

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
Yurij 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	defvar \$current-fragment	88
3.4.3	defun read-a-line	88
3.5	Line Handling	89
3.5.1	Line Buffer	89
3.5.2	defstruct \$line	89
3.5.3	defvar \$current-line	90
3.5.4	defmacro line-clear	90
3.5.5	defun line-print	90
3.5.6	defun line-at-end-p	90
3.5.7	defun line-past-end-p	91
3.5.8	defun line-next-char	91
3.5.9	defun line-advance-char	91
3.5.10	defun line-current-segment	92
3.5.11	defun line-new-line	92
3.5.12	defun next-line	92
3.5.13	defun Advance-Char	93
3.5.14	defun storeblanks	93
3.5.15	defun initial-substring	93
3.5.16	defun get-a-line	94
3.5.17	defun make-string-adjustable	94

3.5.18	Parsing stack	94
3.5.19	defstruct \$stack	94
3.5.20	defun stack-load	95
3.5.21	defun stack-clear	95
3.5.22	defmacro stack-/empty	95
3.5.23	defun stack-push	96
3.5.24	defun stack-pop	96
3.5.25	Parsing token	96
3.5.26	defstruct \$token	96
3.5.27	defvar \$prior-token	97
3.5.28	defvar \$nonblank	97
3.5.29	defvar \$current-token	97
3.5.30	defvar \$next-token	97
3.5.31	defvar \$valid-tokens	98
3.5.32	defun token-install	98
3.5.33	defun token-print	98
3.5.34	Parsing reduction	98
3.5.35	defstruct \$reduction	98
4	Parse Transformers	101
4.1	Direct called parse routines	101
4.1.1	defun parseTransform	101
4.1.2	defun parseTran	101
4.1.3	defun parseAtom	102
4.1.4	defun parseTranList	103
4.1.5	defun parseConstruct	103
4.1.6	defun parseConstruct	103
4.2	Indirect called parse routines	104
4.2.1	defun parseAnd	105
4.2.2	defun parseAnd	105
4.2.3	defun parseAtSign	105
4.2.4	defun parseAtSign	106
4.2.5	defun parseType	106
4.2.6	defun parseCategory	106
4.2.7	defun parseCategory	107
4.2.8	defun parseDropAssertions	107
4.2.9	defun parseCoerce	107
4.2.10	defun parseCoerce	108
4.2.11	defun parseColon	108
4.2.12	defun parseColon	108
4.2.13	defun parseDEF	109
4.2.14	defun parseDEF	109
4.2.15	defun parseLhs	110
4.2.16	defun transIs	110
4.2.17	defun transIs1	110
4.2.18	defun isListConstructor	111

4.2.19	defun parseDollarGreaterthan	112
4.2.20	defun parseDollarGreaterThan	112
4.2.21	defun parseDollarGreaterEqual	112
4.2.22	defun parseDollarGreaterEqual	112
4.2.23	defun parseDollarLessEqual	113
4.2.24	defun parseDollarNotEqual	113
4.2.25	defun parseDollarNotEqual	113
4.2.26	defun parseEquivalence	114
4.2.27	defun parseEquivalence	114
4.2.28	defun parseExit	114
4.2.29	defun parseExit	115
4.2.30	defun parseGreaterEqual	115
4.2.31	defun parseGreaterEqual	115
4.2.32	defun parseGreaterThan	116
4.2.33	defun parseGreaterThan	116
4.2.34	defun parseHas	116
4.2.35	defun parseHas	116
4.2.36	defun parseHasRhs	118
4.2.37	defun loadIfNecessary	119
4.2.38	defun loadLibIfNecessary	119
4.2.39	defun updateCategoryFrameForConstructor	120
4.2.40	defun convertOpAlist2compilerInfo	120
4.2.41	defun updateCategoryFrameForCategory	121
4.2.42	defun parseIf	121
4.2.43	defun parseIf	122
4.2.44	defun parseIf,ifTran	122
4.2.45	defun parseImplies	124
4.2.46	defun parseImplies	124
4.2.47	defun parseIn	125
4.2.48	defun parseIn	125
4.2.49	defun parseInBy	126
4.2.50	defun parseInBy	126
4.2.51	defun parseIs	127
4.2.52	defun parseIs	127
4.2.53	defun parseIsnt	127
4.2.54	defun parseIsnt	128
4.2.55	defun parseJoin	128
4.2.56	defun parseJoin	128
4.2.57	defun parseLeave	129
4.2.58	defun parseLeave	129
4.2.59	defun parseLessEqual	129
4.2.60	defun parseLessEqual	130
4.2.61	defun parseLET	130
4.2.62	defun parseLET	130
4.2.63	defun parseLETD	131
4.2.64	defun parseLETD	131

4.2.65	defun parseMDEF	131
4.2.66	defun parseMDEF	131
4.2.67	defun parseNot	132
4.2.68	defun parseNot	132
4.2.69	defun parseNot	132
4.2.70	defun parseNotEqual	133
4.2.71	defun parseNotEqual	133
4.2.72	defun parseOr	133
4.2.73	defun parseOr	133
4.2.74	defun parsePretend	134
4.2.75	defun parsePretend	134
4.2.76	defun parseReturn	135
4.2.77	defun parseReturn	135
4.2.78	defun parseSegment	135
4.2.79	defun parseSegment	136
4.2.80	defun parseSeq	136
4.2.81	defun parseSeq	136
4.2.82	defun parseVCONS	137
4.2.83	defun parseVCONS	137
4.2.84	defun parseWhere	137
4.2.85	defun parseWhere	137
5	Compile Transformers	139
5.1	Routines for handling forms	139
5.2	Functions which handle == statements	141
5.2.1	defun compDefineAddSignature	141
5.2.2	defun hasFullSignature	141
5.2.3	defun addEmptyCapsuleIfNecessary	142
5.2.4	defun getTargetFromRhs	142
5.2.5	defun giveFormalParametersValues	143
5.2.6	defun macroExpandInPlace	143
5.2.7	defun macroExpand	144
5.2.8	defun macroExpandList	144
5.2.9	defun compDefineCategory1	145
5.2.10	defun makeCategoryPredicates	146
5.2.11	defun mkCategoryPackage	146
5.2.12	defun mkEvaluableCategoryForm	148
5.2.13	defun compDefineCategory2	149
5.2.14	defun augLisplibModemapsFromCategory	152
5.2.15	defun evalAndRwriteLispForm	154
5.2.16	defun rwriteLispForm	154
5.2.17	defun mkConstructor	154
5.2.18	defun compDefineCategory	155
5.2.19	defun compDefineLisplib	155
5.2.20	defun compileDocumentation	158
5.2.21	defun lisplibDoRename	158

5.2.22	defun initializeLisplib	159
5.2.23	defun writeLib1	160
5.2.24	defun finalizeLisplib	160
5.2.25	defun getConstructorOpsAndAtts	162
5.2.26	defun getCategoryOpsAndAtts	162
5.2.27	defun getSlotFromCategoryForm	163
5.2.28	defun transformOperationAlist	163
5.2.29	defun getFunctorOpsAndAtts	165
5.2.30	defun getSlotFromFunctor	165
5.2.31	defun mergeSignatureAndLocalVarAlists	165
5.2.32	defun lisplibWrite	166
5.2.33	defun compDefineFunctor	166
5.2.34	defun compDefineFunctor1	167
5.2.35	defun augmentLisplibModemapsFromFunctor	174
5.2.36	defun disallowNilAttribute	176
5.2.37	defun compFunctorBody	176
5.2.38	defun reportOnFunctorCompilation	177
5.2.39	defun displayMissingFunctions	177
5.2.40	defun makeFunctorArgumentParameters	178
5.2.41	defun genDomainViewList0	180
5.2.42	defun genDomainViewList	181
5.2.43	defun genDomainView	181
5.2.44	defun genDomainOps	182
5.2.45	defun mkOpVec	183
5.2.46	defun compDefWhereClause	184
5.3	Functions to manipulate modemaps	186
5.3.1	defun addDomain	186
5.3.2	defun isFunctor	188
5.3.3	defun getDomainsInScope	188
5.3.4	defun putDomainsInScope	189
5.3.5	defun isSuperDomain	189
5.3.6	defun addNewDomain	190
5.3.7	defun augModemapsFromDomain	190
5.3.8	defun augModemapsFromDomain1	191
5.3.9	defun substituteCategoryArguments	192
5.3.10	defun addConstructorModemaps	192
5.3.11	defun getModemap	193
5.3.12	defun getUniqueSignature	193
5.3.13	defun getUniqueModemap	194
5.3.14	defun getModemapList	194
5.3.15	defun getModemapListFromDomain	195
5.3.16	defun domainMember	195
5.3.17	defun augModemapsFromCategory	195
5.3.18	defun addModemapKnown	196
5.3.19	defun addModemap0	196
5.3.20	defun addEltModemap	197

5.3.21	defun addModemap1	198
5.3.22	defun mkNewModemapList	198
5.3.23	defun mergeModemap	199
5.3.24	defun evalAndSub	201
5.3.25	defun getOperationAlist	201
5.3.26	defvar \$FormalMapVariableList	202
5.3.27	defun substNames	202
5.3.28	defun augModemapsFromCategoryRep	203
5.4	Indirect called comp routines	204
5.4.1	defun compAdd plist	204
5.4.2	defun compAdd	205
5.4.3	defun compAtSign plist	207
5.4.4	defun compAtSign	207
5.4.5	defun compCapsule plist	207
5.4.6	defun compCapsule	208
5.4.7	defun compCapsuleInner	208
5.4.8	defun compCase plist	209
5.4.9	defun compCase	209
5.4.10	defun compCase1	210
5.4.11	defun compCat plist	211
5.4.12	defun compCat plist	211
5.4.13	defun compCat plist	211
5.4.14	defun compCat	211
5.4.15	defun compCategory plist	212
5.4.16	defun compCategory	212
5.4.17	defun compCoerce plist	213
5.4.18	defun compCoerce	213
5.4.19	defun compCoerce1	214
5.4.20	defun compColon plist	215
5.4.21	defun compColon	215
5.4.22	defun compCons plist	218
5.4.23	defun compCons	219
5.4.24	defun compCons1	219
5.4.25	defun compConstruct plist	220
5.4.26	defun compConstruct	220
5.4.27	defun compConstructorCategory plist	221
5.4.28	defun compConstructorCategory plist	221
5.4.29	defun compConstructorCategory plist	222
5.4.30	defun compConstructorCategory plist	222
5.4.31	defun compConstructorCategory	222
5.4.32	defun compDefine plist	222
5.4.33	defun compDefine	223
5.4.34	defun compDefine1	223
5.4.35	defun compElt plist	225
5.4.36	defun compElt	225
5.4.37	defun compExit plist	227

5.4.38	defun compExit	227
5.4.39	defun compHas plist	228
5.4.40	defun compHas	228
5.4.41	defun compIf plist	228
5.4.42	defun compIf	228
5.4.43	defun compImport plist	229
5.4.44	defun compImport	230
5.4.45	defun compIs plist	230
5.4.46	defun compIs	230
5.4.47	defun compJoin plist	231
5.4.48	defun compJoin	231
5.4.49	defun compLambda plist	233
5.4.50	defun compLambda	233
5.4.51	defun compLeave plist	234
5.4.52	defun compLeave	234
5.4.53	defun compMacro plist	235
5.4.54	defun compMacro	235
5.4.55	defun compPretend plist	236
5.4.56	defun compPretend	236
5.4.57	defun compQuote plist	237
5.4.58	defun compQuote	237
5.4.59	defun compReduce plist	237
5.4.60	defun compReduce	238
5.4.61	defun compReduce1	238
5.4.62	defun compRepeatOrCollect plist	240
5.4.63	defun compRepeatOrCollect plist	240
5.4.64	defun compRepeatOrCollect	240
5.4.65	defun compReturn plist	242
5.4.66	defun compReturn	242
5.4.67	defun compSeq plist	243
5.4.68	defun compSeq	244
5.4.69	defun compSeq1	244
5.4.70	defun compSeqItem	245
5.4.71	defun compSetq plist	245
5.4.72	defun compSetq plist	245
5.4.73	defun compSetq	245
5.4.74	defun compSetq1	246
5.4.75	defun setqSetelt	246
5.4.76	defun setqSingle	247
5.4.77	defun isDomainForm	248
5.4.78	defun isDomainConstructorForm	249
5.4.79	defun compString plist	250
5.4.80	defun compString	250
5.4.81	defun compSubDomain plist	250
5.4.82	defun compSubDomain	250
5.4.83	defun compSubDomain1	251

5.4.84	defun compSubsetCategory plist	252
5.4.85	defun compSubsetCategory	252
5.4.86	defun compSuchthat plist	253
5.4.87	defun compSuchthat	253
5.4.88	defun compVector plist	254
5.4.89	defun compVector	254
5.4.90	defun compWhere plist	255
5.4.91	defun compWhere	255
6	Post Transformers	257
6.1	Direct called postparse routines	257
6.1.1	defun postTransform	257
6.1.2	defun postTran	258
6.1.3	defun postOp	259
6.1.4	defun postAtom	259
6.1.5	defun postTranList	260
6.1.6	defun postScriptsForm	260
6.1.7	defun postTranScripts	260
6.1.8	defun postTransformCheck	261
6.1.9	defun postcheck	261
6.1.10	defun postError	262
6.1.11	defun postForm	262
6.2	Indirect called postparse routines	263
6.2.1	defun postAdd plist	264
6.2.2	defun postAdd	264
6.2.3	defun postCapsule	265
6.2.4	defun postBlockItemList	265
6.2.5	defun postBlockItem	266
6.2.6	defun postAtSign plist	266
6.2.7	defun postAtSign	267
6.2.8	defun postType	267
6.2.9	defun postBigFloat plist	267
6.2.10	defun postBigFloat	268
6.2.11	defun postBlock plist	268
6.2.12	defun postBlock	268
6.2.13	defun postCategory plist	269
6.2.14	defun postCategory	269
6.2.15	defun postCollect,finish	270
6.2.16	defun postMakeCons	270
6.2.17	defun postCollect plist	271
6.2.18	defun postCollect	271
6.2.19	defun postIteratorList	272
6.2.20	defun postColon plist	272
6.2.21	defun postColon	273
6.2.22	defun postColonColon plist	273
6.2.23	defun postColonColon	273

6.2.24	defun postComma plist	274
6.2.25	defun postComma	274
6.2.26	defun comma2Tuple	274
6.2.27	defun postFlatten	274
6.2.28	defun postConstruct plist	275
6.2.29	defun postConstruct	275
6.2.30	defun postTranSegment	276
6.2.31	defun postDef plist	276
6.2.32	defun postDef	276
6.2.33	defun postDefArgs	278
6.2.34	defun postExit plist	279
6.2.35	defun postExit	279
6.2.36	defun postIf plist	279
6.2.37	defun postIf	279
6.2.38	defun postin plist	280
6.2.39	defun postin	280
6.2.40	defun postInSeq	280
6.2.41	defun postIn plist	281
6.2.42	defun postIn	281
6.2.43	defun postJoin plist	281
6.2.44	defun postJoin	282
6.2.45	defun postMapping plist	282
6.2.46	defun postMapping	282
6.2.47	defun postMDef plist	283
6.2.48	defun postMDef	283
6.2.49	defun postPretend plist	284
6.2.50	defun postPretend	284
6.2.51	defun postQUOTE plist	285
6.2.52	defun postQUOTE	285
6.2.53	defun postReduce plist	285
6.2.54	defun postReduce	285
6.2.55	defun postRepeat plist	286
6.2.56	defun postRepeat	286
6.2.57	defun postScripts plist	286
6.2.58	defun postScripts	287
6.2.59	defun postSemiColon plist	287
6.2.60	defun postSemiColon	287
6.2.61	defun postFlattenLeft	287
6.2.62	defun postSignature plist	288
6.2.63	defun postSignature	288
6.2.64	defun removeSuperfluousMapping	289
6.2.65	defun killColons	289
6.2.66	defun postSlash plist	289
6.2.67	defun postSlash	289
6.2.68	defun postTuple plist	290
6.2.69	defun postTuple	290

6.2.70	defun postTupleCollect plist	290
6.2.71	defun postTupleCollect	291
6.2.72	defun postWhere plist	291
6.2.73	defun postWhere	291
6.2.74	defun postWith plist	292
6.2.75	defun postWith	292
6.3	Support routines	292
6.3.1	defun setDefOp	292
6.3.2	defun aplTran	293
6.3.3	defun aplTran1	293
6.3.4	defun aplTranList	295
6.3.5	defun hasAplExtension	295
6.3.6	defun deepestExpression	296
6.3.7	defun containsBang	296
6.3.8	defun getScriptName	297
6.3.9	defun decodeScripts	297
7	DEF forms	299
7.0.10	defvar \$defstack	299
7.0.11	defvar \$is-spill	299
7.0.12	defvar \$is-spill-list	299
7.0.13	defvar \$vl	300
7.0.14	defvar \$is-gensymlist	300
7.0.15	defvar \$initial-gensym	300
7.0.16	defvar \$is-eqlist	300
7.0.17	defun hackforis	300
7.0.18	defun hackforis1	301
7.0.19	defun unTuple	301
7.0.20	defun errhuh	301
8	PARSE forms	303
8.1	The original meta specification	303
8.2	The PARSE code	308
8.2.1	defvar \$tmptok	308
8.2.2	defvar \$tok	308
8.2.3	defvar \$ParseMode	309
8.2.4	defvar \$definition-name	309
8.2.5	defvar \$lablasoc	309
8.2.6	defun PARSE-NewExpr	309
8.2.7	defun PARSE-Command	310
8.2.8	defun PARSE-SpecialKeyword	310
8.2.9	defun PARSE-SpecialCommand	311
8.2.10	defun PARSE-TokenCommandTail	311
8.2.11	defun PARSE-TokenOption	312
8.2.12	defun PARSE-TokenList	312
8.2.13	defun PARSE-CommandTail	313

8.2.14	defun PARSE-PrimaryOrQM	313
8.2.15	defun PARSE-Option	314
8.2.16	defun PARSE-Statement	314
8.2.17	defun PARSE-InfixWith	315
8.2.18	defun PARSE-With	315
8.2.19	defun PARSE-Category	315
8.2.20	defun PARSE-Expression	317
8.2.21	defun PARSE-Import	317
8.2.22	defun PARSE-Expr	318
8.2.23	defun PARSE-LedPart	318
8.2.24	defun PARSE-NudPart	318
8.2.25	defun PARSE-Operation	319
8.2.26	defun PARSE-leftBindingPowerOf	319
8.2.27	defun PARSE-rightBindingPowerOf	320
8.2.28	defun PARSE-getSemanticForm	320
8.2.29	defun PARSE-Prefix	320
8.2.30	defun PARSE-Infix	321
8.2.31	defun PARSE-TokTail	322
8.2.32	defun PARSE-Qualification	322
8.2.33	defun PARSE-Reduction	323
8.2.34	defun PARSE-ReductionOp	323
8.2.35	defun PARSE-Form	323
8.2.36	defun PARSE-Application	324
8.2.37	defun PARSE-Label	325
8.2.38	defun PARSE-Selector	325
8.2.39	defun PARSE-PrimaryNoFloat	326
8.2.40	defun PARSE-Primary	326
8.2.41	defun PARSE-Primary1	326
8.2.42	defun PARSE-Float	327
8.2.43	defun PARSE-FloatBase	328
8.2.44	defun PARSE-FloatBasePart	328
8.2.45	defun PARSE-FloatExponent	329
8.2.46	defun PARSE-Enclosure	330
8.2.47	defun PARSE-IntegerTok	330
8.2.48	defun PARSE-FormalParameter	331
8.2.49	defun PARSE-FormalParameterTok	331
8.2.50	defun PARSE-Quad	331
8.2.51	defun PARSE-String	331
8.2.52	defun PARSE-VarForm	332
8.2.53	defun PARSE-Scripts	332
8.2.54	defun PARSE-ScriptItem	333
8.2.55	defun PARSE-Name	333
8.2.56	defun PARSE-Data	334
8.2.57	defun PARSE-Sexpr	334
8.2.58	defun PARSE-Sexpr1	334
8.2.59	defun PARSE-NBGliphTok	335

