unknownTypeError, 229
- qcdr
 - calledby addArgumentConditions, 285
 - calledby addConstructorModemaps, 234
 - calledby addEltModemap, 237
 - calledby autoCoerceByModemap, 345
 - calledby canReturn, 298
 - calledby coerceByModemap, 345
 - calledby coerceExtraHard, 340
 - calledby compAdd, 248
 - calledby compCategoryItem, 263
 - calledby compCategory, 262
 - calledby compCons1, 271
 - calledby compDefWhereClause, 200
 - calledby compDefineFunctor1, 182
 - calledby compHasFormat, 295
 - calledby compJoin, 305
 - calledby compLambda, 307
 - calledby compSetq1, 321
 - calledby compWithMappingMode1, 518
 - calledby compileCases, 283
 - calledby compile, 145
 - calledby decodeScripts, 391
 - calledby doIt, 253
 - calledby eltModemapFilter, 511
 - calledby fixUpPredicate, 160
 - calledby flattenSignatureList, 158
 - calledby genDomainViewList, 196
 - calledby genDomainView, 197
 - calledby getFormModemaps, 510
 - calledby getInverseEnvironment, 301
 - calledby getModemapList, 236
 - calledby getSignatureFromMode, 278
 - calledby getSignature, 288
 - calledby getSuccessEnvironment, 300
 - calledby interactiveModemapForm, 158
 - calledby isDomainConstructorForm, 330
 - calledby isDomainForm, 329
 - calledby isMacro, 259
 - calledby makeFunctorArgumentParameters, 194
 - calledby mkAlistOfExplicitCategoryOps, 156
 - calledby mkEvalableCategoryForm, 140
 - calledby mkNewModemapList, 238
 - calledby mkOpVec, 199
 - calledby mkRepititionAssoc, 149
 - calledby mkUnion, 347
 - calledby modemapPattern, 167
 - calledby optCatch, 220
 - calledby optCond, 222
 - calledby optXLAMCond, 206
 - calledby optimizeFunctionDef, 203
 - calledby optimize, 205
 - calledby orderPredTran, 162
 - calledby orderPredicateItems, 161
 - calledby outputComp, 328
 - calledby parseHasRhs, 110
 - calledby parseHas, 108
 - calledby parseTranCheckForRecord, 457
 - calledby postCollect,finish, 364

- calledby postTran, 352
- calledby replaceExitEtc, 319
- calledby setqMultiple, 322
- calledby spadCompileOrSetq, 151
- calledby stripOffArgumentConditions, 287
- calledby stripOffSubdomainConditions, 287
- calledby transIs1, 102
- qslessp
 - calledby addDomain, 228
 - calledby getUniqueModemap, 235
- qsmminus, 217
 - defplist, 217
- quote, 311, 379
 - defplist, 311, 379
- quote-if-string, 442
 - calledby match-advance-string, 441
 - calledby unget-tokens, 444
 - calls escape-keywords, 442
 - calls pack, 442
 - calls strconc, 442
 - calls token-nonblank, 442
 - calls token-symbol, 442
 - calls token-type, 442
 - calls underscore, 442
 - uses \$boot, 442
 - uses \$spad, 442
 - defun, 442
- quotify
 - calledby compIf, 296
- rassoc
 - calledby modemapPattern, 167
- rbp
 - usedby PARSE-NudPart, 413
 - usedby PARSE-Operation, 413
- rdefiostream
 - calledby compileDocumentation, 172
 - calledby writeLib1, 174
- read-a-line, 533
 - calledby get-a-line, 540
 - calledby read-a-line, 533
 - calls Line-New-Line, 533
 - calls read-a-line, 533
 - calls subseq, 533
 - uses *eof*, 533
 - uses File-Closed, 533
- defun, 533
- recompile-lib-file-if-necessary, 529
 - calledby compileSpadLispCmd, 528
 - calls compile-lib-file, 529
 - uses *lisp-bin-filetype*, 529
- defun, 529
- Record, 261
 - defplist, 261
- recordAttributeDocumentation
 - calledby PARSE-Category, 410