8.2.60	defun PARSE-GlyphTok	336
8.2.61	defun PARSE-AnyId	336
8.2.62	defun PARSE-Sequence	337
8.2.63	defun PARSE-Sequence1	337
8.2.64	defun PARSE-OpenBracket	338
8.2.65	defun PARSE-OpenBrace	338
8.2.66	defun PARSE-IteratorTail	339
8.2.67	defun PARSE-Iterator	339
8.2.68	The PARSE implicit routines	340
8.2.69	defun PARSE-Suffix	340
8.2.70	defun PARSE-SemiColon	341
8.2.71	defun PARSE-Return	341
8.2.72	defun PARSE-Exit	341
8.2.73	defun PARSE-Leave	342
8.2.74	defun PARSE-Seg	342
8.2.75	defun PARSE-Conditional	343
8.2.76	defun PARSE-ElseClause	343
8.2.77	defun PARSE-Loop	344
8.2.78	defun PARSE-LabelExpr	344
8.2.79	defun PARSE-FloatTok	345
8.3	The PARSE support routines	345
8.3.1	String grabbing	346
8.3.2	defun match-string	346
8.3.3	defun skip-blanks	346
8.3.4	defun token-lookahead-type	347
8.3.5	defun match-advance-string	347
8.3.6	defun initial-substring-p	348
8.3.7	defun quote-if-string	348
8.3.8	defun escape-keywords	349
8.3.9	defun isTokenDelimiter	349
8.3.10	defun underscore	350
8.3.11	Token Handling	350
8.3.12	defun getToken	350
8.3.13	defun unget-tokens	350
8.3.14	defun match-current-token	351
8.3.15	defun match-token	351
8.3.16	defun match-next-token	352
8.3.17	defun current-symbol	352
8.3.18	defun make-symbol-of	352
8.3.19	defun current-token	353
8.3.20	defun try-get-token	353
8.3.21	defun next-token	354
8.3.22	defun advance-token	354
8.3.23	defvar \$XTokenReader	355
8.3.24	defun get-token	355
8.3.25	Character handling	355

8.3.26	defun current-char	355
8.3.27	defun next-char	355
8.3.28	defun char-eq	356
8.3.29	defun char-ne	356
8.3.30	Error handling	356
8.3.31	defvar \$meta-error-handler	356
8.3.32	defun meta-syntax-error	357
8.3.33	Floating Point Support	357
8.3.34	defun floatexpid	357
8.3.35	Dollar Translation	357
8.3.36	defun dollarTran	357
8.3.37	Applying metagrammatical elements of a production (e.g., Star).	358
8.3.38	defmacro Bang	358
8.3.39	defmacro must	358
8.3.40	defun action	359
8.3.41	defun optional	359
8.3.42	defmacro star	359
8.3.43	Stacking and retrieving reductions of rules.	360
8.3.44	defvar \$reduce-stack	360
8.3.45	defmacro reduce-stack-clear	360
8.3.46	defun push-reduction	360
9	Utility Functions	361
9.0.47	defun translablel	361
9.0.48	defun translablel1	361
9.0.49	defun displayPreCompilationErrors	362
9.0.50	defun bumperrorcount	363
9.0.51	defun parseTranCheckForRecord	363
9.0.52	defun new2OldLisp	364
9.0.53	defun makeSimplePredicateOrNil	364
9.0.54	defun parse-spadstring	364
9.0.55	defun parse-string	365
9.0.56	defun parse-identifier	365
9.0.57	defun parse-number	366
9.0.58	defun parse-keyword	366
9.0.59	defun parse-argument-designator	367
9.0.60	defun print-package	367
9.0.61	defun checkWarning	367
9.0.62	defun tuple2List	368
9.0.63	defmacro pop-stack-1	368
9.0.64	defmacro pop-stack-2	369
9.0.65	defmacro pop-stack-3	369
9.0.66	defmacro pop-stack-4	369
9.0.67	defmacro nth-stack	370
9.0.68	defun Pop-Reduction	370
9.0.69	defun addclose	370

9.0.70	defun blankp	371
9.0.71	defun drop	371
9.0.72	defun escaped	371
9.0.73	defvar \$comblocklist	371
9.0.74	defun fincomblock	372
9.0.75	defun indent-pos	372
9.0.76	defun infxtok	373
9.0.77	defun is-console	373
9.0.78	defun next-tab-loc	373
9.0.79	defun nonblankloc	374
9.0.80	defun parseprint	374
9.0.81	defun skip-to-endif	374
10	The Compiler	375
10.1	Compiling EQ.spad	375
10.1.1	The top level compiler command	378
10.1.2	The Spad compiler top level function	380
10.1.3	defun compilerDoit	384
10.1.4	defun /RQ,LIB	385
10.1.5	defun /rf-1	386
10.1.6	defun spad	395
10.1.7	defun Interpreter interface to the compiler	396
10.1.8	defun print-defun	399
10.1.9	defun def-rename	399
10.1.10	defun def-rename1	399
10.1.11	defun compTopLevel	400
10.1.12	defun compOrCroak	401
10.1.13	defun compOrCroak1	402
10.1.14	defun comp	403
10.1.15	defun compNoStacking	404
10.1.16	defun compNoStacking1	404
10.1.17	defun comp2	405
10.1.18	defun comp3	405
10.1.19	defun compTypeOf	408
10.1.20	defun compColonInside	408
10.1.21	defun compAtom	409
10.1.22	defun convert	411
10.1.23	defun primitiveType	411
10.1.24	defun compSymbol	411
10.1.25	defun compList	413
10.1.26	defun compExpression	413
10.1.27	defun compForm	414
10.1.28	defun compForm1	414
10.1.29	defun compForm2	416
10.1.30	defun compArgumentsAndTryAgain	418
10.1.31	defun compWithMappingMode	419

10.1.32 defun compWithMappingModel	419
10.1.33 defun extractCodeAndConstructTriple	427
10.1.34 defun hasFormalMapVariable	427
10.1.35 defun argsToSig	428
10.1.36 defun compMakeDeclaration	429
10.1.37 defun modifyModeStack	429
10.1.38 defun Create a list of unbound symbols	430
10.1.39 defun compOrCroak1,compactify	431
10.1.40 defun Compiler/Interpreter interface	431
10.1.41 defun compileSpadLispCmd	431
10.1.42 defun recompile-lib-file-if-necessary	433
10.1.43 defun spad-fixed-arg	433
10.1.44 defun compile-lib-file	433
10.1.45 defun compileFileQuietly	434
10.1.46 defvar \$byConstructors	434
10.1.47 defvar \$constructorsSeen	434

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 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 |$CategoryFrame|
  (|put| 'Equation| 'isFunctor|
    '(((|eval| ($ $ (|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      == [l, r]
  equation(l, r) == [l, 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 p94]
[$index p72]
[$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 p374]
[ifcar p??]
[$comblocklist p371]

```



```

[$skipme p??]
[$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 (pairp $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 p372]
[parsepiles p84]
[preparse1 doSystemCommand (vol5)]
[escaped p371]
[indent-pos p372]
[make-full-cvec p??]
[maxindex p??]
[preparse1 strposl (vol5)]
[is-console p373]
[spad-reader p??]
[$linelist p72]
[$echolinestack p72]
[$byConstructors p434]
[$skipme p??]
[$constructorsSeen p434]
[$preparse-last-line p72]
```

— 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))
  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 p373]
[drop p371]
[addclose p370]
[nonblankloc p374]

```

— 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 (infxtok 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 p93]
[string2BootTree p??]
[storeblanks p93]
[skip-to-endif p374]
[preparseReadLine p85]

```

— defun preparseReadLine —

```

(defun preparseReadLine (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)) (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 p93]
[string2BootTree p??]
[storeblanks p93]

```

— 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 p94]
[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 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)
```

3.4.3 defun read-a-line

[subseq p??]
[Line-New-Line p??]
[read-a-line p88]
[*eof* p??]

— defun read-a-line —

```
(defun read-a-line (&optional (stream t))
  (let (cp)
    (declare (special *eof*))
```

```
(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)
    (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))))))
```

3.5 Line Handling

3.5.1 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.

3.5.2 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))
```

3.5.3 defvar \$current-line

The current input line.

— **initvars** —

```
(defvar current-line (make-line))
```

3.5.4 defmacro line-clear

[*\$line* p89]

— **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)))
```

3.5.5 defun line-print

[*\$line* p89]

— **defun line-print** —

```
(defun line-print (line)
  (format out-stream "&~5D> ~A~%" (Line-Number line) (Line-Buffer line))
  (format out-stream "~v@T~%" (+ 7 (Line-Current-Index line))))
```

3.5.6 defun line-at-end-p

[*\$line* p89]

— **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)))
```

3.5.7 defun line-past-end-p

[*\$line* p89]

— **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)))
```

3.5.8 defun line-next-char

[*\$line* p89]

— **defun line-next-char** —

```
(defun line-next-char (line)
  (elt (line-buffer line) (1+ (line-current-index line))))
```

3.5.9 defun line-advance-char

[*\$line* p89]

— **defun line-advance-char** —

```
(defun line-advance-char (line)
  (setf (line-current-char line)
        (elt (line-buffer line) (incf (line-current-index line)))))
```

3.5.10 defun line-current-segment

[*\$line* p89]

— 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))))
```

—————

3.5.11 defun line-new-line

[*\$line* p89]

— 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)))))
```

—————

3.5.12 defun next-line

— defun next-line —

```
(defun next-line (&optional (in-stream t))
  (funcall Line-Handler in-stream))
```

—————

3.5.13 defun Advance-Char

[Line-At-End-P p??]
 [Line-Advance-Char p??]
 [next-line p92]
 [current-char p355]
 [\$line p89]

— **defun Advance-Char** —

```
(defun Advance-Char ()
  "Advances IN-STREAM, invoking Next Line if necessary."
  (loop
    (cond
      ((not (Line-At-End-P Current-Line))
       (return (Line-Advance-Char Current-Line)))
      ((next-line in-stream)
       (return (current-char)))
      ((return nil)))))
```

—————

3.5.14 defun storeblanks

— **defun storeblanks** —

```
(defun storeblanks (line n)
  (do ((i 0 (1+ i)))
      ((= i n) line)
    (setf (char line i) #\ )))
```

—————

3.5.15 defun initial-substring

[mismatch p??]

— **defun initial-substring** —

```
(defun initial-substring (pattern line)
  (let ((ind (mismatch pattern line)))
    (or (null ind) (eql ind (size pattern)))))
```

—————

3.5.16 defun get-a-line

[is-console p373]
 [get-a-line mkprompt (vol5)]
 [read-a-line p88]
 [make-string-adjustable p94]

— defun get-a-line —

```
(defun get-a-line (stream)
  (when (is-console stream) (princ (mkprompt))))
  (let ((l1 (read-a-line stream)))
    (if (stringp l1)
        (make-string-adjustable l1)
        l1)))
```

3.5.17 defun make-string-adjustable

— defun make-string-adjustable —

```
(defun make-string-adjustable (s)
  (if (adjustable-array-p s)
      s
      (make-array (array-dimensions s) :element-type 'string-char
                   :adjustable t :initial-contents s)))
```

3.5.18 Parsing stack

3.5.19 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)
```

)

3.5.20 defun stack-load

[*\$stack* p94]

— 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)))
```

3.5.21 defun stack-clear

[*\$stack* p94]

— 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))
```

3.5.22 defmacro stack-/-empty

[*\$stack* p94]

— defmacro stack-/-empty —

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

3.5.23 defun stack-push

[\$stack p94]

— 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)
```

—————

3.5.24 defun stack-pop

[\$stack p94]

— 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))
```

—————

3.5.25 Parsing token

3.5.26 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))
```

—————

3.5.27 defvar \$prior-token

[\$token p96]

— **initvars** —

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

—————

3.5.28 defvar \$nonblank— **initvars** —

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

—————

3.5.29 defvar \$current-token

Token at head of input stream. [\$token p96]

— **initvars** —

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

—————

3.5.30 defvar \$next-token

[\$token p96]

— **initvars** —

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

—————

3.5.31 defvar \$valid-tokens

[\$token p96]

— **initvars** —

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

—————

3.5.32 defun token-install

[\$token p96]

— **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)
```

—————

3.5.33 defun token-print

[\$token p96]

— **defun token-print** —

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

—————

3.5.34 Parsing reduction**3.5.35 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 p101]  
[$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 p102]  
[parseConstruct p103]  
[parseTran p101]  
[parseTranList p103]  
[getl p??]  
[$op p??]
```

— defun parseTran —

```

(defun |parseTran| (x)
  (labels (
    (g (op)
      (let (tmp1 tmp2 x)
        (seq
          (if (and (pairp op) (eq (qcar op) '|elt|))
              (progn
                (setq tmp1 (qcdr op))
                (and (pairp tmp1)
                  (progn
                    (setq op (qcar tmp1))
                    (setq tmp2 (qcdr tmp1))
                    (and (pairp tmp2)
                      (eq (qcdr tmp2) nil)
                      (progn (setq x (qcar 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 |$op|))
          (cond
            ((eq u '|construct|)
             (setq r (|parseConstruct| arg1))
             (if (and (pairp |$op|) (eq (qcar |$op|) '|elt|))
                 (cons (|parseTran| |$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 p129]
 [\$NoValue p??]

— defun parseAtom —

```

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

```

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

4.1.4 defun parseTranList

```
[parseTran p101]
[parseTranList p103]
```

— defun parseTranList —

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

4.1.5 defun parseConstruct

— postvars —

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

4.1.6 defun parseConstruct

```
[parseTranList p103]
[$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 defun parseAnd

— postvars —

```

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

```

4.2.2 defun parseAnd

```

[parseTran p101]
[parseAnd p105]
[parseTranList p103]
[parseIf p122]
[$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 defun parseAtSign

— postvars —

```

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

```

4.2.4 defun parseAtSign

```
[parseTran p101]
[parseType p106]
[$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 p101]
```

— defun parseType —

```
(defun |parseType| (x)
  (declare (special |$EmptyMode| |$quadSymbol|))
  (setq x (msubst |$EmptyMode| |$quadSymbol| x))
  (if (and (pairp x) (eq (qcar x) '|typeOf|)
        (pairp (qcdr x)) (eq (qcdr (qcdr x)) nil))
    (list '|typeOf| (|parseTran| (qcar (qcdr x))))
    x))
```

4.2.6 defun parseCategory

— postvars —

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

4.2.7 defun parseCategory

```
[parseTranList p103]
[parseDropAssertions p107]
[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 p107]
```

— defun parseDropAssertions —

```
(defun |parseDropAssertions| (x)
  (cond
    ((not (pairp x)) x)
    ((and (pairp (qcar x)) (eq (qcar (qcar x)) 'if)
          (pairp (qcdr (qcar x)))
          (eq (qcar (qcdr (qcar x))) '|asserted|))
      (|parseDropAssertions| (qcdr x)))
    (t (cons (qcar x) (|parseDropAssertions| (qcdr x))))))
```

—————

4.2.9 defun parseCoerce

— postvars —

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

—————

4.2.10 defun parseCoerce

```
[parseType p106]
[parseTran p101]
[$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 defun parseColon

— postvars —

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

—————

4.2.12 defun parseColon

```
[parseTran p101]
[parseType p106]
[$InteractiveMode p??]
[$insideConstructIfTrue p??]
```

— defun parseColon —

```
(defun |parseColon| (arg)
  (cond
    ((and (pairp arg) (eq (qcdr arg) nil))
     (list '|:| (|parseTran| (first arg))))
    ((and (pairp arg) (pairp (qcdr arg)) (eq (qcdr (qcdr 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 defun parseDEF

— postvars —

```

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

```

4.2.14 defun parseDEF

```

[setDefOp p292]
[parseLhs p110]
[parseTranList p103]
[parseTranCheckForRecord p363]
[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 p101]
[transIs p110]
```

— **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 p111]
[transIs1 p110]
```

— **defun transIs** —

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

4.2.17 defun transIs1

```
[qcar p??]
[qcdr p??]
[pairp p??]
[nreverse0 p??]
[transIs p110]
[transIs1 p110]
```

— **defun transIs1** —

```

(defun |transIs1| (u)
  (let (x h v tmp3)
    (cond
      ((and (pairp u) (eq (qcar u) '|construct|))
        (dolist (x (qcdr u) (nreverse0 tmp3))
          (push (|transIs1| x) tmp3)))
      ((and (pairp u) (eq (qcar u) '|append|) (pairp (qcdr u))
        (pairp (qcdr (qcdr u))) (eq (qcdr (qcdr (qcdr u))) nil))
        (setq x (qcar (qcdr u)))
        (setq h (list '|:| (|transIs1| x)))
        (setq v (|transIs1| (qcar (qcdr (qcdr u))))))
      (cond
        ((and (pairp v) (eq (qcar v) '|:|)
          (pairp (qcdr v)) (eq (qcdr (qcdr v)) nil))
          (list h (qcar (qcdr v))))
        ((eq v '|nil|) (car (cdr h)))
        ((atom v) (list h (list '|:| v)))
        (t (cons h v))))
      ((and (pairp u) (eq (qcar u) '|cons|) (pairp (qcdr u))
        (pairp (qcdr (qcdr u))) (eq (qcdr (qcdr (qcdr u))) nil))
        (setq h (|transIs1| (qcar (qcdr u))))
        (setq v (|transIs1| (qcar (qcdr (qcdr u))))))
      (cond
        ((and (pairp v) (eq (qcar v) '|:|) (pairp (qcdr v))
          (eq (qcdr (qcdr v)) nil))
          (cons h (list (qcar (qcdr 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 (pairp u) (|member| (qcar u) '(|construct| |append| |cons|))))

```

4.2.19 defun parseDollarGreaterthan

— postvars —

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

4.2.20 defun parseDollarGreaterThan

```
[msubst p??]
[parseTran p101]
[$op p??]
```

— defun parseDollarGreaterThan —

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

4.2.21 defun parseDollarGreaterEqual

— postvars —

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

4.2.22 defun parseDollarGreaterEqual

```
[msubst p??]
[parseTran p101]
[$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 p101]
[$op p??]
```

— defun parseDollarLessEqual —

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

4.2.24 defun parseDollarNotEqual

— postvars —

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

4.2.25 defun parseDollarNotEqual

```
[parseTran p101]
[msubst p??]
```

`[$op p??]`

— `defun parseDollarNotEqual` —

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

—————

4.2.26 `defun parseEquivalence`

— `postvars` —

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

—————

4.2.27 `defun parseEquivalence`

`[parseIf p122]`

— `defun parseEquivalence` —

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

—————

4.2.28 `defun parseExit`

— `postvars` —

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

—————

4.2.29 defun parseExit

```
[parseTran p101]
[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 defun parseGreaterEqual

— postvars —

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

4.2.31 defun parseGreaterEqual

```
[parseTran p101]
[$op p??]
```

— defun parseGreaterEqual —

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

4.2.32 defun parseGreaterThan

— postvars —

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

—————

4.2.33 defun parseGreaterThan

```
[parseTran p101]
[$op p??]
```

— defun parseGreaterThan —

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

—————

4.2.34 defun 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 p118]
[member p??]
[parseType p106]
[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 (pairp arg) (eq (qcar arg) '|:|) (pairp (qcdr arg))
            (pairp (qcdr (qcdr arg))) (eq (qcdr (qcdr (qcdr arg))) nil)
            (pairp (qcar (qcdr (qcdr arg))))
            (eq (qcar (qcar (qcdr (qcdr 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 (pairp arg) (eq (qcar arg) '|Join|))
           (dolist (z (rest arg) tmp4)
             (setq tmp4 (append tmp4 (fn z))))))
          ((and (pairp arg) (eq (qcar 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 (pairp arg) (eq (qcar arg) 'attribute))
             (list arg))
            ((and (pairp arg) (eq (qcar 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 (pairp tmp1) (pairp (qcdr tmp1)) (pairp (qcdr (qcdr tmp1)))
        (eq (qcdr (qcdr (qcdr 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 (pairp tmp2) (eq (qcdr tmp2) nil))
    (qcar tmp2)
    (cons '|and| tmp2))))

```

4.2.36 defun parseHasRhs

```

[get p??]
[qcar p??]
[qcdr p??]
[member p??]
[abbreviation? p??]
[loadIfNecessary p119]
[unabbrevAndLoad p??]
[$CategoryFrame p??]