- RecordCategory, 273
 - defplist, 273
- recordcopy, 227
 - defplist, 227
- recordelt, 225
 - defplist, 225
- recordHeaderDocumentation
 - calledby postDef, 371
- recordSignatureDocumentation
 - calledby PARSE-Category, 410
- reduce, 312, 379
 - defplist, 312, 379
- reduce-stack, 454
 - usedby push-reduction, 454
 - uses \$stack, 454
 - defvar, 454
- reduce-stack-clear, 454
 - defmacro, 454
- reduction, 92
 - defstruct, 92
- reduction-value
 - calledby nth-stack, 464
 - calledby pop-stack-1, 462
 - calledby pop-stack-2, 463
 - calledby pop-stack-3, 463
 - calledby pop-stack-4, 463
- refvecp
 - calledby translable1, 455
- remdup
 - calledby compDefineFunctor1, 182
 - calledby displayPreCompilationErrors, 456
 - calledby getSignature, 288
 - calledby hasSigInTargetCategory, 290
 - calledby mkAlistOfExplicitCategoryOps, 156
 - calledby mkExplicitCategoryFunction, 265

- calledby orderByDependency, 202
- removeEnv
 - calledby getSuccessEnvironment, 300
 - calledby setqSingle, 326
- removeSuperfluousMapping, 383
 - calledby postSignature, 382
 - defun, 383
- removeZeroOne
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby finalizeLisplib, 174
- repeat, 314, 380
 - defplist, 314, 380
- replaceExitEsc
 - calledby coerceExit, 342
- replaceExitEtc, 319
 - calledby compDefineCapsuleFunction, 280
 - calledby compSeq1, 318
 - calledby replaceExitEtc, 319
 - calls convertOrCroak, 319
 - calls intersectionEnvironment, 319
 - calls qcar, 319
 - calls qcdr, 319
 - calls replaceExitEtc, 319
 - calls rplac, 319
 - local def \$finalEnv, 319
 - local ref \$finalEnv, 319
 - defun, 319
- replaceFile
 - calledby compileDocumentation, 172
 - calledby lisplibDoRename, 172
- replaceVars, 159
 - calledby interactiveModemapForm, 158
 - calls msubst, 159
 - defun, 159
- reportOnFunctorCompilation, 192
 - calledby compDefineFunctor1, 182
 - calls addStats, 192
 - calls displayMissingFunctions, 192
 - calls displaySemanticErrors, 192
 - calls displayWarnings, 192
 - calls normalizeStatAndStringify, 192
 - calls sayBrightly, 192
 - uses \$functionStats, 193
 - uses \$functorStats, 193
 - uses \$op, 193
 - uses \$semanticErrorStack, 193
 - uses \$warningStack, 193
 - defun, 192
- resolve, 347
 - calledby coerceExit, 342
 - calledby compCategory, 262
 - calledby compCoerce1, 344
 - calledby compConstructorCategory, 274
 - calledby compDefineCapsuleFunction, 280
 - calledby compIf, 296
 - calledby compReturn, 317
 - calledby compString, 331
 - calledby convert, 504
 - calledby modifyModeStack, 525
 - calls mkUnion, 347
 - calls modeEqual, 347
 - calls nequal, 347
 - local ref \$EmptyMode, 347
 - local ref \$NoValueMode, 347
 - local ref \$String, 347
 - defun, 347
- return, 127, 316
 - defplist, 127, 316
- rpackfile
 - calledby compDefineLisplib, 170
 - calledby compileDocumentation, 172
- rplac
 - calledby coerce, 337
 - calledby getAbbreviation, 277
 - calledby optCall, 209
 - calledby optCatch, 220
 - calledby optCond, 222
 - calledby optSpecialCall, 212
 - calledby optXLAMCond, 206
 - calledby optimizeFunctionDef, 203
 - calledby optimize, 205
 - calledby replaceExitEtc, 319
- rplaca
 - calledby doItIf, 257
 - calledby optPackageCall, 210
 - calledby optSpecialCall, 212
- rplacd
 - calledby doItIf, 257
 - calledby optCond, 222
 - calledby optPackageCall, 210
- rplacw

- calledby optSpecialCall, 212
- rshut
 - calledby compDefineLisplib, 169
 - calledby compileDocumentation, 172
- rwrite
 - calledby compilerDoitWithScreenedLisplib, 349
- rwrite128
 - calledby lisplibWrite, 180
- rwriteLispForm, 168
 - calledby evalAndRwriteLispForm, 168
 - local ref \$libFile, 168
 - local ref \$lisplib, 168
 - defun, 168
- s-process, 490
 - calledby spad, 489
 - calls compTopLevel, 491
 - calls curstrm, 490
 - calls def-process, 491
 - calls def-rename, 490
 - calls displayPreCompilationErrors, 490
 - calls displaySemanticErrors, 491
 - calls get-internal-run-time, 491
 - calls new2OldLisp, 490
 - calls parseTransform, 490
 - calls postTransform, 490
 - calls prettyprint, 490
 - calls processInteractive[5], 491
 - calls terpri, 491
 - uses \$DomainFrame, 491
 - uses \$EmptyEnvironment, 491
 - uses \$EmptyMode, 491
 - uses \$Index, 491
 - uses \$InteractiveFrame, 491
 - uses \$LocalFrame, 491
 - uses \$PolyMode, 491
 - uses \$PrintOnly, 491
 - uses \$TranslateOnly, 491
 - uses \$Translation, 491
 - uses \$VariableCount, 491
 - uses \$compUniquelyIfTrue, 491
 - uses \$currentFunction, 491
 - uses \$currentLine, 491
 - uses \$exitModeStack, 491
 - uses \$exitMode, 491
- uses \$e, 491
- uses \$form, 491
- uses \$genFVar, 491
- uses \$genSDVar, 491
- uses \$insideCapsuleFunctionIfTrue, 491
- uses \$insideCategoryIfTrue, 491
- uses \$insideCoerceInteractiveHardIfTrue, 491
- uses \$insideExpressionIfTrue, 491
- uses \$insideFunctorIfTrue, 491
- uses \$insideWhereIfTrue, 491
- uses \$leaveLevelStack, 491
- uses \$leaveMode, 491
- uses \$macroassoc, 491
- uses \$newspad, 491
- uses \$postStack, 491
- uses \$previousTime, 491
- uses \$returnMode, 491
- uses \$semanticErrorStack, 491
- uses \$top-level, 491
- uses \$topOp, 491
- uses \$warningStack, 491
- uses curoutstream, 491
- defun, 490
- say
 - calledby canReturn, 298
 - calledby compOrCroak1, 497
 - calledby getSignature, 288
 - calledby modifyModeStack, 525
 - calledby optimize, 205
 - calledby orderByDependency, 202
 - calledby putDomainsInScope, 230
- sayBrightly
 - calledby checkAndDeclare, 289
 - calledby compDefineCapsuleFunction, 280
 - calledby compDefineFunctor1, 181
 - calledby compMacro, 309
 - calledby compilerDoit, 478
 - calledby compile, 145
 - calledby displayMissingFunctions, 193
 - calledby displayPreCompilationErrors, 456
 - calledby doIt, 253
 - calledby reportOnFunctorCompilation, 192
 - calledby spadCompileOrSetq, 152
- sayBrightlyI
 - calledby optimizeFunctionDef, 203

- sayKeyedMsg[5]
 - called by compileSpad2Cmd, 476
 - called by compileSpadLispCmd, 528
- sayMath
 - calledby displayPreCompilationErrors, 456
- sayMSG
 - calledby compDefineLisplib, 169
 - calledby finalizeLisplib, 174
- ScanOrPairVec[5]
 - called by hasFormalMapVariable, 524
- Scripts, 380
 - defplist, 380
- segment, 127, 128
 - defplist, 127, 128
- selectOptionLC[5]
 - called by compileSpad2Cmd, 476
 - called by compileSpadLispCmd, 528
 - called by compiler, 473
- seq, 214, 318
 - defplist, 214, 318
- setDefOp, 386
 - calledby parseDEF, 101
 - calledby postcheck, 355
 - uses \$defOp, 386
 - uses \$topOp, 386
 - defun, 386
- setdifference
 - calledby orderPredTran, 162
- setelt
 - calledby modifyModeStack, 525