```

— defun parseHasRhs —

```

(defun |parseHasRhs| (u)
  (let (tmp1 y)
    (declare (special |$CategoryFrame|))
    (setq tmp1 (|get| u '|value| |$CategoryFrame|))
    (cond
      ((and (pairp tmp1) (pairp (qcdr tmp1))
        (pairp (qcdr (qcdr tmp1))) (eq (qcdr (qcdr (qcdr 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 p119]

— defun loadIfNecessary —

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

—————

4.2.38 defun loadLibIfNecessary

[loadLibIfNecessary p119]

```
[functionp p??]
[macrop p??]
[getl p??]
[loadLib p??]
[lassoc p??]
[getProplist p??]
[getdatabase p??]
[updateCategoryFrameForCategory p121]
[updateCategoryFrameForConstructor p120]
[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 p120]
[addModemap p??]
[$CategoryFrame p??]
[$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 p??]
[$CategoryFrame p??]
[$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 defun parseIf

— postvars —

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

4.2.43 defun parseIf

```
[parseIf,ifTran p122]
[parseTran p101]
```

— defun parseIf —

```
(defun |parseIf| (arg)
  (if (null (and (pairp arg) (pairp (qcdr arg))
                (pairp (qcdr (qcdr arg))) (eq (qcdr (qcdr (qcdr arg))) nil)))
      arg
      (|parseIf,ifTran|
        (|parseTran| (first arg))
        (|parseTran| (second arg))
        (|parseTran| (third arg)))))
```

4.2.44 defun parseIf,ifTran

```
[parseIf,ifTran p122]
[incExitLevel p??]
[makeSimplePredicateOrNil p364]
[incExitLevel p??]
[parseTran p101]
[$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 (pairp pred) (eq (qcar pred) '|not|)
            (pairp (qcdr pred)) (eq (qcdr (qcdr pred)) nil))
       (|parseIf,ifTran| (second pred) b a))
```

```

((and (pairp pred) (eq (qcar pred) 'if)
  (progn
    (setq tmp1 (qcdr pred))
    (and (pairp tmp1)
      (progn
        (setq pp (qcar tmp1))
        (setq tmp2 (qcdr tmp1))
        (and (pairp tmp2)
          (progn
            (setq ap (qcar tmp2))
            (setq tmp3 (qcdr tmp2))
            (and (pairp tmp3)
              (eq (qcdr tmp3) nil)
              (progn (setq bp (qcar tmp3)) t))))))))
  (|parseIf,ifTran| pp
    (|parseIf,ifTran| ap (copy a) (copy b))
    (|parseIf,ifTran| bp a b)))
((and (pairp pred) (eq (qcar pred) 'seq)
  (pairp (qcdr pred)) (progn (setq tmp2 (reverse (qcdr pred))) t)
  (and (pairp tmp2)
    (pairp (qcar tmp2))
    (eq (qcar (qcar tmp2)) '|exit|)
    (progn
      (setq tmp4 (qcdr (qcar tmp2)))
      (and (pairp tmp4)
        (equal (qcar tmp4) 1)
        (progn
          (setq tmp5 (qcdr tmp4))
          (and (pairp tmp5)
            (eq (qcdr tmp5) nil)
            (progn (setq pp (qcar tmp5)) t))))
      (progn (setq z (qcdr tmp2)) t))
    (progn (setq z (nreverse z)) t))
  (cons 'seq
    (append z
      (list
        (list '|exit| 1 (|parseIf,ifTran| pp
          (|incExitLevel| a)
          (|incExitLevel| b)))))))
((and (pairp a) (eq (qcar a) 'if) (pairp (qcdr a))
  (equal (qcar (qcdr a)) pred) (pairp (qcdr (qcdr a)))
  (pairp (qcdr (qcdr (qcdr a))))
  (eq (qcdr (qcdr (qcdr (qcdr a)))) nil))
  (list 'if pred (third a) b))
((and (pairp b) (eq (qcar b) 'if)
  (pairp (qcdr b)) (equal (qcar (qcdr b)) pred)
  (pairp (qcdr (qcdr b)))
  (pairp (qcdr (qcdr (qcdr b))))
  (eq (qcdr (qcdr (qcdr (qcdr b)))) nil))
  (list 'if pred a (fourth b)))

```

```

((progn
  (setq tmp1 (|makeSimplePredicateOrNil| pred))
  (and (pairp tmp1) (eq (qcar tmp1) 'seq)
    (progn
      (setq tmp2 (qcdr tmp1))
      (and (and (pairp tmp2)
        (progn (setq tmp3 (reverse tmp2)) t))
        (and (pairp tmp3)
          (progn
            (setq tmp4 (qcar tmp3))
            (and (pairp tmp4) (eq (qcar tmp4) '|exit|)
              (progn
                (setq tmp5 (qcdr tmp4))
                (and (pairp tmp5) (equal (qcar tmp5) 1)
                  (progn
                    (setq tmp6 (qcdr tmp5))
                    (and (pairp tmp6) (eq (qcdr tmp6) nil)
                      (progn (setq val (qcar tmp6)) t)))))))
                (progn (setq s (qcdr 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 defun parseImplies

— postvars —

```

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

```

4.2.46 defun parseImplies

[parseIf p122]

— defun parseImplies —


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

4.2.47 defun parseIn

— postvars —

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

4.2.48 defun parseIn

```
[parseTran p101]
[postError p262]
```

— defun parseIn —

```
(defun |parseIn| (arg)
  (let (i n)
    (setq i (|parseTran| (first arg)))
    (setq n (|parseTran| (second arg)))
    (cond
      ((and (pairp n) (eq (qcar n) 'segment)
            (pairp (qcdr n)) (eq (qcdr (qcdr n)) nil))
        (list 'step i (second n) 1))
      ((and (pairp n) (eq (qcar n) '|reverse|)
            (pairp (qcdr n)) (eq (qcdr (qcdr n)) nil)
            (pairp (qcar (qcdr n))) (eq (qcar (qcar (qcdr n))) 'segment)
            (pairp (qcdr (qcar (qcdr n))))
            (eq (qcdr (qcdr (qcar (qcdr n)))) nil))
        (|postError| (list " You cannot reverse an infinite sequence." )))
      ((and (pairp n) (eq (qcar n) 'segment)
            (pairp (qcdr n)) (pairp (qcdr (qcdr n)))
            (eq (qcdr (qcdr (qcdr n))) nil))
        (if (third n)
            (list 'step i (second n) 1 (third n))
            (list 'step i (second n) 1)))
      ((and (pairp n) (eq (qcar n) '|reverse|)
            (pairp (qcdr n)) (eq (qcdr (qcdr n)) nil)
            (pairp (qcar (qcdr n))) (eq (qcar (qcar (qcdr n))) 'segment)
```

```

      (pairp (qcdr (qcar (qcdr n))))
      (pairp (qcdr (qcdr (qcar (qcdr n)))))
      (eq (qcdr (qcdr (qcdr (qcar (qcdr 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 (pairp n) (eq (qcar n) '|tails|)
          (pairp (qcdr n)) (eq (qcdr (qcdr n)) nil))
      (list 'on i (second n)))
    (t
      (list 'in i n))))

```

4.2.49 defun parseInBy

— postvars —

```

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

```

4.2.50 defun parseInBy

```

[postError p262]
[parseTran p101]
[bright p??]
[parseIn p125]

```

— 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 (pairp u) (eq (qcar u) 'step)
                  (pairp (qcdr u))
                  (pairp (qcdr (qcdr u)))
                  (pairp (qcdr (qcdr (qcdr 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 defun parseIs

— postvars —

```

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

```

4.2.52 defun parseIs

```

[parseTran p101]
[transIs p110]

```

— defun parseIs —

```

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

```

4.2.53 defun parseIsnt

— postvars —

```

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

```

4.2.54 defun parseIsnt

```
[parseTran p101]
[transIs p110]
```

— **defun parseIsnt** —

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

4.2.55 defun parseJoin

— **postvars** —

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

4.2.56 defun parseJoin

```
[parseTranList p103]
```

— **defun parseJoin** —

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

4.2.57 defun parseLeave

— postvars —

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

4.2.58 defun parseLeave

[parseTran p101]

— 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 defun parseLessEqual

— postvars —

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

4.2.60 defun parseLessEqual

```
[parseTran p101]
[$op p??]
```

— defun parseLessEqual —

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

—————

4.2.61 defun parseLET

— postvars —

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

—————

4.2.62 defun parseLET

```
[parseTran p101]
[parseTranCheckForRecord p363]
[opOf p??]
[transIs p110]
```

— 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 defun parseLETD

— postvars —

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

—————

4.2.64 defun parseLETD

```
[parseTran p101]
[parseType p106]
```

— defun parseLETD —

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

—————

4.2.65 defun parseMDEF

— postvars —

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

—————

4.2.66 defun parseMDEF

```
[parseTran p101]
[parseTranList p103]
[parseTranCheckForRecord p363]
[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 defun parseNot

— postvars —

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

4.2.68 defun parseNot

— postvars —

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

4.2.69 defun parseNot

```
[parseTran p101]
[$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 defun parseNotEqual

— postvars —

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

4.2.71 defun parseNotEqual

```
[parseTran p101]
[msubst p??]
[$op p??]
```

— defun parseNotEqual —

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

4.2.72 defun parseOr

— postvars —

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

4.2.73 defun parseOr

```
[parseTran p101]
[parseTranList p103]
```

```
[parseIf p122]
[parseOr p133]
```

— 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 (pairp x) (eq (qcar x) '|not|)
            (pairp (qcdr x)) (eq (qcdr (qcdr x)) nil))
       (|parseIf| (list (second x) (|parseOr| (cdr arg)) '|true|)))
      (t
       (|parseIf| (list x '|true| (|parseOr| (cdr arg))))))))
```

4.2.74 defun parsePretend

— postvars —

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

4.2.75 defun parsePretend

```
[parseTran p101]
[parseType p106]
```

— 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 defun parseReturn

— postvars —

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

4.2.77 defun parseReturn

```
[parseTran p101]
[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 defun parseSegment

— postvars —

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

4.2.79 defun parseSegment

[parseTran p101]

— **defun parseSegment** —

```
(defun |parseSegment| (arg)
  (if (and (pairp arg) (pairp (qcdr arg)) (eq (qcdr (qcdr arg)) nil))
      (if (second arg)
          (list 'segment (|parseTran| (first arg)) (|parseTran| (second arg)))
          (list 'segment (|parseTran| (first arg))))
      (cons 'segment arg)))
```

4.2.80 defun parseSeq— **postvars** —

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

4.2.81 defun parseSeq

```
[postError p262]
[transSeq p??]
[mapInto p??]
[last p??]
```

— **defun parseSeq** —

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

4.2.82 defun parseVCONS

— postvars —

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

4.2.83 defun parseVCONS

[parseTranList p103]

— defun parseVCONS —

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

4.2.84 defun 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

5.1 Routines for handling forms

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

- `add` (p205) `compAdd(form mode env) → (form mode env)`
- `@` (p207) `compAtSign(form mode env) →`
- `CAPSULE` (p208) `compCapsule(form mode env) →`
- `case` (p209) `compCase(form mode env) →`
- `Mapping` (p211) `compCat(form mode env) →`
- `Record` (p211) `compCat(form mode env) →`
- `Union` (p211) `compCat(form mode env) →`
- `CATEGORY` (p212) `compCategory(form mode env) →`
- `::` (p213) `compCoerce(form mode env) →`
- `:` (p215) `compColon(form mode env) →`
- `CONS` (p219) `compCons(form mode env) →`
- `construct` (p220) `compConstruct(form mode env) →`
- `ListCategory` (p222) `compConstructorCategory(form mode env) →`
- `RecordCategory` (p222) `compConstructorCategory(form mode env) →`
- `UnionCategory` (p222) `compConstructorCategory(form mode env) →`
- `VectorCategory` (p222) `compConstructorCategory(form mode env) →`

- `DEF` (p223) `compDefine(form mode env) →`
- `elt` (p225) `compElt(form mode env) →`
- `exit` (p227) `compExit(form mode env) →`
- `has` (p228) `compHas(pred mode $e) →`
- `IF` (p228) `compIf(form mode env) →`
- `import` (p230) `compImport(form mode env) →`
- `is` (p230) `compIs(form mode env) →`
- `Join` (p231) `compJoin(form mode env) →`
- `+->` (p233) `compLambda(form mode env) →`
- `leave` (p234) `compLeave(form mode env) →`
- `MDEF` (p235) `compMacro(form mode env) →`
- `pretend` (p236) `compPretend →`
- `QUOTE` (p237) `compQuote(form mode env) →`
- `REDUCE` (p238) `compReduce(form mode env) →`
- `COLLECT` (p240) `compRepeatOrCollect(form mode env) →`
- `REPEAT` (p240) `compRepeatOrCollect(form mode env) →`
- `return` (p242) `compReturn(form mode env) →`
- `SEQ` (p244) `compSeq(form mode env) →`
- `LET` (p245) `compSetq(form mode env) →`
- `SETQ` (p245) `compSetq(form mode env) →`
- `String` (p250) `compString(form mode env) →`
- `SubDomain` (p250) `compSubDomain(form mode env) →`
- `SubsetCategory` (p252) `compSubsetCategory(form mode env) →`
- `|` (p253) `compSuchthat(form mode env) →`
- `VECTOR` (p254) `compVector(form mode env) →`
- `where` (p255) `compWhere(form mode eInit) →`

5.2 Functions which handle == statements

5.2.1 defun compDefineAddSignature

```
[hasFullSignature p141]
[assoc p??]
[lassoc p??]
[getProplist p??]
[comp p403]
[$EmptyMode p??]
```

— 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 p142]
[compOrCroak p401]

```

— defun getTargetFromRhs —

```

(defun |getTargetFromRhs| (lhs rhs env)
  (declare (special |$EmptyMode|))
  (cond
    ((and (pairp rhs) (eq (qcar rhs) 'capsule))
     (|stackSemanticError|
      (list "target category of " lhs
            " cannot be determined from definition")
      nil))
    ((and (pairp rhs) (eq (qcar rhs) '|SubDomain|) (pairp (qcdr rhs)))
     (|getTargetFromRhs| lhs (second rhs) env))
    ((and (pairp rhs) (eq (qcar rhs) '|add|)
          (pairp (qcdr rhs)) (pairp (qcdr (qcdr rhs)))
          (eq (qcdr (qcdr (qcdr rhs))) nil)
          (pairp (qcar (qcdr (qcdr rhs))))
          (eq (qcar (qcar (qcdr (qcdr rhs)))) 'capsule))
     (|getTargetFromRhs| lhs (second rhs) env))
    ((and (pairp rhs) (eq (qcar rhs) '|Record|))

```

```

      (cons '|RecordCategory| (rest rhs)))
    ((and (pairp rhs) (eq (qcar rhs) '|Union|))
     (cons '|UnionCategory| (rest rhs)))
    ((and (pairp rhs) (eq (qcar rhs) '|List|))
     (cons '|ListCategory| (rest rhs)))
    ((and (pairp rhs) (eq (qcar 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 p144]

— 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 p144]
[macroExpandList p144]

— defun macroExpand —

```
(defun |macroExpand| (form env)
  (let (u)
    (cond
      ((atom form)
       (if (setq u (|get| form '|macro| env))
           (|macroExpand| u env)
           form))
      ((and (pairp form) (eq (qcar form) 'def)
            (pairp (qcdr form))
            (pairp (qcdr (qcdr form)))
            (pairp (qcdr (qcdr (qcdr form))))
            (pairp (qcdr (qcdr (qcdr (qcdr form))))))
       (eq (qcdr (qcdr (qcdr (qcdr (qcdr 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 p144]
[getdatabase p??]

— defun macroExpandList —

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

5.2.9 defun compDefineCategory1

```
[compDefineCategory2 p149]
[makeCategoryPredicates p146]
[compDefine1 p223]
[mkCategoryPackage p146]
[$insideCategoryPackageIfTrue p??]
[$EmptyMode p??]
[$categoryPredicateList p??]
[$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|) $ $) (NIL NIL NIL)
    ;;                (IF (= |x| |y|) |false| |true|)))))
    (setq form (second df))
    (setq sig (third df))
    (setq sc (fourth df))
    (setq body (fifth df))
    (setq categoryCapsule
      (when (and (pairp body) (eq (qcar body) '|add|)
                (pairp (qcdr body)) (pairp (qcdr (qcdr body)))
                (eq (qcdr (qcdr (qcdr body))) nil))
        (setq tmp1 (third body))
        (setq body (second body))
        tmp1))
    (setq tmp3 (|compDefineCategory2| form sig sc body mode env prefix fal))
    (setq d (first tmp3))
    (setq mode (second tmp3))
    (setq env (third tmp3))
    (when (and categoryCapsule (null |$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

```
[$FormalMapVariableList p202]
[$TriangleVariableList p??]
[$mvl p??]
[$tv1 p??]
```

— defun makeCategoryPredicates —

```
(defun |makeCategoryPredicates| (form u)
  (labels (
    (fn (u pl)
      (declare (special |$tv1| |$mvl|))
      (cond
        ((and (pairp u) (eq (qcar u) '|Join|) (pairp (qcdr u)))
         (fn (car (reverse (qcdr u))) pl))
        ((and (pairp u) (eq (qcar u) '|has|))
         (|insert| (eqsubstlist |$mvl| |$tv1| u) pl))
        ((and (pairp u) (member (qcar u) '(signature attribute))) pl)
        ((atom u) pl)
        (t (fnl u pl))))
    (fnl (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 p202]
```

— **defun mkCategoryPackage** —

```
(defun |mkCategoryPackage| (form cat def)
  (labels (
    (fn (x oplist)
      (cond
        ((atom x) oplist)
        ((and (pairp x) (eq (qcar x) 'def) (pairp (qcdr x)))
         (cons (second x) oplist))
        (t
         (fn (cdr x) (fn (car x) oplist))))))
    (gn (cat)
      (cond
        ((and (pairp cat) (eq (qcar cat) 'category)) (cddr cat))
        ((and (pairp cat) (eq (qcar cat) '|Join|)) (gn (|last| (qcdr cat))))
        (t nil))))
    (let (|$options| op argl packageName packageAbb nameForDollar packageArg1
          capsuleDefAlist explicitCatPart catvec fullCatOpList op1 sig
          catOpList packageCategory nils packageSig)
      (declare (special |$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 packageArg1 (cons nameForDollar argl))
      (setq capsuleDefAlist (fn def nil))
      (setq explicitCatPart (gn cat))
      (setq catvec (|eval| (|mkEvalableCategoryForm| 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 packageArg1)
    packageSig (cons nil nils) def))))))

```

5.2.12 defun mkEvaluableCategoryForm

```

[pairp p??]
[qcar p??]
[qcdr p??]
[mkEvaluableCategoryForm p148]
[compOrCroak p401]
[getdatabase p??]
[get p??]
[quotifyCategoryArgument p??]
[mkq p??]
[$Category p??]
[$e p??]
[$EmptyMode p??]
[$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 (pairp c)
      (progn
        (setq op (qcar c))
        (setq arg1 (qcdr c))
        (cond
          ((eq op '|Join|)
            (cons '|Join|
              (loop for x in arg1
                collect (|mkEvaluableCategoryForm| x))))
          ((eq op '|DomainSubstitutionMacro|)
            (|mkEvaluableCategoryForm| (cadr arg1))))
      nil)

```



```

((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 (|quotifyCategoryArgument| 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 p??]
[giveFormalParametersValues p143]
[take p??]
[sublis p??]
[compMakeDeclaration p429]
[nequal p??]
[opOf p??]
[optFunctorBody p??]
[compOrCroak p401]
[mkConstructor p154]
[compile p??]
[lisplibWrite p166]
[removeZeroOne p??]
[mkq p??]
[evalAndRwriteLispForm p154]
[eval p??]
[getParentsFor p??]
[computeAncestorsOf p??]
[constructor? p??]
[augLisplibModemapsFromCategory p152]
[$prefix p??]

```

```

[$formalArgList p??]
[$insideCategoryIfTrue p??]
[$stop-level p??]
[$definition p??]
[$form p??]
[$op p??]
[$extraParms p??]
[$functionStats p??]
[$functorStats p??]
[$frontier p??]
[$getDomainCode p??]
[$addForm p??]
[$lisplibAbbreviation p??]
[$lisplibAncestors p??]
[$lisplibCategory p??]
[$FormalMapVariableList p202]
[$lisplibParents p??]
[$lisplibModemap p??]
[$lisplibKind p??]
[$lisplibForm p??]
[$lisplib p??]
[$domainShell p??]
[$libFile p??]
[$TriangleVariableList 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| |$lisplibAbbreviation|
                      |$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 augLisplibModemapsFromCategory

```

[sublis p??]
[mkAlistOfExplicitCategoryOps p??]
[isCategoryForm p??]
[lassoc p??]

```

```

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

```

5.2.15 defun evalAndRwriteLispForm

[eval p??]
[rwriteLispForm p154]

— defun evalAndRwriteLispForm —

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

5.2.16 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)))
```

5.2.17 defun mkConstructor

[mkConstructor p154]

— 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)))))))
```

5.2.18 defun compDefineCategory

```
[compDefineLisplib p155]
[compDefineCategory1 p145]
[$domainShell p??]
[$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))))
```

5.2.19 defun compDefineLisplib

```
[sayMSG p??]
[fillerSpaces p??]
[getConstructorAbbreviation p??]
[compileDocumentation p158]
[bright p??]
[finalizeLisplib p160]
[rshut p??]
[lisplibDoRename p158]
[filep p??]
[rpackfile p??]
[unloadOneConstructor p??]
[localdatabase p??]
[getdatabase p??]
[updateCategoryFrameForCategory p121]
[updateCategoryFrameForConstructor p120]
[$compileDocumentation p158]
[$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))))

```

5.2.20 defun compileDocumentation

```
[make-input-filename p??]
[rdefiostream p??]
[lisplibWrite p166]
[finalizeDocumentation p??]
[rshut p??]
[rpackfile p??]
[replaceFile p??]
[$fcopy p??]
[$spadLibFT p??]
[$EmptyMode p??]
[$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|)))
```

—————

5.2.21 defun lisplibDoRename

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

— defun lisplibDoRename —

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

—————

5.2.22 defun initializeLisplib

```

[erase p??]
[writeLib1 p160]
[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))))
```

5.2.23 defun writeLib1

```
[rdefiostream p??]
```

— defun writeLib1 —

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

5.2.24 defun finalizeLisplib

```
[lisplibWrite p166]
[removeZeroOne p??]
[namestring p??]
[getConstructorOpsAndAtts p162]
[NRTgenInitialAttributeAlist p??]
[mergeSignatureAndLocalVarAlists p165]
[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 p202]
[$lisplibSuperDomain p??]
[$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))))

```

5.2.25 defun getConstructorOpsAndAtts

[getCategoryOpsAndAtts p162]
 [getFunctorOpsAndAtts p165]

— defun getConstructorOpsAndAtts —

```

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

```

5.2.26 defun getCategoryOpsAndAtts

[transformOperationAlist p163]
 [getSlotFromCategoryForm p163]
 [getSlotFromCategoryForm p163]

— defun getCategoryOpsAndAtts —

```

(defun |getCategoryOpsAndAtts| (catForm)

```

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

5.2.27 defun getSlotFromCategoryForm

```
[eval p??]
[|take p??|]
[|systemErrorHere p??|]
[$FormalMapVariableList p202]
```

— defun getSlotFromCategoryForm —

```
(defun |getSlotFromCategoryForm| (opargs index)
  (let (op arg1 u)
    (declare (special |$FormalMapVariableList|))
    (setq op (first opargs))
    (setq arg1 (rest opargs))
    (setq u
      (|eval| (cons op (mapcar 'mkq (take (|#| arg1) |$FormalMapVariableList|))))))
    (if (null (vecp u))
      (|systemErrorHere| "getSlotFromCategoryForm")
      (elt u index))))
```

5.2.28 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 tmp1 tmp2 impOp kind u n
        signatureItem itemList newAlist)
    (declare (special |$functionLocations|))
    (setq newAlist nil)
    (dolist (item operationAlist)
      (setq op (caar item))
      (setq sig (cadar item))
      (setq condition (cadr item))
      (setq implementation (caddr item))
      (setq kind
        (cond
          ((and (pairp implementation) (pairp (qcdr implementation))
                (pairp (qcdr (qcdr implementation)))
                (eq (qcdr (qcdr (qcdr implementation))) nil)
                (progn (setq n (qcar (qcdr (qcdr implementation)))) t)
                (|member| (setq eltEtc (qcar implementation)) '(const elt))))
            eltEtc)
          ((pairp implementation)
            (setq impOp (qcar implementation))
            (cond
              ((eq impOp 'xlam) implementation)
              ((|member| impOp '(const |Subsumed|)) impOp)
              (t (|keyedSystemError| 's2il0025 (list impOp))))
            ((eq implementation '|mkRecord|) '|mkRecord|)
            (t (|keyedSystemError| 's2il0025 (list implementation))))
        (when (setq u (|assoc| (list op sig) |$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))
```

5.2.29 defun getFunctorOpsAndAtts

```
[transformOperationAlist p163]
[getSlotFromFunctor p165]
```

— defun getFunctorOpsAndAtts —

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

5.2.30 defun getSlotFromFunctor

```
[compMakeCategoryObject p??]
[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))))
```

5.2.31 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))))))
```

5.2.32 defun lisplibWrite

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

— defun lisplibWrite —

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

5.2.33 defun compDefineFunctor

```
[compDefineLisplib p155]
[compDefineFunctor1 p167]
[$domainShell p??]
[$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))))
```

5.2.34 defun compDefineFunctor1

```

[isCategoryPackageName p??]
[getArgumentModeOrMoan p??]
[modemap2Signature p??]
[getModemap p193]
[giveFormalParametersValues p143]
[compMakeCategoryObject p??]
[sayBrightly p??]
[pp p??]
[strconc p??]
[pname p??]
[disallowNilAttribute p176]
[remdup p??]
[NRTgenInitialAttributeAlist p??]
[NRTgetLocalIndex p??]
[compMakeDeclaration p429]
[pairp p??]
[qcar p??]
[qcdr p??]
[augModemapsFromCategoryRep p203]
[augModemapsFromCategory p195]
[sublis p??]
[isPackageFunction p??]
[maxindex p??]
[makeFunctorArgumentParameters p178]
[compFunctorBody p176]
[reportOnFunctorCompilation p177]
[compile p??]
[augmentLisplibModemapsFromFunctor p174]
[reportOnFunctorCompilation p177]
[getParentsFor p??]
[computeAncestorsOf p??]
[constructor? p??]
[nequal p??]
[NRTmakeSlot1Info p??]
[isCategoryPackageName p??]
[lisplibWrite p166]
[mkq p??]
[getdatabase p??]
[NRTgetLookupFunction p??]
[simpBool p??]
[removeZeroOne p??]
[evalAndRwriteLispForm p154]
[$lisplib p??]
[$top-level p??]

```

```

[$bootStrapMode p??]
[$CategoryFrame p??]
[$CheckVectorList p??]
[$FormalMapVariableList p202]
[$LocalDomainAlist p??]
[$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 (pairp cb) (pairp (qcar cb)) (eq (qcar (qcar cb)) 'let)
          (pairp (qcdr (qcar cb))) (eq (qcar (qcdr (qcar cb))) '|Rep|)
          (pairp (qcdr (qcdr (qcar 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| |$functionLocations|
          |$isOpPackageName| |$lookupFunction| |$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 $stop_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| $isOpPackageName| $libFile|
|$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 $stop_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| |$form|)
(unless (car signaturep)
  (setq signaturep (|modemap2Signature| (|getModemap| |$form| |$e|))))
(setq target (first signaturep))
(setq |$functorTarget| target)
(setq |$e| (|giveFormalParametersValues| argl |$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 |$functionLocations| nil)
    (loop for x in argl do (|NRTgetLocalIndex| x))
    (setq |$e|
      (third (|compMakeDeclaration| (list '|:| '$ target) mode |$e|))))

```

```

(unless |$insideCategoryPackageIfTrue|
  (if
    (and (pairp body) (eq (qcar body) '|add|)
      (pairp (qcdr body))
      (pairp (qcar (qcdr body)))
      (pairp (qcdr (qcdr body)))
      (eq (qcdr (qcdr (qcdr body))) nil)
      (pairp (qcar (qcdr (qcdr body))))
      (eq (qcar (qcar (qcdr (qcdr body)))) '|capsule|)
      (member (qcar (qcar (qcdr body))) '(|List| |Vector|))
      (equal (FindRep (qcdr (qcar (qcdr (qcdr body)))) (second body)))
    (setq |$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))
  (when (|isPackageFunction|)
    (setq |$functorLocalParameters|
      (cons nil
        (let (tmp1 result)
          (loop for i from 6 to (maxindex |$domainShell|) do
            (setq tmp1 (elt |$domainShell| i))
            (when
              (and (pairp tmp1) (pairp (qcdr tmp1)) (pairp (qcdr (qcdr tmp1))))
                (eq (qcdr (qcdr (qcdr tmp1))) nil)
                (pairp (qcar (qcdr (qcdr tmp1))))
                (eq (qcar (qcar (qcdr (qcdr tmp1)))) '|elt|)
                (pairp (qcdr (qcar (qcdr (qcdr tmp1))))))
                  (eq (qcar (qcdr (qcar (qcdr (qcdr tmp1)))) '$)
                    (pairp (qcdr (qcdr (qcar (qcdr (qcdr tmp1))))))
                    (eq (qcdr (qcdr (qcdr (qcar (qcdr (qcdr tmp1)))))) nil))
              (push nil result)))
          result))))
    (setq argPars (|makeFunctorArgumentParameters| arg1
      (cdr signaturep) (car signaturep)))
    (setq |$functorLocalParameters| arg1)
    (setq opp |$opl|)
    (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 |$lisplibKind|
        (if (and (pairp |$functorTarget|)
          (eq (qcar |$functorTarget|) 'category)
          (pairp (qcdr |$functorTarget|))
          (nequal (qcar (qcdr |$functorTarget|)) '|domain|))
          '|package|
          '|domain|))
      (setq |$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|) ''|infovec| (|getInfovecCode|))))))
      (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))))

```

5.2.35 defun augmentLisplibModemapsFromFunctor

```

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

```

— defun augmentLisplibModemapsFromFunctor —

```

(defun |augmentLisplibModemapsFromFuncor| (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))) argl)
              (|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|))))))

```

5.2.36 defun disallowNilAttribute

— defun disallowNilAttribute —

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

5.2.37 defun compFuncorBody

```
[bootStrapError p??]
[compOrCroak p401]
[/editfile p??]
[$NRTaddForm p??]
[$functorForm p??]
[$bootStrapMode p??]
```

— defun compFuncorBody —

```
(defun |compFuncorBody| (form mode env parForm)
  (declare (ignore parForm))
  (let (tt)
    (declare (special |$NRTaddForm| |$functorForm| |$bootStrapMode| /editfile))
    (if |$bootStrapMode|
      (list (|bootStrapError| |$functorForm| /editfile) mode env)
      (progn
        (setq tt (|compOrCroak| form mode env))
        (if (and (pairp form) (member (qcar form) '(|add| capsule)))
          tt
          (progn
            (setq |$NRTaddForm|
              (if (and (pairp form) (eq (qcar form) '|SubDomain|)
                (pairp (qcdr form)) (pairp (qcdr (qcdr form)))
                (eq (qcdr (qcdr (qcdr form))) nil))
              (qcar (qcdr form))
              form))
            tt))))))
```

5.2.38 defun reportOnFunctorCompilation

```
[displayMissingFunctions p177]
[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 '|%1|
         (append (|bright| " Cumulative Statistics for Constructor"
                          (list |$op|))))))
  (|sayBrightly|
   (cons " Time:"
         (append (|bright| (|normalizeStatAndStringify| (second |$functorStats|))
                          (list "seconds")))))
  (|sayBrightly| " ")
  '|done|)
```

5.2.39 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|))
                  (pairp (|getmode| (caar cvl) |$env|))
                  (eq (qcar (|getmode| (caar cvl) |$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)))))))))))))
```

—————

5.2.40 defun makeFunctorArgumentParameters

```
[assq p??]
[msubst p??]
[isCategoryForm p??]
[pairp p??]
[qcar p??]
[qcdr p??]
[genDomainViewList0 p180]
[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 (pairp s) (eq (qcar 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 (pairp s) (eq (qcar s) '|Join|))
          (|genDomainViewList0| a (rest s))
          (list (|genDomainView| a s '|getDomainView|)))
        (list a)))
    (findExtras (a target)
      (cond
        ((and (pairp target) (eq (qcar target) '|Join|))
         (reduce #'|union|
          (loop for x in (qcdr target)
            collect (findExtras a x))))
        ((and (pairp target) (eq (qcar target) 'category))
         (reduce #'|union|
          (loop for x in (qcdr (qcdr target))
            collect (findExtras1 a x))))))
    (findExtras1 (a x)
      (cond
        ((and (pairp x) (or (eq (qcar x) 'and) (eq (qcar x) 'or))
          (reduce #'|union|
            (loop for y in (rest x) collect (findExtras1 a y))))
         ((and (pairp x) (eq (qcar x) 'if)
          (pairp (qcdr x)) (pairp (qcdr (qcdr x)))
          (pairp (qcdr (qcdr (qcdr x))))
          (eq (qcdr (qcdr (qcdr (qcdr x)))) nil))
          (|union| (findExtrasP a (second x))
```

```

        (|union|
          (findExtras1 a (third x))
          (findExtras1 a (fourth x))))))
(findExtrasP (a x)
  (cond
    ((and (pairp x) (or (eq (qcar x) 'and)) (eq (qcar x) 'or))
      (reduce #'|union|
        (loop for y in (rest x) collect (findExtrasP a y))))
    ((and (pairp x) (eq (qcar x) '|has|)
      (pairp (qcdr x)) (pairp (qcdr (qcdr x)))
      (pairp (qcdr (qcdr (qcdr x))))
      (eq (qcdr (qcdr (qcdr (qcdr x)))) nil))
      (|union| (findExtrasP a (second x))
        (|union|
          (findExtras1 a (third x))
          (findExtras1 a (fourth x))))))
    ((and (pairp x) (eq (qcar x) '|has|)
      (pairp (qcdr x)) (equal (qcar (qcdr x)) a)
      (pairp (qcdr (qcdr x)))
      (eq (qcdr (qcdr (qcdr x))) nil)
      (pairp (qcar (qcdr (qcdr x))))
      (eq (qcar (qcar (qcdr (qcdr 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))))))

```

5.2.41 defun genDomainViewList0

[getDomainViewList p??]

— defun genDomainViewList0 —

```

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

```

5.2.42 defun genDomainViewList

```
[pairp p??]
[qcdr p??]
[isCategoryForm p??]
[genDomainView p181]
[genDomainViewList p181]
[$EmptyEnvironment p??]
```

— **defun genDomainViewList** —

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

—————

5.2.43 defun genDomainView

```
[genDomainOps p182]
[pairp p??]
[qcar p??]
[qcdr p??]
[augModemapsFromCategory p195]
[mkDomainConstructor p??]
[member p??]
[$e p??]
[$getDomainCode p??]
```

— **defun genDomainView** —

```
(defun |genDomainView| (name c viewSelector)
  (let (code cd)
    (declare (special |$getDomainCode| |$e|))
    (cond
      ((and (pairp c) (eq (qcar c) 'category) (pairp (qcdr c)))
       (|genDomainOps| name name c))
      (t
```

```

(setq code
  (if (and (pairp c) (eq (qcar c) '|SubsetCategory|)
        (pairp (qcdr c)) (pairp (qcdr (qcdr c)))
        (eq (qcdr (qcdr (qcdr 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)))

```

5.2.44 defun genDomainOps

```

[getOperationAlist p201]
[substNames p202]
[mkq p??]
[mkDomainConstructor p??]
[addModemap p??]
[$e p??]
[$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))

```

5.2.45 defun mkOpVec

```

[getPrincipalView p??]
[getOperationAlistFromLisplib p??]
[opOf p??]
[length p??]
[assq p??]
[assoc p??]
[pairp p??]
[qcar p??]
[qcdr p??]
[sublis p??]
[AssocBarGensym p??]
[msubst p??]
[$FormalMapVariableList p202]
[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 (pairp tmp1) (pairp (qcdr tmp1))
          (pairp (qcdr (qcdr tmp1))) (pairp (qcdr (qcdr (qcdr tmp1)))))

```

```

      (eq (qcdr (qcdr (qcdr (qcdr tmp1)))) nil)
      (eq (qcar (qcdr (qcdr (qcdr 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 (pairp tmp1) (pairp (qcdr tmp1)) (pairp (qcdr (qcdr tmp1)))
          (pairp (qcdr (qcdr (qcdr tmp1)))))
     (eq (qcdr (qcdr (qcdr (qcdr tmp1)))) nil)
     (eq (qcar (qcdr (qcdr (qcdr tmp1)))) 'elt))
    (setelt ops i (elt dom (second tmp1))))
  (t
   (setelt ops i (cons |Undef| (cons (list (elt dom 0) i) opSig))))))
ops))

```

5.2.46 defun compDefWhereClause

```

[pairp p??]
[qcar p??]
[qcdr p??]
[getmode p??]
[userError p??]
[concat p??]
[lassoc p??]
[pairList p??]
[union p??]
[listOfIdentifiersIn p??]
[delete p??]
[orderByDependency p??]
[assocleft p??]
[assocright p??]
[comp p403]
[$sigAlist p??]
[$predAlist p??]

```

— defun compDefWhereClause —