- seteltModemapFilter, 512
 - calls isConstantId, 512
 - calls stackMessage, 512
 - defun, 512
- setq, 321
 - defplist, 321
- setqMultiple, 322
 - calledby compSetq1, 321
 - calls addBinding, 322
 - calls compSetq1, 322
 - calls convert, 323
 - calls genSomeVariable, 323
 - calls genVariable, 322
 - calls length, 323
 - calls mkprogn, 323
 - calls nreverse0, 322
 - calls put, 323
 - calls qcar, 322
 - calls qcdr, 322
 - calls setqMultipleExplicit, 322
 - calls stackMessage, 322
 - local ref \$EmptyMode, 323
 - local ref \$NoValueMode, 323
 - local ref \$noEnv, 323
 - defun, 322
- setqMultipleExplicit, 324
 - calledby setqMultiple, 322
 - calls compSetq1, 324
 - calls genVariable, 324
 - calls last, 324
 - calls nequal, 324
 - calls stackMessage, 324
 - local ref \$EmptyMode, 325
 - local ref \$NoValueMode, 325
 - defun, 324
- setqSetelt, 326
 - calledby compSetq1, 321
 - calls comp, 326
 - defun, 326
- setqSingle, 326
 - calledby compSetq1, 321
 - calls NRTassocIndex, 326
 - calls addBinding[5], 326
 - calls assignError, 326
 - calls augModemapsFromDomain1, 326
 - calls comp, 326
 - calls consProplistOf, 326
 - calls convert, 326
 - calls getProplist[5], 326
 - calls getmode, 326
 - calls get, 326
 - calls identp[5], 326
 - calls isDomainForm, 326
 - calls isDomainInScope, 326
 - calls maxSuperType, 326
 - calls nequal, 326
 - calls outputComp, 326
 - calls profileRecord, 326
 - calls removeEnv, 326
 - calls stackWarning, 326
 - uses \$EmptyMode, 326
 - uses \$NoValueMode, 326

- uses \$QuickLet, 326
 - uses \$form, 326
 - uses \$insideSetqSingleIfTrue, 326
 - uses \$profileCompiler, 326
- defun, 326
- setrecordelt, 226
 - defplist, 226
- shut[5]
 - called by spad, 489
- Signature, 382
 - defplist, 382
- signatureTran, 161
 - calledby orderPredicateItems, 161
 - calledby signatureTran, 161
 - calls isCategoryForm, 161
 - calls signatureTran, 161
 - local ref \$e, 161
 - defun, 161
- simpBool
 - calledby compDefineFunctor1, 182
- skip-blanks, 440
 - calledby match-string, 440
 - calls advance-char, 440
 - calls current-char, 440
 - calls token-lookahead-type, 440
 - defun, 440
- skip-iffblock, 86
 - calledby skip-iffblock, 86
 - calls initial-substring, 86
 - calls preparseReadLine1, 86
 - calls skip-iffblock, 86
 - calls storeblanks, 86
 - calls string2BootTree, 86
 - defun, 86
- skip-to-endif, 468
 - calledby preparseReadLine, 85
 - calledby skip-to-endif, 468
 - calls initial-substring, 468
 - calls preparseReadLine1, 468
 - calls preparseReadLine, 468
 - calls skip-to-endif, 468
 - defun, 468
- SourceLevelSubsume
 - calledby getSignature, 288
- spad, 489
 - calls PARSE-NewExpr, 489
 - calls addBinding[5], 489
 - calls init-boot/spad-reader[5], 489
 - calls initialize-preparse, 489
 - calls ioclear, 489
 - calls makeInitialModemapFrame[5], 489
 - calls pop-stack-1, 489
 - calls preparse, 489
 - calls s-process, 489
 - calls shut[5], 489
 - uses *comp370-apply*, 489
 - uses *eof*, 489
 - uses *fileactq-apply*, 489
 - uses /editfile, 489
 - uses \$InitialDomainsInScope, 489
 - uses \$InteractiveFrame, 489
 - uses \$InteractiveMode, 489
 - uses \$boot, 489
 - uses \$noSubsumption, 489
 - uses \$spad, 489
 - uses boot-line-stack, 489
 - uses curoutstream, 489
 - uses echo-meta, 489