```

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

```

```

((atom x) x)
((and (pairp x) (eq (qcar x) '|:|) (pairp (qcdr x))
      (pairp (qcdr (qcdr x))) (eq (qcdr (qcdr (qcdr x))) nil))
 (setq |$sigAlist|
       (cons (cons (second x) (transformType (third x)))
             |$sigAlist|))
 x)
((and (pairp x) (eq (qcar x) '|Record|)) x)
(t
 (cons (first x)
       (loop for y in (rest x)
             collect (transformType y))))))
(removeSuchthat (x)
 (declare (special |$predAlist|))
 (if (and (pairp x) (eq (qcar x) '|\\|') (pairp (qcdr x))
          (pairp (qcdr (qcdr x))) (eq (qcdr (qcdr (qcdr x))) nil))
     (progn
      (setq |$predAlist| (cons (cons (second x) (third x)) |$predAlist|)
            (second x))
      x))
(fetchType (a x env form)
 (if x
     x
     (or (|getmode| a env)
         (|userError| (|concat|
                       "There is no mode for argument" a "of function" (first form))))))
(addSuchthat (x y)
 (let (p)
  (declare (special |$predAlist|))
  (if (setq p (lassoc x |$predAlist|)) (list '|\\| y p) y)))
)
(let (|$sigAlist| |$predAlist| form signature specialCases body sigList
      argList argSigAlist argDepAlist varList whereList formxx signaturex
      deform formx)
 (declare (special |$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)))

```

5.3 Functions to manipulate modemap

5.3.1 defun addDomain

```

[identp p??]
[qslessp p??]
[getDomainsInScope p188]
[domainMember p195]

```

```

[isLiteral p??]
[addNewDomain p190]
[getmode p??]
[paip p??]
[isCategoryForm p??]
[isFunctor p188]
[constructor? p??]
[member p??]
[unknownTypeError p??]

```

— 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 (paip tmp1) (eq (qcar tmp1) '|Mapping|)
                  (paip (qcdr 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))))

```

5.3.2 defun isFunctor

```
[opOf p??]
[identp p??]
[getdatabase p??]
[get p??]
[constructor? p??]
[updateCategoryFrameForCategory p121]
[updateCategoryFrameForConstructor p120]
[$CategoryFrame p??]
[$InteractiveMode p??]
```

— defun isFunctor —

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

5.3.3 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)))
```

5.3.4 defun putDomainsInScope

```
[getDomainsInScope p188]
[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))))
```

5.3.5 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))))))
```

5.3.6 defun addNewDomain

```
[augModemapsFromDomain p190]
```

— **defun addNewDomain** —

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

5.3.7 defun augModemapsFromDomain

```
[member p??]
[kar p??]
[getDomainsInScope p188]
[getdatabase p??]
[opOf p??]
[addNewDomain p190]
[listOrVectorElementNode p??]
[stripUnionTags p??]
[augModemapsFromDomain1 p191]
[$Category p??]
[$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 (|listOrVectorElementMode| name))
    (setq env (|addDomain| innerDom env)))
  (when (and (pairp name) (eq (qcar name) '|Union|))
    (dolist (d (|stripUnionTags| (qcdr name)))
      (setq env (|addDomain| d env))))
  (|augModemapsFromDomain1| name functorForm env))))

```

5.3.8 defun augModemapsFromDomain1

```

[getl p??]
[kar p??]
[addConstructorModemaps p192]
[getmode p??]
[augModemapsFromCategory p195]
[getmodeOrMapping p??]
[substituteCategoryArguments p192]
[stackMessage p??]

```

— defun augModemapsFromDomain1 —

```

(defun |augModemapsFromDomain1| (name functorForm env)
  (let (mappingForm categoryForm functArgTypes catform)
    (cond
      ((getl (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))))
```

5.3.9 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)))
```

5.3.10 defun addConstructorModemaps

```
[putDomainsInScope p189]
[getl p??]
[msubst p??]
[pairp p??]
[qcar p??]
[qcdr p??]
[addModemap p??]
[$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 (pairp opcode) (pairp (qcdr opcode))
             (pairp (qcdr (qcdr opcode)))
             (eq (qcdr (qcdr (qcdr opcode))) nil)
             (eq (qcar 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))

```

5.3.11 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))))))

```

5.3.12 defun getUniqueSignature

```

[getUniqueModemap p194]

```

— defun getUniqueSignature —

```

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

```

5.3.13 defun getUniqueModemap

```
[getModemapList p194]
[qslessp p??]
[stackWarning p??]
```

— defun getUniqueModemap —

```
(defun |getUniqueModemap| (op numOfArgs env)
  (let (mml)
    (cond
      ((eql 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))))
```

5.3.14 defun getModemapList

```
[pairp p??]
[qcar p??]
[qcdr p??]
[getModemapListFromDomain p195]
[nreverse0 p??]
[get p??]
```

— defun getModemapList —

```
(defun |getModemapList| (op numOfArgs env)
  (let (result)
    (cond
      ((and (pairp op) (eq (qcar op) '|elt|) (pairp (qcdr op))
       (pairp (qcdr (qcdr op))) (eq (qcdr (qcdr (qcdr 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)))))))
```

5.3.15 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))
```

5.3.16 defun domainMember

[modeEqual p??]

— defun domainMember —

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

5.3.17 defun augModemapsFromCategory

```
[evalAndSub p201]
[compilerMessage p??]
[putDomainsInScope p189]
[addModemapKnown p196]
[$base p??]
```

— defun augModemapsFromCategory —

```
(defun |augModemapsFromCategory| (domainName functorform categoryForm env)
  (let (tmp1 op sig cond fnsl)
    (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 fnset (caddr u))
(setq env (|addModemapKnown| op domainName sig cond fnset env)))
env))

```

5.3.18 defun addModemapKnown

```

[addModemap0 p196]
[$e p??]
[$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|)))

```

5.3.19 defun addModemap0

```

[pairp p??]
[qcar p??]
[addEltModemap p197]
[addModemap1 p198]
[$functorForm p??]

```

— defun addModemap0 —

```

(defun |addModemap0| (op mc sig pred fn env)
  (declare (special |$functorForm|))
  (cond
    ((and (pairp |$functorForm|)
      (eq (qcar |$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))))

```

5.3.20 defun addEltModemap

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

```

[qcar p??]
[qcdr p??]
[makeLiteral p??]
[addModemap1 p198]
[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|) (pairp sig))
        (setq tmp1 (reverse sig))
        (setq sel (qcar tmp1))
        (setq lt (nreverse (qcdr tmp1)))
        (cond
          ((stringp sel)
            (setq id (intern sel))
            (if |$insideCapsuleFunctionIfTrue|
              (setq |$e| (|makeLiteral| id |$e|))
              (setq env (|makeLiteral| id env)))
            (|addModemap1| op mc (append lt (list id)) pred fn env))
          (t (|addModemap1| op mc sig pred fn env))))
      ((and (eq op '|setelt|) (pairp sig))
        (setq tmp1 (reverse sig))
        (setq v (qcar tmp1))
        (setq sel (qcar (qcdr tmp1)))
        (setq lt (nreverse (qcdr (qcdr tmp1))))
        (cond
          ((stringp sel) (setq id (intern sel))
            (if |$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")))))

```

5.3.21 defun addModemap1

```

[msubst p??]
[getProplist p??]
[mkNewModemapList p198]
[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 '|Repl|) (setq sig (msubst '$ '|Repl| 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)))

```

5.3.22 defun mkNewModemapList

```

[member p??]
[assoc p??]
[pairp p??]
[qcar p??]
[qcdr p??]
[mergeModemap p199]
[nequal p??]
[nreverse0 p??]
[insertModemap p??]

```

```
[$InteractiveMode p??]
[$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))
            (pairp oldMap) (pairp (qcdr oldMap))
            (pairp (qcar (qcdr oldMap)))
            (pairp (qcdr (qcar (qcdr oldMap))))
            (eq (qcdr (qcar (qcdr oldMap)))) nil)
            (equal (qcar (qcdr (qcar (qcdr oldMap)))) fn))
        (setq opred (qcar (qcar (qcdr 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))))
```

—————

5.3.23 defun mergeModemap

```
[isSuperDomain p189]
[TruthP p??]
[$forceAdd p??]
```

— **defun mergeModemap** —

```
(defun |mergeModemap| (entry modemapList env)
```

```

(let (mc sig pred mcp sigp predp newmm mm)
(declare (special |$forceAdd|))
; break out the condition, signature, and predicate fields of the new entry
(setq mc (caar entry))
(setq sig (cdar entry))
(setq pred (caadr entry))
(seq
; walk across the successive tails of the modemap list
(do ((mmtail modemapList (cdr mmtail)))
((atom mmtail) nil)
(setq mcp (caaar mmtail))
(setq sigp (cdaaar mmtail))
(setq predp (caadar mmtail))
(cond
((or (equal mc mcp) (|isSuperDomain| mcp mc env))
; if this is a duplicate condition
(exit
(progn
(setq newmm nil)
(setq mm modemapList)
; copy the unique modemap terms
(loop while (not (eq mm mmtail)) do
(setq newmm (cons (car mm) newmm))
(setq mm (cdr mm)))
; if the conditions and signatures are equal
(when (and (equal mc mcp) (equal sig sigp))
; we only need one of these unless the conditions are hairy
(cond
((and (null |$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)))

```

5.3.24 defun evalAndSub

```
[isCategory p??]
[substNames p202]
[contained p??]
[put p??]
[get p??]
[getOperationAlist p201]
[$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|))))))
```

—

5.3.25 defun getOperationAlist

```
[getdatabase p??]
[isFunctor p188]
[systemError p??]
[compMakeCategoryObject p??]
[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 (|isFunction| 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|)))))

```

5.3.26 defvar \$FormalMapVariableList

— initvars —

```

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

```

5.3.27 defun substNames

```

[substq p??]
[isCategoryPackageName p??]
[eqsubstlist p??]
[nreverse0 p??]
[$FormalMapVariableList p202]

```

— 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))))

```

5.3.28 defun augModemapsFromCategoryRep

```

[evalAndSub p201]
[isCategory p??]
[compilerMessage p??]
[putDomainsInScope p189]
[assoc p??]
[msubst p??]
[addModemap p??]
[$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 (pairp u)
          (setq op (qcar u))
          (setq z (qcdr 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 fnsel 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 fnsel (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 fnsel env))))
env)))

```

—

5.4 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)))

```

5.4.1 defun compAdd plist

— postvars —

```

(eval-when (eval load)
 (setf (get '|add| 'special) 'compAdd))

```


5.4.2 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 p403]
[qcdr p??]
[qcar p??]
[compSubDomain1 p251]
[pairp p??]
[nreverse0 p??]
[NRTgetLocalIndex p??]
[compTuple2Record p??]
[compOrCroak p401]
[compCapsule p208]
[/editfile p??]
[$addForm p??]
[$addFormLhs p??]
[$EmptyMode p??]
[$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 |$bootStrapMode| t)
        (cond
          ((and (pairp |$addForm|) (eq (qcar |$addForm|) '|@Tuple|))
            (setq code nil))
          (t
            (setq tmp3 (|comp| |$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 (pairp |$addForm|) (eq (qcar |$addForm|) '|SubDomain|)
                (pairp (qcdr |$addForm|)) (pairp (qcdr (qcdr |$addForm|)))
                (eq (qcdr (qcdr (qcdr |$addForm|))) nil))
                (setq domainForm (second |$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 (pairp |$addForm|) (eq (qcar |$addForm|) '|@Tuple|))
                    (append (qcdr |$addForm|) |$packagesUsed|)
                    (cons |$addForm| |$packagesUsed|)))
                (setq |$NRTaddForm| |$addForm|)
                (setq tmp3
                  (cond
                    ((and (pairp |$addForm|) (eq (qcar |$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))))

```

5.4.3 defun compAtSign plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|@| 'special) 'compAtSign))

```

5.4.4 defun compAtSign

```

[addDomain p186]
[comp p403]
[coerce p??]

```

— 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))))

```

5.4.5 defun compCapsule plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'capsule 'special) '|compCapsule|))
```

5.4.6 defun compCapsule

```
[bootstrapError p??]
[compCapsuleInner p208]
[addDomain p186]
[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))))))
```

5.4.7 defun compCapsuleInner

```
[addInformation p??]
[compCapsuleItems p??]
[processFunctorOrPackage p??]
[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
          (|processFunctorOrPackage|
           |$form| |$signature| data localParList mode env)))
    (cons (mkpf (append |$getDomainCode| (list code)) 'progn) (list mode env))))
```

—————

5.4.8 defun compCase plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|case| 'special) '|compCase|))
```

—————

5.4.9 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 p186]
 [compCase1 p210]
 [coerce p??]

— 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))))
```

5.4.10 defun compCase1

```
[comp p403]
[getModemapList p194]
[nreverse0 p??]
[modeEqual p??]
[$Boolean p??]
[$EmptyMode p??]
```

— 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 |$EmptyMode| env))
      (setq xp (first tmp3))
      (setq mp (second tmp3))
      (setq ep (third tmp3))
      (when
        (setq u
          (dolist (modemap (|getModemapList| '|case| 2 ep) (nreverse0 tmp5))
            (setq map (first modemap))
            (when
              (and (pairp map) (pairp (qcdr map)) (pairp (qcdr (qcdr map)))
                (pairp (qcdr (qcdr (qcdr map))))
                (eq (qcdr (qcdr (qcdr (qcdr map)))) nil)
                (|modeEqual| (fourth map) mode)
                (|modeEqual| (third map) mp))
              (push (second modemap) tmp5))))
        (when
          (setq fn
            (dolist (onepair u tmp6)
              (when (first onepair) (setq tmp6 (or tmp6 (second onepair))))))
            (list (list '|call| fn xp |$Boolean| ep))))))
```

5.4.11 defun compCat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Record| 'special) '|compCat|))
```

5.4.12 defun compCat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Mapping| 'special) '|compCat|))
```

5.4.13 defun compCat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Union| 'special) '|compCat|))
```

5.4.14 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 (getl 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))))

```

5.4.15 defun compCategory plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'category 'special) '|compCategory|))

```

5.4.16 defun compCategory

```

[resolve p??]
[qcar p??]
[qcdr p??]
[compCategoryItem p??]
[mkExplicitCategoryFunction p??]
[systemErrorHere 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|))
        (pairp form)
        (eq (qcar form) 'category)

```



```

    (pairp (qcdr form)))
  (setq domainOrPackage (second form))
  (setq z (qcdr (qcdr form)))
  (setq |$sigList| nil)
  (setq |$atList| nil)
  (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"))))

```

5.4.17 defun compCoerce plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|::| 'special) '|compCoerce|))

```

5.4.18 defun compCoerce

```

[addDomain p186]
[getmode p??]
[compCoerce1 p214]
[coerce p??]

```

— 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 (pairp tmp1) (eq (qcar tmp1) '|Mapping|)

```

```

      (pairp (qcdr tmp1)) (eq (qcdr (qcdr tmp1)) nil)
      (pairp (qcar (qcdr tmp1)))
      (eq (qcar (qcar (qcdr tmp1))) '|UnionCategory|))
    (setq z (qcdr (qcar (qcdr tmp1))))
    (when
      (setq td
        (dolist (mode1 z tmp4)
          (setq tmp4 (or tmp4 (|compCoerce1| newform mode1 env))))))
      (|coerce| (list (car td) newmode (third td) mode))))))

```

5.4.19 defun compCoerce1

```

[comp p403]
[resolve p??]
[coerce p??]
[coerceByModemap p??]
[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))))))

```

5.4.20 defun compColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|:| 'special) '|compColon|))
```

5.4.21 defun compColon

```
;compColon([":",f,t],m,e) ==
; $insideExpressionIfTrue=true => compColonInside(f,m,e,t)
; --if inside an expression, ":" means to convert to m "on faith"
; $lhsOfColon: local:= f
; t:=
;   atom t and (t':= ASSOC(t,getDomainsInScope e)) => t'
;   isDomainForm(t,e) and not $insideCategoryIfTrue =>
;     (if not MEMBER(t,getDomainsInScope e) then e:= addDomain(t,e); t)
;   isDomainForm(t,e) or isCategoryForm(t,e) => t
;   t is ["Mapping",m',:r] => t
;   unknownTypeError t
;   t
; f is ["LISTOF",:l] =>
;   (for x in l repeat T:= [.,.,e]:= compColon([":",x,t],m,e); T)
; e:=
;   f is [op,:argl] and not (t is ["Mapping",:]) =>
;     --for MPOLY--replace parameters by formal arguments: RDJ 3/83
;     newTarget:= EQSUBSTLIST(take(#argl,$FormalMapVariableList),
;       [(x is [":",a,m] => a; x) for x in argl],t)
;     signature:=
;       ["Mapping",newTarget,:
;         [(x is [":",a,m] => m;
;           getmode(x,e) or systemErrorHere "compColonOld") for x in argl]]
;     put(op,"mode",signature,e)
;     put(f,"mode",t,e)
;   if not $bootStrapMode and $insideFunctorIfTrue and
;     makeCategoryForm(t,e) is [catform,e] then
;     e:= put(f,"value",[genSomeVariable(),t,$noEnv],e)
;   ["/throwAway",getmode(f,e),e]
```

```
[compColonInside p408]
[assoc p??]
[getDomainsInScope p188]
[isDomainForm p248]
[compColon member (vol5)]
```

```

[addDomain p186]
[isDomainForm p248]
[isCategoryForm p??]
[unknownTypeError p??]
[compColon p215]
[eqsubstlist p??]
[take p??]
[length p??]
[nreverse0 p??]
[getmode p??]
[systemErrorHere p??]
[put p??]
[makeCategoryForm p??]
[genSomeVariable p??]
[$lhsOfColon p??]
[$noEnv p??]
[$insideFunctorIfTrue p??]
[$bootStrapMode p??]
[$FormalMapVariableList p202]
[$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 (pairp argt) (eq (qcar argt) '|Mapping|)

```

```

      (progn
        (setq tmp2 (qcdr argt))
        (and (pairp tmp2)
          (progn
            (setq mprime (qcar tmp2))
            (setq r (qcdr 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 (pairp argf)
          (progn
            (setq op (qcar argf))
            (setq arg1 (qcdr argf))
            t)
          (null (and (pairp argt) (eq (qcar argt) '|Mapping|))))
        (setq newTarget
          (eqsubstlist (take (|#| arg1) |$FormalMapVariableList|)
            (dolist (x arg1 (nreverse0 g2))
              (setq g2
                (cons
                  (cond
                    ((and (pairp x) (eq (qcar x) '|:|)
                      (progn
                        (setq tmp2 (qcdr x))
                        (and (pairp tmp2)
                          (progn
                            (setq a (qcar tmp2))
                            (setq tmp3 (qcdr tmp2))
                            (and (pairp tmp3)
                              (eq (qcdr tmp3) nil)
                              (progn
                                (setq mode (qcar tmp3))
                                t))))))
                    t)
                  (t x))
                g2)))
              argt))
    (setq signature
      (cons '|Mapping|
        (cons newTarget

```

```

(dolist (x arg1 (nreverse0 g5))
  (setq g5
    (cons
      (cond
        ((and (pairp x) (eq (qcar x) '|:|)
          (progn
            (setq tmp2 (qcdr x))
            (and (pairp tmp2)
              (progn
                (setq a (qcar tmp2))
                (setq tmp3 (qcdr tmp2))
                (and (pairp tmp3)
                  (eq (qcdr tmp3) nil)
                  (progn
                    (setq mode (qcar 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 (pairp tmp2)
        (progn
          (setq catform (qcar tmp2))
          (setq tmp3 (qcdr tmp2))
          (and (pairp tmp3)
            (eq (qcdr tmp3) nil)
            (progn
              (setq env (qcar tmp3))
              t))))))
    (setq env
      (|put| argf '|value| (list (|genSomeVariable|) argt |$noEnv|)
        env))))
(list '|/throwAway| (|getmode| argf env) env )))))))

```

5.4.22 defun compCons plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'cons 'special) '|compCons|))
```

5.4.23 defun compCons

```
[compCons1 p219]
[compForm p414]
```

— defun compCons —

```
(defun |compCons| (form mode env)
  (or (|compCons1| form mode env) (|compForm| form mode env)))
```

5.4.24 defun compCons1

```
[comp p403]
[convert p411]
[pairp p??]
[qcar p??]
[qcdr p??]
[$EmptyMode p??]
```

— 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 (pairp my) (eq (qcar my) '|List|) (pairp (qcdr 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 (pairp (car ytp)) (eq (qcar (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))))))

```

5.4.25 defun compConstruct plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|construct| 'special) '|compConstruct|))

```

5.4.26 defun compConstruct

```

[modeIsAggregateOf p??]
[compList p413]
[convert p411]
[compForm p414]
[compVector p254]
[getDomainsInScope p188]

```

— 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))))))

```

5.4.27 defun compConstructorCategory plist

— postvars —

```

(eval-when (eval load)
 (setf (get '|ListCategory| 'special) '|compConstructorCategory|))

```

5.4.28 defun compConstructorCategory plist

— postvars —

```

(eval-when (eval load)
 (setf (get '|RecordCategory| 'special) '|compConstructorCategory|))

```

5.4.29 defun compConstructorCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|UnionCategory| 'special) '|compConstructorCategory|))
```

—————

5.4.30 defun compConstructorCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|VectorCategory| 'special) '|compConstructorCategory|))
```

—————

5.4.31 defun compConstructorCategory

```
[resolve p??]
[$Category p??]
```

— defun compConstructorCategory —

```
(defun |compConstructorCategory| (form mode env)
  (declare (special |$Category|))
  (list form (|resolve| |$Category| mode) env))
```

—————

5.4.32 defun compDefine plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'def 'special) '|compDefine|))
```

—————

5.4.33 defun compDefine

```
[compDefine1 p223]
[$tripleCache p??]
[$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)))
```

5.4.34 defun compDefine1

```
[macroExpand p144]
[isMacro p??]
[getSignatureFromMode p??]
[compDefine1 p223]
[compInternalFunction p??]
[compDefineAddSignature p141]
[compDefWhereClause p184]
[compDefineCategory p155]
[isDomainForm p248]
[getTargetFromRhs p142]
[giveFormalParametersValues p143]
[addEmptyCapsuleIfNecessary p142]
[compDefineFunctor p166]
[stackAndThrow p??]
[strconc p??]
[getAbbreviation p??]
[length p??]
[compDefineCapsuleFunction p??]
[$insideExpressionIfTrue p??]
[$formalArgList p??]
[$form p??]
```

```

[$op p??]
[$prefix p??]
[$insideFunctorIfTrue p??]
[$Category p??]
[$insideCategoryIfTrue p??]
[$insideCapsuleFunctionIfTrue p??]
[$ConstructorNames p??]
[$NoValueMode p??]
[$EmptyMode p??]
[$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 |$NoValueMode|))))
      (list lhs mode (|put| (car lhs) '|macro| rhs env)))
      ((and (null (car signature)) (consp rhs)
            (null (member (qcar rhs) |$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|))))))

```

5.4.35 defun compElt plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|elt| 'special) '|compElt|))

```

5.4.36 defun compElt

```

[compForm p414]
[isDomainForm p248]
[addDomain p186]
[getModemapListFromDomain p195]
[length p??]
[stackMessage p??]
[stackWarning p??]
[convert p411]
[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 (pairp form) (eq (qcar form) '|elt|)
                  (pairp (qcdr form)) (pairp (qcdr (qcdr form)))
                  (eq (qcdr (qcdr (qcdr 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
              ((eql 1 n) (elt mmList 0))
              ((eql 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 (pairp val) (eq (qcar val) '|elt|))))
              (setq val (|genDeltaEntry| (cons (|opOf| anOp) modemap)))
              (|convert| (list (list '|call| val) (second sig) env) mode))))
          (t
            (|compForm| form mode env))))))
```

5.4.37 defun compExit plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|exit| 'special) '|compExit|))
```

5.4.38 defun compExit

```
[comp p403]
[modifyModeStack p429]
[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))))))))))
```

5.4.39 defun compHas plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'has| 'special) '|compHas|))
```

—————

5.4.40 defun compHas

```
[chaseInferences p??]
[compHasFormat p??]
[coerce p??]
[$e p??]
```

— defun compHas —

```
(defun |compHas| (pred mode |$e|)
  (declare (special |$e|))
  (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)))
```

—————

5.4.41 defun compIf plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'if 'special) '|compIf|))
```

—————

5.4.42 defun compIf

```
[canReturn p??]
[intersectionEnvironment p??]
```



```
[compBoolean p??]
[compFromIf p??]
[resolve p??]
[coerce p??]
[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 (|quotify| (first xbp)) (|quotify| xc)))
              (setq returnEnv (environ (third xbp) Ec (first xbp) xc env))
              (list x mc returnEnv))))))))))
```

5.4.43 defun compImport plist

— **postvars** —

```
(eval-when (eval load)
```

```
(setf (get '|import| 'special) '|compImport|))
```

5.4.44 defun compImport

```
[addDomain p186]
[$NoValueMode p??]
```

— 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))
```

5.4.45 defun compIs plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|is| 'special) '|compIs|))
```

5.4.46 defun compIs

```
[comp p403]
[coerce p??]
[$Boolean p??]
[$EmptyMode p??]
```

— 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))))

```

5.4.47 defun compJoin plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|Join| 'special) '|compJoin|))

```

5.4.48 defun compJoin

```

[nreverse0 p??]
[compForMode p??]
[stackSemanticError p??]
[nreverse0 p??]
[isCategoryForm p??]
[union p??]
[compJoin,getParms p??]
[pairp p??]
[qcar p??]
[qcdr p??]
[wrapDomainSub p??]
[convert p411]
[$Category p??]

```

— defun compJoin —

```

(defun |compJoin| (form mode env)
  (labels (

```

```

(getParms (y env)
  (cond
    ((atom y)
      (when (|isDomainForm| y env) (list y)))
    ((and (pairp y) (eq (qcar y) 'length)
      (pairp (qcdr y)) (eq (qcdr (qcdr 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))
                (x)
                ((and (pairp x) (eq (qcar x) '|DomainSubstitutionMacro|)
                  (pairp (qcdr x)) (pairp (qcdr (qcdr x)))
                  (eq (qcdr (qcdr (qcdr x))) nil))
                  (setq pl (second x))
                  (setq body (third x))
                  (setq parameters (|union| pl parameters)) body)
                ((and (pairp x) (eq (qcar 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))))))

```

5.4.49 defun compLambda plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|+>| 'special) '|compLambda|))
```

5.4.50 defun compLambda

```
[qcar p??]
[qcdr p??]
[argsToSig p428]
[compAtSign p207]
[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 (pairp v1) (eq (qcar v1) '|:|))
        (progn
          (setq tmp1 (qcdr v1))
          (and (pairp tmp1)
            (progn
              (setq args (qcar tmp1))
              (setq tmp2 (qcdr tmp1))
              (and (pairp tmp2)
                (eq (qcdr tmp2) nil)
                (progn
                  (setq target (qcar tmp2))
                  t))))))
        (when (and (pairp args) (eq (qcar args) '|@Tuple|))
          (setq args (qcdr 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 )))))

```

5.4.51 defun compLeave plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|leave| 'special) '|compLeave|))

```

5.4.52 defun compLeave

```

[comp p403]
[modifyModeStack p429]
[$exitModeStack p??]
[$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 ))))

```

5.4.53 defun compMacro plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'mdef 'special) '|compMacro|))
```

—————

5.4.54 defun compMacro

```
[qcar p??]
[formatUnabbreviated p??]
[sayBrightly p??]
[put p??]
[macroExpand p144]
[$macroIfTrue p??]
[$NoValueMode p??]
[$EmptyMode p??]
```

— 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 (pairp rhs) (eq (qcar rhs) 'category))
         (list "-- the constructor category"))
        ((and (pairp rhs) (eq (qcar rhs) '|Join|))
         (list "-- the constructor category"))
        ((and (pairp rhs) (eq (qcar rhs) 'capsule))
         (list "-- the constructor capsule"))
        ((and (pairp rhs) (eq (qcar 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))))))

```

5.4.55 defun compPretend plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|pretend| 'special) '|compPretend|))

```

5.4.56 defun compPretend

```

[addDomain p186]
[comp p403]
[opOf p??]
[nequal p??]
[stackSemanticError p??]
[stackWarning p??]
[$newCompilerUnionFlag p??]
[$EmptyMode p??]

```

— 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))))))

```

5.4.57 defun compQuote plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'quote 'special) '|compQuote|))

```

5.4.58 defun compQuote

— defun compQuote —

```

(defun |compQuote| (form mode env)
  (list form mode env))

```

5.4.59 defun compReduce plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'reduce 'special) '|compReduce|))

```

5.4.60 defun compReduce

```
[compReduce1 p238]
[$formalArgList p??]
```

— defun compReduce —

```
(defun |compReduce| (form mode env)
  (declare (special |$formalArgList|))
  (|compReduce1| form mode env |$formalArgList|))
```

5.4.61 defun compReduce1

```
[systemError p??]
[nreverse0 p??]
[compIterator p??]
[comp p403]
[parseTran p101]
[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 |$e| env)
  (setq itl
    (dolist (x itl (nreverse0 tmp2))
      (setq tmp1 (or (|comp|Iterator| x |$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 )))))))))))
```

5.4.62 defun compRepeatOrCollect plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'collect 'special) '|compRepeatOrCollect|))
```

5.4.63 defun compRepeatOrCollect plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'repeat 'special) '|compRepeatOrCollect|))
```

5.4.64 defun compRepeatOrCollect

```
[length p??]
[compIterator p??]
[modeIsAggregateOf p??]
[stackMessage p??]
[compOrCroak p401]
[comp p403]
[msubst p??]
[coerceExit p??]
[ p??]
[ p??]
[$until p??]
[$Boolean p??]
[$NoValueMode p??]
[$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)))

```

5.4.65 defun compReturn plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'return 'special) '|compReturn|))

```

5.4.66 defun compReturn

```

[stackSemanticError p??]
[nequal p??]
[userError p??]
[resolve p??]
[comp p403]
[modifyModeStack p429]

```

```
[$exitModeStack p??]
[$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)))))))
```

—————

5.4.67 defun compSeq plist

— **postvars** —

```
(eval-when (eval load)
  (setf (get 'seq 'special) '|compSeq|))
```

—————

5.4.68 defun compSeq

```
[compSeq1 p244]
[$exitModeStack p??]
```

— defun compSeq —

```
(defun |compSeq| (form mode env)
  (declare (special |$exitModeStack|))
  (|compSeq1| (cdr form) (cons mode |$exitModeStack|) env))
```

—————

5.4.69 defun compSeq1

```
[nreverse0 p??]
[compSeqItem p245]
[mkq p??]
[replaceExitEtc p??]
[$exitModeStack p??]
[$insideExpressionIfTrue p??]
[$finalEnv p??]
[$NoValueMode p??]
```

— 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|))))
```

5.4.70 defun compSeqItem

[comp p403]
[macroExpand p144]

— defun compSeqItem —

```
(defun |compSeqItem| (form mode env)
  (|comp| (|macroExpand| form env) mode env))
```

5.4.71 defun compSetq plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'let 'special) '|compSetq|))
```

5.4.72 defun compSetq plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'setq 'special) '|compSetq|))
```

5.4.73 defun compSetq

[compSetq1 p246]

— defun compSetq —

```
(defun |compSetq| (form mode env)
  (|compSetq1| (second form) (third form) mode env))
```

5.4.74 defun compSetq1

```
[setqSingle p247]
[compSetq1 identp (vol5)]
[compMakeDeclaration p429]
[compSetq p245]
[qcar p??]
[qcdr p??]
[setqMultiple p??]
[setqSetelt p246]
[$EmptyMode p??]
```

— defun compSetq1 —

```
(defun |compSetq1| (form val mode env)
  (let (x y ep op z)
    (declare (special |$EmptyMode|))
    (cond
      ((identp form) (|setqSingle| form val mode env))
      ((and (pairp form) (eq (qcar form) '|:|') (pairp (qcdr form))
        (pairp (qcdr (qcdr form))) (eq (qcdr (qcdr (qcdr form))) nil)))
        (setq x (second form))
        (setq y (third form))
        (setq ep (third (|compMakeDeclaration| form |$EmptyMode| env)))
        (|compSetq| (list 'let x val) mode ep))
      ((pairp form)
        (setq op (qcar form))
        (setq z (qcdr 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)))))))
```

5.4.75 defun setqSetelt

```
[comp p403]
```

— defun setqSetelt —

```
(defun |setqSetelt| (form val mode env)
  (|comp| (cons '|setelt| (cons (car form) (append (cdr form) (list val))))
    mode env))
```

5.4.76 defun setqSingle

```

[setqSingle getProplist (vol5)]
[getmode p??]
[get p??]
[nequal p??]
[maxSuperType p??]
[comp p403]
[getmode p??]
[assignError p??]
[convert p411]
[setqSingle identp (vol5)]
[profileRecord p??]
[consProplistOf p??]
[removeEnv p??]
[setqSingle addBinding (vol5)]
[isDomainForm p248]
[isDomainInScope p??]
[stackWarning p??]
[augModemapsFromDomain1 p191]
[NRTassocIndex p??]
[isDomainForm p248]
[outputComp p??]
[$insideSetqSingleIfTrue p??]
[$QuickLet p??]
[$form p??]
[$profileCompiler p??]
[$EmptyMode p??]
[$NoValueMode p??]

```

— 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 |$form|)) '|arguments| '|locals|))
    (|profileRecord| key form (second td)))
  (setq newProplist
    (|consProplistOf| form currentProplist '|value|
      (|removeEnv| (cons val (cdr td)))))
  (setq ep (if (pairp form) ep (|addBinding| form newProplist ep)))
  (when (|isDomainForm| val ep)
    (when (|isDomainInScope| form ep)
      (|stackWarning|
        (list '|domain valued variable| '|%b| form '|%d|
              '|has been reassigned within its scope| )))
      (setq ep (|augModemapsFromDomain1| form val ep)))
  (if (setq k (|NRTassocIndex| form))
    (setq newform (list 'setelt '$ 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))))

```

5.4.77 defun isDomainForm

```

[kar p??]
[pairp p??]
[qcar p??]
[qcdr p??]

```

```
[isFunctor p188]
[isCategoryForm p??]
[isDomainConstructorForm p249]
[$SpecialDomainNames p??]
```

— **defun isDomainForm** —

```
(defun |isDomainForm| (d env)
  (declare (special |$SpecialDomainNames|))
  (or (member (kar d) |$SpecialDomainNames|) (|isFunctor| d)
      (and (progn
              (setq tmp1 (|getmodel| d env))
              (and (pairp tmp1) (eq (qcar tmp1) '|Mapping|) (pairp (qcdr tmp1))))
          (|isCategoryForm| (qcar (qcdr tmp1)) env))
       (|isCategoryForm| (|getmodel| d env) env)
       (|isDomainConstructorForm| d env)))
```

5.4.78 defun isDomainConstructorForm

```
[pairp p??]
[qcar p??]
[qcdr p??]
[isCategoryForm p??]
[eqsubstlist p??]
[$FormalMapVariableList p202]
```

— **defun isDomainConstructorForm** —

```
(defun |isDomainConstructorForm| (d env)
  (let (u)
    (declare (special |$FormalMapVariableList|))
    (when
      (and (pairp d)
           (setq u (|get| (qcar d) '|value| env))
           (pairp u)
           (pairp (qcdr u))
           (pairp (qcar (qcdr u)))
           (eq (qcar (qcar (qcdr u))) '|Mapping|)
           (pairp (qcdr (qcar (qcdr u)))))
      (|isCategoryForm|
        (eqsubstlist (rest d) |$FormalMapVariableList| (cadadr u)) env))))
```

5.4.79 defun compString plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|String| 'special) '|compString|))
```

—————

5.4.80 defun compString

```
[resolve p??]
[$StringCategory p??]
```

— defun compString —

```
(defun |compString| (form mode env)
  (declare (special |$StringCategory|))
  (list form (|resolve| |$StringCategory| mode) env))
```

—————

5.4.81 defun compSubDomain plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|SubDomain| 'special) '|compSubDomain|))
```

—————

5.4.82 defun compSubDomain

```
[compSubDomain1 p251]
[compCapsule p208]
[$addFormLhs p??]
[$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)))
```

5.4.83 defun compSubDomain1

```
[compMakeDeclaration p429]
[addDomain p186]
[compOrCroak p401]
[stackSemanticError p??]
[lispize p??]
[evalAndRwriteLispForm p154]
[$CategoryFrame p??]
[$op p??]
[$lisplibSuperDomain p??]
[$Boolean p??]
[$EmptyMode p??]
```

— 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)))

```

5.4.84 defun compSubsetCategory plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|SubsetCategory| 'special) '|compSubsetCategory|))

```

5.4.85 defun compSubsetCategory

TPDHERE: See LocalAlgebra for an example call [put p??]

```

[comp p403]
[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)))

```

5.4.86 defun compSuchthat plist

— postvars —

```

(eval-when (eval load)
  (setf (get '\| 'special) '|compSuchthat|))

```

5.4.87 defun compSuchthat

```

[comp p403]
[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))))))

```

5.4.88 defun compVector plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'vector 'special) '|compVector|))
```

—————

5.4.89 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 p403]
[$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))))))
```

—————

5.4.90 defun compWhere plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|where| 'special) '|compWhere|))
```

—

5.4.91 defun compWhere

```
[comp p403]
[macroExpand p144]
[deltaContour p??]
[addContour p??]
[$insideExpressionIfTrue p??]
[$insideWhereIfTrue p??]
[$EmptyMode p??]
```

— 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))))))
```

Chapter 6

Post Transformers

6.1 Direct called postparse routines

6.1.1 defun postTransform

[postTran p258]
[postTransform identp (vol5)]
[postTransformCheck p261]
[aplTran p293]

— defun postTransform —

```
(defun |postTransform| (y)
  (let (x tmp1 tmp2 tmp3 tmp4 tmp5 tt l u)
    (setq x y)
    (setq u (|postTran| x))
    (when
      (and (pairp u) (eq (qcar u) '|@Tuple|))
      (progn
        (setq tmp1 (qcdr u))
        (and (pairp tmp1)
              (progn (setq tmp2 (reverse tmp1)) t)
              (pairp tmp2)
              (progn
                (setq tmp3 (qcar tmp2))
                (and (pairp tmp3)
                      (eq (qcar tmp3) '|:|')
                      (progn
                        (setq tmp4 (qcdr tmp3))
                        (and (pairp tmp4)
                              (progn
                                (setq y (qcar tmp4))
```



```

      (cons (|postTran| op) (cdr (|postTran| (cons b (cdr x))))))
    ((and (pairp op) (eq (qcar 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)
    ((eql x 0) '(|Zero|))
    ((eql x 1) '(|One|))
    ((eq x t) 't$)
    ((and (identp x) (getdatabase x 'niladic)) (list x))
    (t x)))

```

6.1.5 defun postTranList

[postTran p258]

— defun postTranList —

```
(defun |postTranList| (x)
  (loop for y in x collect (|postTran| y)))
```

—————

6.1.6 defun postScriptsForm

[getScriptName p297]

[length p??]