 - uses file-closed, 489
 - uses line, 489
 - uses optionlist, 489
 - catches, 489
 - defun, 489
- spad-fixed-arg, 529
 - defun, 529
- spad2AsTranslatorAutoloadOnceTrigger
 - calledby compileSpad2Cmd, 476
- spad[5]
 - called by /rf-1, 480
- spadcall, 219
 - defplist, 219
- spadCompileOrSetq, 151
 - calledby compile, 146
 - calls LAM,EVALANDFILEACTQ, 152
 - calls bright, 152
 - calls compileConstructor, 152
 - calls comp, 152
 - calls contained, 152
 - calls mkq, 152
 - calls qcar, 151
 - calls qcdr, 151
 - calls sayBrightly, 152

- local ref \$insideCapsuleFunctionIfTrue, 152
 - defun, 151
- SpadInterpretStream[5]
 - called by ncINTERPFILE, 527
- spadPrompt
 - calledby compileSpad2Cmd, 476
 - calledby compileSpadLispCmd, 528
- spadreduce
 - calledby floatexpid, 451
- splitEncodedFunctionName, 149
 - calledby compile, 145
 - calls stringimage, 149
 - calls strpos, 149
 - defun, 149
- stack, 88
 - defstruct, 88
- stack-/empty, 89
 - uses \$stack, 89
 - defmacro, 89
- stack-clear, 89
 - uses \$stack, 89
 - defun, 89
- stack-load, 89
 - uses \$stack, 89
 - defun, 89
- stack-pop, 90
 - calledby Pop-Reduction, 464
 - uses \$stack, 90
 - defun, 90
- stack-push, 89
 - calledby pop-stack-2, 463
 - calledby pop-stack-3, 463
 - calledby pop-stack-4, 463
 - calledby push-reduction, 454
 - uses \$stack, 89
 - defun, 89
- stack-size
 - calledby star, 453
- stack-store
 - calledby nth-stack, 464
- stackAndThrow
 - calledby compDefine1, 275
 - calledby compInternalFunction, 279
 - calledby compLambda, 307
 - calledby compWithMappingMode1, 518
 - calledby getSignatureFromMode, 279
- stackMessage
 - calledby assignError, 328
 - calledby augModemapsFromDomain1, 233
 - calledby autoCoerceByModemap, 346
 - calledby coerce, 337
 - calledby compElt, 292
 - calledby compRepeatOrCollect, 315
 - calledby compSymbol, 505
 - calledby eltModemapFilter, 511
 - calledby getFormModemaps, 510
 - calledby getOperationAlist, 242
 - calledby seteltModemapFilter, 512
 - calledby setqMultipleExplicit, 324
 - calledby setqMultiple, 322
- stackMessageIfNone
 - calledby compExit, 293
 - calledby compForm, 507
- stackSemanticError
 - calledby compColonInside, 502
 - calledby compJoin, 305
 - calledby compOrCroak1, 497
 - calledby compPretend, 310
 - calledby compReturn, 317
 - calledby compSubDomain1, 332
 - calledby constructMacro, 151
 - calledby doIt, 253
 - calledby getArgumentModeOrMoan, 155
 - calledby getSignature, 288
 - calledby getTargetFromRhs, 135
 - calledby unknownTypeError, 229
- stackWarning
 - calledby compColonInside, 502
 - calledby compElt, 292
 - calledby compPretend, 310
 - calledby doIt, 253
 - calledby getUniqueModemap, 235
 - calledby hasSigInTargetCategory, 290
 - calledby setqSingle, 326
- star, 453
 - calledby PARSE-Application, 418
 - calledby PARSE-Category, 410
 - calledby PARSE-CommandTail, 407
 - calledby PARSE-Expr, 412
 - calledby PARSE-Import, 411
 - calledby PARSE-IteratorTail, 433

- calledby PARSE-Loop, 438
- calledby PARSE-Option, 408
- calledby PARSE-ScriptItem, 427
- calledby PARSE-Sexpr1, 429
- calledby PARSE-SpecialCommand, 405
- calledby PARSE-Statement, 408
- calledby PARSE-TokenCommandTail, 406
- calledby PARSE-TokenList, 406
- calls pop-stack-1, 453
- calls push-reduction, 453
- calls stack-size, 453