[postTranScripts p260]

— defun postScriptsForm —

```
(defun |postScriptsForm| (form argl)
  (let ((op (second form)) (a (third form)))
    (cons (|getScriptName| op a (|#| argl))
          (append (|postTranScripts| a) argl))))
```

—————

6.1.7 defun postTranScripts

[postTranScripts p260]

[postTran p258]

— defun postTranScripts —

```
(defun |postTranScripts| (a)
  (labels (
    (fn (x)
      (if (and (pairp x) (eq (qcar x) '|@Tuple|))
          (qcdr x)
          (list x))))
    (let (tmp1 tmp2 tmp3)
      (cond
        ((and (pairp a) (eq (qcar a) '|PrefixSC|))
         (progn
          (setq tmp1 (qcdr a))
```



```

      (and (pairp tmp1) (eq (qcdr tmp1) nil))))
    (|postTranScripts| (qcar tmp1)))
  ((and (pairp a) (eq (qcar a) '|;|))
   (dolist (y (qcdr a) tmp2)
    (setq tmp2 (append tmp2 (|postTranScripts| y)))))
  ((and (pairp a) (eq (qcar a) '|,|))
   (dolist (y (qcdr a) tmp3)
    (setq tmp3 (append tmp3 (fn (|postTran| y))))))
  (t (list (|postTran| a)))))

```

6.1.8 defun postTransformCheck

[postcheck p261]
 [\$defOp p??]

— defun postTransformCheck —

```

(defun |postTransformCheck| (x)
  (let (|$defOp|)
    (declare (special |$defOp|))
    (setq |$defOp| nil)
    (|postcheck| x)))

```

6.1.9 defun postcheck

[setDefOp p292]
 [postcheck p261]

— defun postcheck —

```

(defun |postcheck| (x)
  (cond
    ((atom x) nil)
    ((and (pairp x) (eq (qcar x) 'def) (pairp (qcdr x)))
     (|setDefOp| (qcar (qcdr x)))
     (|postcheck| (qcdr (qcdr x))))
    ((and (pairp x) (eq (qcar x) 'quote)) nil)
    (t (|postcheck| (car x)) (|postcheck| (cdr x)))))

```

6.1.10 defun postError

```
[nequal p??]
[bumperrorcount p363]
[$defOp p??]
[$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 p260]
[internal p??]
[postTran p258]
[postError p262]
[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 (pairp arglp) (eq (qcdr arglp) nil) (pairp (qcar arglp))
      (eq (qcar (qcar arglp)) '|@Tuple|))
      (|#| (qcdr (qcar arglp))))
    (t 1)))
(internal '* (princ-to-string numOfArgs) (pname op)))
(cons opp arglp))
((and (pairp op) (eq (qcar op) '|Scripts|))
  (append (|postTran| op) (|postTranList| argl)))
(t
  (setq u (|postTranList| u))
  (cond
    ((and (pairp u) (pairp (qcar u)) (eq (qcar (qcar 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 (pairp x) (pairp (qcdr x)) (eq (qcdr (qcdr x)) nil)
      (pairp (qcar (qcdr x))) (eq (qcar (qcar (qcdr x))) '|@Tuple|))
      (cons (car x) (qcdr (qcar (qcdr 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 defun postAdd plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|add| '|postTran|) '|postAdd|))
```

—————

6.2.2 defun postAdd

[postTran p258]
[postCapsule p265]

— 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 p367]
 [postBlockItem p266]
 [postBlockItemList p265]
 [postFlatten p274]

— defun postCapsule —

```
(defun |postCapsule| (x)
  (let (op)
    (cond
      ((null (and (pairp x) (progn (setq op (qcar 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 p266]

— defun postBlockItemList —

```
(defun |postBlockItemList| (args)
  (let (result)
    (dolist (item args (nreverse result))
      (push (|postBlockItem| item) result))))
```

6.2.5 defun postBlockItem

[postTran p258]

— defun postBlockItem —

```
(defun |postBlockItem| (x)
  (let ((tmp1 t) tmp2 y tt z)
    (setq x (|postTran| x))
    (if
      (and (pairp x) (eq (qcar x) '|@Tuple|))
      (progn
        (and (pairp (qcdr x))
          (progn (setq tmp2 (reverse (qcdr x))) t)
          (pairp tmp2)
          (progn
            (and (pairp (qcar tmp2)) (eq (qcar (qcar tmp2)) '|:|))
            (progn
              (and (pairp (qcdr (qcar tmp2)))
                (progn
                  (setq y (qcar (qcdr (qcar tmp2))))
                  (and (pairp (qcdr (qcdr (qcar tmp2))))
                    (eq (qcdr (qcdr (qcdr (qcar tmp2)))) nil)
                    (progn (setq tt (qcar (qcdr (qcdr (qcar tmp2)))) t))))))
              (progn (setq z (qcdr 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)))
    x)))
```

—

6.2.6 defun postAtSign plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|@'|postTran|) '|postAtSign|))
```

—

6.2.7 defun postAtSign

[postTran p258]
[postType p267]

— defun postAtSign —

```
(defun |postAtSign| (arg)
  (cons '@ (cons (|postTran| (second arg)) (|postType| (third arg)))))
```

—————

6.2.8 defun postType

[postTran p258]
[unTuple p301]

— defun postType —

```
(defun |postType| (typ)
  (let (source target)
    (cond
      ((and (pairp typ) (eq (qcar typ) '->) (pairp (qcdr typ))
        (pairp (qcdr (qcdr typ))) (eq (qcdr (qcdr (qcdr typ))) nil))
       (setq source (qcar (qcdr typ)))
       (setq target (qcar (qcdr (qcdr typ))))
      (cond
        ((eq source '|constant|)
         (list (list (|postTran| target)) '|constant|))
        (t
         (list (cons '|Mapping|
                    (cons (|postTran| target)
                          (|unTuple| (|postTran| source))))))
      ((and (pairp typ) (eq (qcar typ) '->)
        (pairp (qcdr typ)) (eq (qcdr (qcdr typ)) nil))
       (list (list '|Mapping| (|postTran| (qcar (qcdr typ)))))
      (t (list (|postTran| typ)))))
```

—————

6.2.9 defun postBigFloat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|:BF:| 'postTran|) '|postBigFloat|))
```

6.2.10 defun postBigFloat

```
[postTran p258]
[$boot p??]
[$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 defun postBlock plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Block| 'postTran|) '|postBlock|))
```

6.2.12 defun postBlock

```
[postBlockItemList p265]
[postTran p258]
```

— 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 defun postCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'category '|postTran|) '|postCategory|))
```

6.2.14 defun postCategory

```
[postTran p258]
[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 p270]
[tuple2List p368]
[postTranList p260]
```

— defun postCollect,finish —

```
(defun |postCollect,finish| (op itl y)
  (let (tmp2 tmp5 newBody)
    (cond
      ((and (pairp y) (eq (qcar y) '|:|)
            (pairp (qcdr y)) (eq (qcdr (qcdr y)) nil))
        (list 'reduce '|append| 0 (cons op (append itl (list (qcar (qcdr y)))))))
      ((and (pairp y) (eq (qcar y) '|Tuple|))
        (setq newBody
          (cond
            ((dolist (x (qcdr y) tmp2)
              (setq tmp2
                (or tmp2 (and (pairp x) (eq (qcar x) '|:|)
                              (pairp (qcdr x)) (eq (qcdr (qcdr x)) nil))))))
            (|postMakeCons| (qcdr y)))
          ((dolist (x (qcdr y) tmp5)
            (setq tmp5 (or tmp5 (and (pairp x) (eq (qcar x) 'segment))))))
            (|tuple2List| (qcdr y)))
            (t (cons '|construct| (|postTranList| (qcdr y))))))
        (list 'reduce '|append| 0 (cons op (append itl (list newBody))))))
      (t (cons op (append itl (list y))))))
```

6.2.16 defun postMakeCons

```
[postMakeCons p270]
[postTran p258]
```

— defun postMakeCons —

```
(defun |postMakeCons| (args)
  (let (a b)
    (cond
      ((null args) '|nil|)
      ((and (pairp args) (pairp (qcar args)) (eq (qcar (qcar args)) '|:|)
            (pairp (qcdr (qcar args))) (eq (qcdr (qcdr (qcar args))) nil))
        (setq a (qcar (qcdr (qcar args))))
```

```

(setq b (qcdr args))
(if b
  (list 'append| (|postTran| a) (|postMakeCons| b))
  (|postTran| a)))
(t (list '|cons| (|postTran| (car args)) (|postMakeCons| (cdr args))))))

```

6.2.17 defun postCollect plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'collect '|postTran|) '|postCollect|))

```

6.2.18 defun postCollect

```

[postCollect,finish p270]
[postCollect p271]
[postIteratorList p272]
[postTran p258]

```

— 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 (pairp x) (pairp (qcar x)) (eq (qcar (qcar x)) '|elt|)
            (pairp (qcdr (qcar x))) (pairp (qcdr (qcdr (qcar x))))
            (eq (qcdr (qcdr (qcdr (qcar x)))) nil)
            (eq (qcar (qcdr (qcdr (qcar x)))) '|construct|))
        (|postCollect|
         (cons (list '|elt| (qcar (qcdr (qcar x))) 'collect)
               (append m (list (cons '|construct| (qcdr x)))))))
      (t
       (setq itl (|postIteratorList| m))
       (setq x
        (if (and (pairp x) (eq (qcar x) '|construct|)

```

```

      (pairp (qcdr x)) (eq (qcdr (qcdr x)) nil))
    (qcar (qcdr x))
    x))
  (|postCollect,finish| constructOp itl (|postTran| x))))))

```

6.2.19 defun postIteratorList

[postTran p258]
 [postInSeq p280]
 [postIteratorList p272]

— defun postIteratorList —

```

(defun |postIteratorList| (args)
  (let (z p y u a b)
    (cond
      ((pairp args)
       (setq p (|postTran| (qcar args)))
       (setq z (qcdr args))
       (cond
         ((and (pairp p) (eq (qcar p) 'in) (pairp (qcdr p))
              (pairp (qcdr (qcdr p))) (eq (qcdr (qcdr (qcdr p))) nil))
          (setq y (qcar (qcdr p)))
          (setq u (qcar (qcdr (qcdr p))))
          (cond
            ((and (pairp u) (eq (qcar u) '|\\|') (pairp (qcdr u))
                 (pairp (qcdr (qcdr u))) (eq (qcdr (qcdr (qcdr u))) nil))
             (setq a (qcar (qcdr u)))
             (setq b (qcar (qcdr (qcdr 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 defun postColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '||| '|postTran|) '|postColon|))
```

6.2.21 defun postColon

```
[postTran p258]
[postType p267]
```

— defun postColon —

```
(defun |postColon| (u)
  (cond
    ((and (pairp u) (eq (qcar u) '|||)
      (pairp (qcdr u)) (eq (qcdr (qcdr u)) nil))
      (list '||| (|postTran| (qcar (qcdr u)))))
    ((and (pairp u) (eq (qcar u) '|||) (pairp (qcdr u))
      (pairp (qcdr (qcdr u))) (eq (qcdr (qcdr (qcdr u))) nil))
      (cons '||| (cons (|postTran| (second u)) (|postType| (third u)))))))
```

6.2.22 defun postColonColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '||::| '|postTran|) '|postColonColon|))
```

6.2.23 defun postColonColon

```
[postForm p262]
[$boot p??]
```

— defun postColonColon —

```
(defun |postColonColon| (u)
  (if (and $boot (pairp u) (eq (qcar u) '||::|) (pairp (qcdr u))
    (pairp (qcdr (qcdr u))) (eq (qcdr (qcdr (qcdr u))) nil))
```

```
(intern (princ-to-string (third u)) (second u))
(|postForm| u)))
```

6.2.24 defun postComma plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|,| '|postTran|) '|postComma|))
```

6.2.25 defun postComma

```
[postTuple p290]
[comma2Tuple p274]
```

— defun postComma —

```
(defun |postComma| (u)
  (|postTuple| (|comma2Tuple| u)))
```

6.2.26 defun comma2Tuple

```
[postFlatten p274]
```

— defun comma2Tuple —

```
(defun |comma2Tuple| (u)
  (cons '|@Tuple| (|postFlatten| u '|,|)))
```

6.2.27 defun postFlatten

```
[postFlatten p274]
```

— defun postFlatten —

```
(defun |postFlatten| (x op)
  (let (a b)
    (cond
      ((and (pairp x) (equal (qcar x) op) (pairp (qcdr x))
        (pairp (qcdr (qcdr x)))) (eq (qcdr (qcdr (qcdr x))) nil))
      (setq a (qcar (qcdr x)))
      (setq b (qcar (qcdr (qcdr x))))
      (append (|postFlatten| a op) (|postFlatten| b op)))
    (t (list x)))))
```

6.2.28 defun postConstruct plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|construct| '|postTran|) '|postConstruct|))
```

6.2.29 defun postConstruct

```
[comma2Tuple p274]
[postTranSegment p276]
[postMakeCons p270]
[tuple2List p368]
[postTranList p260]
[postTran p258]
```

— defun postConstruct —

```
(defun |postConstruct| (u)
  (let (b a tmp4 tmp7)
    (cond
      ((and (pairp u) (eq (qcar u) '|construct|)
        (pairp (qcdr u)) (eq (qcdr (qcdr u)) nil))
      (setq b (qcar (qcdr u)))
      (setq a
        (if (and (pairp b) (eq (qcar b) '|,|))
          (|comma2Tuple| b)
          b))
      (cond
        ((and (pairp a) (eq (qcar a) '|segment|) (pairp (qcdr a))
```

```

      (pairp (qcdr (qcdr a))) (eq (qcdr (qcdr (qcdr a))) nil))
    (list '|construct| (|postTranSegment| (second a) (third a)))
    ((and (pairp a) (eq (qcar a) '|@Tuple|))
     (cond
      ((dolist (x (qcdr a) tmp4)
       (setq tmp4
        (or tmp4
         (and (pairp x) (eq (qcar x) '|:|)
          (pairp (qcdr x)) (eq (qcdr (qcdr x)) nil))))))
      (|postMakeCons| (qcdr a)))
     ((dolist (x (qcdr a) tmp7)
      (setq tmp7 (or tmp7 (and (pairp x) (eq (qcar x) 'segment))))))
      (|tuple2List| (qcdr a)))
     (t (cons '|construct| (|postTranList| (qcdr a)))))
    (t (list '|construct| (|postTran| a)))))
  (t u)))

```

6.2.30 defun postTranSegment

[postTran p258]

— defun postTranSegment —

```

(defun |postTranSegment| (p q)
  (list 'segment (|postTran| p) (when q (|postTran| q))))

```

6.2.31 defun postDef plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|==| '|postTran|) '|postDef|))

```

6.2.32 defun postDef

[postMDef p283]

[recordHeaderDocumentation p??]


```
[nequal p??]
[postTran p258]
[postDefArgs p278]
[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 (pairp lhs) (eq (qcar lhs) '|macro|)
            (pairp (qcdr lhs)) (eq (qcdr (qcdr 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 (pairp lhs) (eq (qcar lhs) '|:|') (cdr lhs) (list lhs nil)))
            (setf (first tmp1) lhs)
            (setf (second tmp1) targetType)
            (when (and (null |$InteractiveMode|) (atom form)) (setf form (list form)))
            (setf newLhs
              (if (atom form)
                  form
                  (progn
                    (setf tmp1
                      (dolist (x form (nreverse0 tmp4))
                        (push
                          (if (and (pairp x) (eq (qcar x) '|:|') (pairp (qcdr x))
                                  (pairp (qcdr (qcdr x))) (eq (qcdr (qcdr (qcdr x))) nil))
                              (second x)
                              x)
                          tmp4)))
                    (setf op (car tmp1))
                    (setf arg1 (cdr tmp1))
```

```

      (cons op (|postDefArgs| arg1))))))
(setq argTypeList
  (unless (atom form)
    (dolist (x (cdr form) (nreverse0 tmp6))
      (push
        (when (and (pairp x) (eq (qcar x) '|:|) (pairp (qcdr x))
          (pairp (qcdr (qcdr x))) (eq (qcdr (qcdr (qcdr 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 p262]
 [postDefArgs p278]

— defun postDefArgs —

```

(defun |postDefArgs| (args)
  (let (a b)
    (cond
      ((null args) args)
      ((and (pairp args) (pairp (qcar args)) (eq (qcar (qcar args)) '|:|)
        (pairp (qcdr (qcar args))) (eq (qcdr (qcdr (qcar args))) nil))
        (setq a (qcar (qcdr (qcar args))))
        (setq b (qcdr args))
        (cond
          (b (|postError|
            (list " Argument" a "of indefinite length must be last"))
            ((or (atom a) (and (pairp a) (eq (qcar a) 'quote)))
              a)
            (t
              (|postError|
                (list " Argument" a "of indefinite length must be a name")))))
          (t (cons (car args) (|postDefArgs| (cdr args)))))))
  (t (cons (car args) (|postDefArgs| (cdr args))))))

```

6.2.34 defun postExit plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|=>' '|postTran|) '|postExit|))
```

—————

6.2.35 defun postExit

[postTran p258]

— defun postExit —

```
(defun |postExit| (arg)
  (list 'if (|postTran| (second arg))
        (list '|exit| (|postTran| (third arg))
              '|noBranch|))
```

—————

6.2.36 defun postIf plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|if|' '|postTran|) '|postIf|))
```

—————

6.2.37 defun postIf

```
[nreverse0 p??]
[postTran p258]
[$boot p??]
```

— defun postIf —

```
(defun |postIf| (arg)
```

```

(let (tmp1)
  (if (null (and (pairp arg) (eq (qcar arg) '|if|)))
    arg
    (cons 'if
      (dolist (x (qcdr arg) (nreverse0 tmp1))
        (push
          (if (and (null (setq x (|postTran| x))) (null $boot)) '|noBranch| x)
          tmp1))))))

```

6.2.38 defun postin plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|in| '|postTran|) '|postin|))

```

6.2.39 defun postin

```

[systemErrorHere p??]
[postTran p258]
[postInSeq p280]

```

— defun postin —

```

(defun |postin| (arg)
  (if (null (and (pairp arg) (eq (qcar arg) '|in|) (pairp (qcdr arg))
                (pairp (qcdr (qcdr arg))) (eq (qcdr (qcdr (qcdr arg))) nil)))
    (|systemErrorHere| "postin")
    (list '|in| (|postTran| (second arg)) (|postInSeq| (third arg)))))

```

6.2.40 defun postInSeq

```

[postTranSegment p276]
[tuple2List p368]
[postTran p258]

```

— defun postInSeq —

```
(defun |postInSeq| (seq)
  (cond
    ((and (pairp seq) (eq (qcar seq) 'segment) (pairp (qcdr seq))
      (pairp (qcdr (qcdr seq))) (eq (qcdr (qcdr (qcdr seq))) nil))
      (|postTranSegment| (second seq) (third seq)))
    ((and (pairp seq) (eq (qcar seq) '@Tuple|))
      (|tuple2List| (qcdr seq)))
    (t (|postTran| seq))))
```

6.2.41 defun postIn plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'in '|postTran|) '|postIn|))
```

6.2.42 defun postIn