- defmacro, 453
- step
 - calledby floatexpid, 451
- storeblanks, 539
 - calledby prepareReadLine, 85
 - calledby skip-ifblock, 86
 - defun, 539
- strconc
 - calledby compDefine1, 275
 - calledby compDefineFunctor1, 181
 - calledby compileSpad2Cmd, 476
 - calledby compile, 145
 - calledby decodeScripts, 391
 - calledby getCaps, 150
 - calledby mkCategoryPackage, 139
 - calledby prepareReadLine1, 87
 - calledby quote-if-string, 442
 - calledby unget-tokens, 444
- String, 330
 - defplist, 330
- string-not-greaterp
 - calledby initial-substring-p, 442
- string2BootTree
 - calledby prepareReadLine, 85
 - calledby skip-ifblock, 86
- string2id-n
 - calledby infixtok, 467
- stringimage
 - calledby encodeFunctionName, 148
 - calledby encodeItem, 150
 - calledby getCaps, 150
 - calledby splitEncodedFunctionName, 149
 - calledby substituteCategoryArguments, 233
- stringPrefix?
 - calledby comp3, 500
- stripOffArgumentConditions, 287
 - calledby compDefineCapsuleFunction, 280
 - calls msubst, 287
 - calls qcar, 287
 - calls qcdr, 287
 - local def \$argumentConditionList, 288
 - local ref \$argumentConditionList, 287
 - defun, 287
- stripOffSubdomainConditions, 287
 - calledby compDefineCapsuleFunction, 280
 - calls assoc, 287
 - calls mkpf, 287
 - calls qcar, 287
 - calls qcdr, 287
 - local def \$argumentConditionList, 287
 - local ref \$argumentConditionList, 287
 - defun, 287
- stripUnionTags
 - calledby augModemapsFromDomain, 232
- stros
 - calledby splitEncodedFunctionName, 149
- strospl[5]
 - called by prepare1, 81
- SubDomain, 331
 - defplist, 331
- sublis
 - calledby augLisplibModemapsFromCategory, 155
 - calledby coerceable, 342
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 182
 - calledby compForm2, 513
 - calledby doIt, 253
 - calledby formal2Pattern, 190
 - calledby getModemap, 234
 - calledby mkOpVec, 199
 - calledby substituteCategoryArguments, 233
- sublislis
 - calledby compHasFormat, 295
 - calledby mkCategoryPackage, 139
- subrname, 208
 - calledby optimize, 205
 - calls bpiname, 208
 - calls compiled-function-p, 208
 - calls identp, 208
 - calls mbpip, 208

- defun, 208
- subseq
 - calledby match-string, 440
 - calledby read-a-line, 533
- SubsetCategory, 333
 - defplist, 333
- substituteCategoryArguments, 233
 - calledby augModemapsFromDomain1, 233
 - calls internl, 233
 - calls msubst, 233
 - calls stringimage, 233
 - calls sublis, 233
 - defun, 233
- substNames, 243
 - calledby evalAndSub, 241
 - calledby genDomainOps, 198
 - calls eqsubstlist, 243
 - calls isCategoryPackageName, 243
 - calls nreverse0, 243
 - calls substq, 243
 - uses \$FormalMapVariableList, 243
 - defun, 243
- substq
 - calledby substNames, 243
- substVars, 166
 - calledby interactiveModemapForm, 158
 - calls contained, 166
 - calls msubst, 166
 - calls nsubst, 166
 - local ref \$FormalMapVariableList, 166
 - defun, 166
- suffix
 - calledby addclose, 464
- systemCommand[5]
 - called by PARSE-CommandTail, 407
 - called by PARSE-TokenCommandTail, 406
- systemError
 - calledby compReduce1, 312
 - calledby errhuh, 395
 - calledby getOperationAlist, 242
- systemErrorHere
 - calledby addArgumentConditions, 285
 - calledby addEltModemap, 237
 - calledby canReturn, 298
 - calledby compCategory, 262
 - calledby compColon, 267
 - calledby getSlotFromCategoryForm, 177
 - calledby getSlotFromFunctor, 179
 - calledby optCall, 209
 - calledby postIn, 375
 - calledby postin, 374
- take
 - calledby compColon, 267
 - calledby compDefineCategory2, 142
 - calledby compForm2, 513
 - calledby compHasFormat, 295
 - calledby compWithMappingMode1, 518
 - calledby drop, 465
 - calledby getSignatureFromMode, 279
 - calledby getSlotFromCategoryForm, 177
- terminateSystemCommand[5]
 - called by compileSpad2Cmd, 476
 - called by compileSpadLispCmd, 528
- terpri
 - calledby s-process, 491
- throwKeyedMsg
 - calledby compileSpad2Cmd, 476
 - calledby compileSpadLispCmd, 528
 - calledby compiler, 473
 - calledby loadLibIfNecessary, 111
- throwkeyedmsg
 - calledby postMDef, 377
- throws
 - compForm3, 515
- tmptok, 402
 - usedby PARSE-Operation, 413
 - defvar, 402
- tok, 402
 - usedby PARSE-GlyphTok, 430
 - usedby PARSE-NBGlyphTok, 429
 - defvar, 402
- token, 90
 - defstruct, 90
- token-install, 92
 - uses \$token, 92
 - defun, 92
- token-lookahead-type, 441
 - calledby skip-blanks, 440
 - uses Escape-Character, 441
 - defun, 441
- token-nonblank

- calledby PARSE-FloatBasePart, 423
 - calledby quote-if-string, 442
 - calledby unget-tokens, 445
- token-print, 92
 - uses \$token, 92
 - defun, 92
- token-symbol
 - calledby PARSE-SpecialKeyword, 404
 - calledby match-token, 446
 - calledby parse-argument-designator, 460
 - calledby parse-identifier, 459
 - calledby parse-keyword, 460
 - calledby parse-number, 460
 - calledby parse-spadstring, 458
 - calledby parse-string, 459
 - calledby quote-if-string, 442
- token-type
 - calledby match-token, 446
 - calledby quote-if-string, 442
- TPDHERE
 - See LocalAlgebra for an example call, 333
 - The use of and in spadreduce is undefined. rewrite this to loop, 451
 - test with BASTYPE, 134
- transformOperationAlist, 177
 - calledby getCategoryOpsAndAtts, 177
 - calledby getFunctorOpsAndAtts, 179
 - calls assoc, 178
 - calls insertAlist, 178
 - calls keyedSystemError, 178
 - calls lassq, 178
 - calls member, 178
 - local ref \$functionLocations, 178
 - defun, 177
- transIs, 102
 - calledby parseIsnt, 120
 - calledby parseIs, 119
 - calledby parseLET, 122
 - calledby parseLhs, 102
 - calledby transIs1, 102
 - calls isListConstructor, 102
 - calls transIs1, 102
 - defun, 102
- transIs1, 102
 - calledby transIs1, 102
- calledby transIs, 102
 - calls nreverse0, 102
 - calls qcar, 102
 - calls qcdr, 102
 - calls transIs1, 102
 - calls transIs, 102
 - defun, 102
- translabel, 455
 - calledby PARSE-Data, 428
 - calls translabel1, 455
 - defun, 455
- translabel1, 455
 - calledby translabel1, 455
 - calledby translabel, 455
 - calls lassoc, 455
 - calls maxindex, 455
 - calls refvecp, 455
 - calls translabel1, 455
 - defun, 455
- transSeq
 - calledby parseSeq, 128
- TruthP, 241
 - calledby mergeModemap, 239
 - calledby optCond, 222
 - calls qcar, 241
 - defun, 241
- try-get-token, 447
 - calledby advance-token, 448
 - calledby current-token, 447
 - calledby next-token, 448
 - calls get-token, 447
 - uses valid-tokens, 447
 - defun, 447
- tuple2List, 461
 - calledby postCollect,finish, 364
 - calledby postConstruct, 369
 - calledby postInSeq, 374
 - calledby tuple2List, 461
 - calls postTranSegment, 461
 - calls postTran, 462
 - calls tuple2List, 461
 - uses \$InteractiveMode, 462
 - uses \$boot, 462
 - defun, 461
- TupleCollect, 384
 - defplist, 384

- unabbrevAndLoad
 - calledby parseHasRhs, 110
 - calledby parseHas, 108
- unAbbreviateKeyword[5]
 - called by PARSE-SpecialKeyWord, 404
- uncons, 322
 - calledby uncons, 322
 - calls uncons, 322
 - defun, 322
- Undef
 - usedby mkOpVec, 199
- underscore, 444
 - calledby quote-if-string, 442
 - calls vector-push, 444
 - defun, 444
- unembed
 - calledby compilerDoitWithScreenedLisplib, 349
- unErrorRef
 - calledby addModemap1, 246
- unget-tokens, 444
 - calledby match-string, 440
 - calls line-current-segment, 444
 - calls line-new-line, 445
 - calls line-number, 445
 - calls quote-if-string, 444
 - calls strconc, 444
 - calls token-nonblank, 445
 - uses valid-tokens, 445
 - defun, 444
- Union, 261
 - defplist, 261
- union
 - calledby compDefWhereClause, 200
 - calledby compJoin, 305
 - calledby makeFunctorArgumentParameters, 194
 - calledby mkAlistOfExplicitCategoryOps, 156
 - calledby mkExplicitCategoryFunction, 265
 - calledby mkUnion, 347
- UnionCategory, 273
 - defplist, 273
- unionq
 - calledby freelist, 526
 - calledby orderPredTran, 162
- unknownTypeError, 229
 - calledby addDomain, 228
 - calledby compColon, 267
 - calls qcar, 229
 - calls stackSemanticError, 229
 - defun, 229
- unloadOneConstructor
 - calledby compDefineLisplib, 170
- unTuple, 395
 - calledby postMapping, 376
 - calledby postTran, 352
 - calledby postType, 361
 - defun, 395
- updateCategoryFrameForCategory, 113
 - calledby compDefineLisplib, 170
 - calledby isFunctor, 229
 - calledby loadLibIfNecessary, 111
 - calls addModemap, 113
 - calls getdatabase, 113
 - calls put, 113
 - local def \$CategoryFrame, 113
 - local ref \$CategoryFrame, 113
 - defun, 113
- updateCategoryFrameForConstructor, 112
 - calledby compDefineLisplib, 170
 - calledby isFunctor, 229
 - calledby loadLibIfNecessary, 111
 - calls addModemap, 112
 - calls convertOpAlist2compilerInfo, 112
 - calls getdatabase, 112
 - calls put, 112
 - local def \$CategoryFrame, 112
 - local ref \$CategoryFrame, 112
 - defun, 112
- updateSourceFiles[5]
 - called by compileSpad2Cmd, 476
- userError
 - calledby compDefWhereClause, 200
 - calledby compOrCroak1, 497
 - calledby compReturn, 317
 - calledby compile, 145
 - calledby convertOrCroak, 320
 - calledby doItIf, 257
 - calledby orderByDependency, 202
- valid-tokens, 91

- usedby advance-token, 448
- usedby current-token, 447
- usedby next-token, 448
- usedby try-get-token, 447
- usedby unget-tokens, 445
- uses \$token, 91
- defvar, 91
- vcons, 129
 - defplist, 129
- Vector
 - calledby optCallEval, 213
- vector, 335
 - defplist, 335
- vector-push
 - calledby underscore, 444
- VectorCategory, 273
 - defplist, 273
- vmlisp::optionlist
 - usedby print-defun, 493
- where, 129, 336, 385
 - defplist, 129, 336, 385
- with, 386
 - defplist, 386
- wrapDomainSub, 266
 - calledby compJoin, 305
 - calledby mkExplicitCategoryFunction, 265
 - defun, 266
- wrapSEQExit
 - calledby makeSimplePredicateOrNil, 458
- writeLib1, 174
 - calledby initializeLisplib, 173
 - calls rdefiostream, 174
 - defun, 174
- XTokenReader, 449
 - calledby get-token, 449
 - usedby get-token, 449
 - defvar, 449