```
[systemErrorHere p??]
[postTran p258]
[postInSeq p280]
```

— defun postIn —

```
(defun |postIn| (arg)
  (if (null (and (pairp arg) (eq (qcar arg) 'in) (pairp (qcdr arg))
    (pairp (qcdr (qcdr arg))) (eq (qcdr (qcdr (qcdr arg))) nil)))
      (|systemErrorHere| "postIn")
      (list 'in (|postTran| (second arg)) (|postInSeq| (third arg)))))
```

6.2.43 defun postJoin plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Join| '|postTran|) '|postJoin|))
```

6.2.44 defun postJoin

```
[postTran p258]
[postTranList p260]
```

— defun postJoin —

```
(defun |postJoin| (arg)
  (let (a l al)
    (setq a (|postTran| (cadr arg)))
    (setq l (|postTranList| (cddr arg)))
    (when (and (pairp l) (eq (qcdr l) nil) (pairp (qcar l))
              (member (qcar (qcar l)) '(attribute signature))))
    (setq l (list (list 'category (qcar l)))))
    (setq al (if (and (pairp a) (eq (qcar a) '|@Tuple|)) (qcdr a) (list a)))
    (cons '|Join| (append al l))))
```

6.2.45 defun postMapping plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|->| '|postTran|) '|postMapping|))
```

6.2.46 defun postMapping

```
[postTran p258]
[unTuple p301]
```

— defun postMapping —

```
(defun |postMapping| (u)
  (if (null (and (pairp u) (eq (qcar u) '|->|)) (pairp (qcdr u))
```

```

      (pairp (qcdr (qcdr u))) (eq (qcdr (qcdr (qcdr u))) nil)))
u
(cons '|Mapping|
  (cons (|postTran| (third u))
    (|unTuple| (|postTran| (second u))))))

```

6.2.47 defun postMDef plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|==>' '|postTran|) '|postMDef|))

```

6.2.48 defun postMDef

```

[postTran p258]
[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 (pairp lhs) (eq (qcar 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 (pairp x) (eq (qcar x) '|:|) (pairp (qcdr x))) (second x) x)
      tmp4)))
(setq typeList
  (cons targetType
    (dolist (x (qcdr form) (nreverse0 tmp5))
      (push
        (when (and (pairp x) (eq (qcar x) '|:|) (pairp (qcdr x))
          (pairp (qcdr (qcdr x))) (eq (qcdr (qcdr (qcdr x))) nil))
          (third x))
        tmp5))))
(list 'mdef newLhs typeList
  (dolist (x form (nreverse0 tmp8)) (push nil tmp8))
  (|postTran| rhs))))))

```

6.2.49 defun postPretend plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|pretend| '|postTran|) '|postPretend|))

```

6.2.50 defun postPretend

```

[postTran p258]
[postType p267]

```

— defun postPretend —

```

(defun |postPretend| (arg)
  (cons '|pretend| (cons (|postTran| (second arg)) (|postType| (third arg)))))

```

6.2.51 defun 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 defun postReduce plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Reduce| '|postTran|) '|postReduce|))
```

6.2.54 defun postReduce

```
[postTran p258]
[postReduce p285]
[$InteractiveMode p??]
```

— defun postReduce —

```
(defun |postReduce| (arg)
  (let (op expr g)
    (setq op (second arg))
    (setq expr (third arg))
    (if (or |$InteractiveMode| (and (pairp expr) (eq (qcar 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 defun postRepeat plist

— postvars —

```
(eval-when (eval load)
 (setf (get 'repeat '|postTran|) '|postRepeat|))
```

6.2.56 defun postRepeat

[postIteratorList p272]
[postTran p258]

— 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 defun postScripts plist

— postvars —

```
(eval-when (eval load)
 (setf (get '|Scripts| '|postTran|) '|postScripts|))
```

6.2.58 defun postScripts

[getScriptName p297]
 [postTranScripts p260]

— defun postScripts —

```
(defun |postScripts| (arg)
  (cons (|getScriptName| (second arg) (third arg) 0)
        (|postTranScripts| (third arg))))
```

—————

6.2.59 defun postSemiColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|;| ' |postTran|) ' |postSemiColon|))
```

—————

6.2.60 defun postSemiColon

[postBlock p268]
 [postFlattenLeft p287]

— defun postSemiColon —

```
(defun |postSemiColon| (u)
  (|postBlock| (cons '|Block| (|postFlattenLeft| u '|;|))))
```

—————

6.2.61 defun postFlattenLeft

[postFlattenLeft p287]

— defun postFlattenLeft —

```
(defun |postFlattenLeft| (x op)
```

```
(let (a b)
  (cond
    ((and (pairp x) (equal (qcar x) op) (pairp (qcdr x))
      (pairp (qcdr (qcdr x))) (eq (qcdr (qcdr (qcdr x))) nil))
      (setq a (qcar (qcdr x)))
      (setq b (qcar (qcdr (qcdr x))))
      (append (|postFlattenLeft| a op) (list b)))
    (t (list x)))))
```

6.2.62 defun postSignature plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Signature| '|postTran|) '|postSignature|))
```

6.2.63 defun postSignature

```
[pairp p??]
[postType p267]
[removeSuperfluousMapping p289]
[killColons p289]
```

— defun postSignature —

```
(defun |postSignature| (arg)
  (let (sig sig1 op)
    (setq op (second arg))
    (setq sig (third arg))
    (when (and (pairp sig) (eq (qcar 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 (pairp sig1) (pairp (qcar sig1)) (eq (qcar (qcar sig1)) '|Mapping|))
      (cons (cdr (qcar sig1)) (qcdr sig1))
      sig1))
```

6.2.65 defun killColons

[killColons p289]

— defun killColons —

```
(defun |killColons| (x)
  (cond
    ((atom x) x)
    ((and (pairp x) (eq (qcar x) '|Record|)) x)
    ((and (pairp x) (eq (qcar x) '|Union|)) x)
    ((and (pairp x) (eq (qcar x) '|:|) (pairp (qcdr x))
          (pairp (qcdr (qcdr x))) (eq (qcdr (qcdr (qcdr x))) nil))
     (|killColons| (third x)))
    (t (cons (|killColons| (car x)) (|killColons| (cdr x))))))
```

6.2.66 defun postSlash plist

— postvars —

```
(eval-when (eval load)
  (setf (get '/' '|postTran|) '|postSlash|))
```

6.2.67 defun postSlash

[postTran p258]

— 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 defun postTuple plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|@Tuple| '|postTran|) '|postTuple|))
```

6.2.69 defun postTuple

[postTranList p260]

— defun postTuple —

```
(defun |postTuple| (arg)
  (cond
    ((and (pairp arg) (eq (qcdr arg) nil) (eq (qcar arg) '|@Tuple|))
     arg)
    ((and (pairp arg) (eq (qcar arg) '|@Tuple|) (pairp (qcdr arg)))
     (cons '|@Tuple| (|postTranList| (cdr arg)))))
```

6.2.70 defun postTupleCollect plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|TupleCollect| '|postTran|) '|postTupleCollect|))
```

6.2.71 defun postTupleCollect

[postCollect p271]

— **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 defun postWhere plist— **postvars** —

```
(eval-when (eval load)
  (setf (get '|where| '|postTran|) '|postWhere|))
```

6.2.73 defun postWhere

[postTran p258]

[postTranList p260]

— **defun postWhere** —

```
(defun |postWhere| (arg)
  (let (b x)
    (setq b (third arg))
    (setq x (if (and (pairp b) (eq (qcar b) '|Block|)) (qcdr b) (list b)))
    (cons '|where| (cons (|postTran| (second arg)) (|postTranList| x)))))
```

6.2.74 defun postWith plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|with| '|postTran|) '|postWith|))
```

—————

6.2.75 defun postWith

```
[postTran p258]
[$insidePostCategoryIfTrue p??]
```

— defun postWith —

```
(defun |postWith| (arg)
  (let (|$insidePostCategoryIfTrue| a)
    (declare (special |$insidePostCategoryIfTrue|))
    (setq |$insidePostCategoryIfTrue| t)
    (setq a (|postTran| (second arg)))
    (cond
      ((and (pairp a) (member (qcar a) '(signature attribute if)))
       (list 'category a))
      ((and (pairp a) (eq (qcar a) 'progn))
       (cons 'category (qcdr 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 (pairp f) (eq (qcar f) '|:|))
```



```

      (pairp (setq tmp1 (qcdr f))))
    (setq f (qcar tmp1)))
  (unless (atom f) (setq f (car f)))
  (if |$topOp|
    (setq |$defOp| f)
    (setq |$topOp| f)))

```

6.3.2 defun aplTran

```

[aplTran1 p293]
[containsBang p296]
[$genno p??]
[$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 p295]
[aplTran1 p293]
[hasAplExtension p295]
[nreverse0 p??]
[ p??]
[$boot p??]

```

— defun aplTran1 —

```

(defun |aplTran1| (x)

```

```

(let (op argl1 argl f y opprime yprime tmp1 arglAssoc futureArgl g)
(declare (special $boot))
(if (atom x)
  x
  (progn
    (setq op (car x))
    (setq argl1 (cdr x))
    (setq argl (|aplTranList| argl1))
    (cond
      ((eq op '!))
      (cond
        ((and (pairp argl)
              (progn
                (setq f (qcar argl))
                (setq tmp1 (qcdr argl))
                (and (pairp tmp1)
                     (eq (qcdr tmp1) nil)
                     (progn
                      (setq y (qcar tmp1))
                      t))))))
        (cond
          ((and (pairp y)
                (progn
                  (setq opprime (qcar y))
                  (setq yprime (qcdr y))
                  t)
                (eq opprime '!)))
          (|aplTran1| (cons op (cons op (cons f yprime))))))
      ($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 (pairp tmp1)
       (progn
        (setq arglAssoc (qcar tmp1))
        (setq futureArgl (qcdr 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 tmp4 (car tmp3)) 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 p293]
 [aplTranList p295]

— defun aplTranList —

```

(defun |aplTranList| (x)
  (if (atom x)
      x
      (cons (|aplTran1| (car x)) (|aplTranList| (cdr x)))))

```

6.3.5 defun hasAplExtension

[nreverse0 p??]
 [deepestExpression p296]
 [genvar p??]
 [aplTran1 p293]
 [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 (pairp x) (eq (qcar x) '!)))))
      (setq u
        (dolist (x arg1 (nreverse0 tmp3))
          (push
            (if (and (pairp x) (eq (qcar x) '!))

```

```

      (pairp (qcdr x)) (eq (qcdr (qcdr x)) nil))
    (progn
      (setq y (qcar (qcdr 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 p296]

— defun deepestExpression —

```

(defun |deepestExpression| (x)
  (if (and (pairp x) (eq (qcar x) '!))
      (pairp (qcdr x)) (eq (qcdr (qcdr x)) nil))
      (|deepestExpression| (qcar (qcdr x)))
      x))

```

6.3.7 defun containsBang

[containsBang p296]

— defun containsBang —

```

(defun |containsBang| (u)
  (let (tmp2)
    (cond
      ((atom u) (eq u '!))
      ((and (pairp u) (equal (qcar u) 'quote)
        (pairp (qcdr u)) (eq (qcdr (qcdr u)) nil))
        nil)
      (t
        (dolist (x u tmp2)
          (setq tmp2 (or tmp2 (|containsBang| x)))))))

```

6.3.8 defun getScriptName

```
[getScriptName identp (vol5)]
[postError p262]
[internal p??]
[decodeScripts p297]
[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 p297]
```

— **defun decodeScripts** —

```
(defun |decodeScripts| (a)
  (labels (
    (fn (a)
      (let ((tmp1 0))
        (if (and (pairp a) (eq (qcar a) '|,|))
          (dolist (x (qcdr a) tmp1) (setq tmp1 (+ tmp1 (fn x))))
          1))))
    (cond
      ((and (pairp a) (eq (qcar a) '|PrefixSC|)
        (pairp (qcdr a)) (eq (qcdr (qcdr a)) nil))
        (strconc (princ-to-string 0) (|decodeScripts| (qcar (qcdr a)))))
      ((and (pairp a) (eq (qcar a) '|;|))
        (apply 'strconc (loop for x in (qcdr a) collect (|decodeScripts| x))))
      ((and (pairp a) (eq (qcar 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 p301]

— 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 (pairp x) (eq (qcar x) '|@Tuple|))  
      (qcdr 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 p346]
[action p359]
[PARSE-NewExpr processSynonyms (vol5)]
[must p358]
[current-symbol p352]
[PARSE-Statement p314]
[definition-name p309]
```

— 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 p347]
[must p358]
[PARSE-SpecialKeyWord p310]
[PARSE-SpecialCommand p311]
[push-reduction p360]
```

— 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 p351]
[action p359]
[token-symbol p??]
[current-token p353]
[PARSE-SpecialKeyWord unAbbreviateKeyword (vol5)]
[current-symbol p352]
```

— 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 p347]
 [bang p??]
 [optional p359]
 [PARSE-Expression p317]
 [push-reduction p360]
 [PARSE-SpecialCommand p311]
 [pop-stack-1 p368]
 [PARSE-CommandTail p313]
 [must p358]
 [current-symbol p352]
 [action p359]
 [PARSE-TokenList p312]
 [PARSE-TokenCommandTail p311]
 [star p359]
 [PARSE-PrimaryOrQM p313]
 [PARSE-CommandTail p313]
 [\$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 p359]

```

[star p359]
[PARSE-TokenOption p312]
[atEndOfLine p??]
[push-reduction p360]
[PARSE-TokenCommandTail p311]
[pop-stack-2 p369]
[pop-stack-1 p368]
[action p359]
[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 p347]
[must p358]
[PARSE-TokenList p312]

```

— **defun PARSE-TokenOption** —

```

(defun |PARSE-TokenOption| ()
  (and (match-advance-string "") (must (|PARSE-TokenList|))))

```

—————

8.2.12 defun PARSE-TokenList

```

[star p359]
[isTokenDelimiter p349]
[push-reduction p360]
[current-symbol p352]
[action p359]
[advance-token p354]

```

— **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 p359]
[star p359]
[push-reduction p360]
[PARSE-Option p314]
[PARSE-CommandTail p313]
[pop-stack-2 p369]
[pop-stack-1 p368]
[action p359]
[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 p347]
[push-reduction p360]
[PARSE-PrimaryOrQM p313]
[PARSE-Primary p326]
```

— 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 p347]
[must p358]
[star p359]
[PARSE-PrimaryOrQM p313]
```

— defun PARSE-Option —

```
(defun |PARSE-Option| ()
  (and (match-advance-string "")
        (must (star repeater (|PARSE-PrimaryOrQM|))))))
```

8.2.16 defun PARSE-Statement

```
[PARSE-Expr p318]
[optional p359]
[star p359]
[match-advance-string p347]
[must p358]
[push-reduction p360]
[pop-stack-2 p369]
[pop-stack-1 p368]
```

— 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 p315]
 [push-reduction p360]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— 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 p347]
 [must p358]
 [push-reduction p360]
 [pop-stack-1 p368]

— 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 p347]
 [must p358]
 [bang p??]
 [optional p359]
 [push-reduction p360]
 [PARSE-Expression p317]
 [PARSE-Category p315]
 [pop-stack-3 p369]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

```
[star p359]
[line-number p??]
[PARSE-Application p324]
[action p359]
[recordSignatureDocumentation p??]
[nth-stack p370]
[recordAttributeDocumentation p??]
[current-line p90]
```

— **defun PARSE-Category** —

```
(defun |PARSE-Category| ()
  (let (g1)
    (or (and (match-advance-string "if") (must (|PARSE-Expression|))
              (must (match-advance-string "then"))
              (must (|PARSE-Category|))
              (bang fil_test
                (optional
                  (and (match-advance-string "else")
                       (must (|PARSE-Category|))))))
        (push-reduction '|PARSE-Category|
          (list '|if| (pop-stack-3) (pop-stack-2) (pop-stack-1))))
    (and (match-advance-string "(") (must (|PARSE-Category|))
          (bang fil_test
            (optional
              (star repeator
                (and (match-advance-string ";")
                     (must (|PARSE-Category|))))))
            (must (match-advance-string ")"))
            (push-reduction '|PARSE-Category|
              (cons 'category
                (cons (pop-stack-2)
                  (append (pop-stack-1) nil))))))
    (and (action (setq g1 (line-number current-line)))
          (|PARSE-Application|)
          (must (or (and (match-advance-string ":")
                        (must (|PARSE-Expression|))
                        (push-reduction '|PARSE-Category|
                          (list '|Signature| (pop-stack-2) (pop-stack-1) ))
                        (action (|recordSignatureDocumentation|
                              (nth-stack 1) g1)))
                    (and (push-reduction '|PARSE-Category|
                          (list '|Attribute| (pop-stack-1) ))
                        (action (|recordAttributeDocumentation|
                              (nth-stack 1) g1))))))))))
```

8.2.20 defun PARSE-Expression

```
[PARSE-Expr p318]
[PARSE-rightBindingPowerOf p320]
[make-symbol-of p352]
[push-reduction p360]
[pop-stack-1 p368]
[ParseMode p309]
[prior-token p97]
```

— **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 p347]
[must p358]
[PARSE-Expr p318]
[bang p??]
[optional p359]
[star p359]
[push-reduction p360]
[pop-stack-2 p369]
[pop-stack-1 p368]
```

— **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 p318]
 [PARSE-LedPart p318]
 [optional p359]
 [star p359]
 [push-reduction p360]
 [pop-stack-1 p368]

— 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 p319]
 [push-reduction p360]
 [pop-stack-1 p368]

— 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 p319]
 [PARSE-Reduction p323]
 [PARSE-Form p323]
 [push-reduction p360]
 [pop-stack-1 p368]

```
[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 p351]
[current-symbol p352]
[PARSE-leftBindingPowerOf p319]
[lt p??]
[getl p??]
[action p359]
[PARSE-rightBindingPowerOf p320]
[PARSE-getSemanticForm p320]
[elemn p??]
[ParseMode p309]
[rbp p??]
[tmptok p308]
```

— **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 p320]
[PARSE-Infix p321]

— **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 p360]
[current-symbol p352]
[action p359]
[advance-token p354]

[optional p359]
 [PARSE-TokTail p322]
 [must p358]
 [PARSE-Expression p317]
 [push-reduction p360]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— **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 p360]
 [current-symbol p352]
 [action p359]
 [advance-token p354]
 [optional p359]
 [PARSE-TokTail p322]
 [must p358]
 [PARSE-Expression p317]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— **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 p352]
[current-char p355]
[char-eq p356]
[copy-token p??]
[action p359]
[PARSE-Qualification p322]
[$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 p347]
[must p358]
[PARSE-Primary1 p326]
[push-reduction p360]
[dollarTran p357]
[pop-stack-1 p368]
```

— **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 p323]
 [must p358]
 [PARSE-Expr p318]
 [push-reduction p360]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— **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 p352]
 [match-next-token p352]
 [action p359]
 [advance-token p354]

— **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 p347]
 [bang p??]
 [optional p359]
 [must p358]
 [push-reduction p360]
 [pop-stack-1 p368]

[PARSE-Application p324]

— **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 p326]

[optional p359]

[star p359]

[PARSE-Selector p325]

[PARSE-Application p324]

[push-reduction p360]

[pop-stack-2 p369]

[pop-stack-1 p368]

— **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 p347]
[must p358]
[PARSE-Name p333]
```

— **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 p352]
[char-ne p356]
[current-char p355]
[match-advance-string p347]
[must p358]
[PARSE-PrimaryNoFloat p326]
[push-reduction p360]
[pop-stack-2 p369]
[pop-stack-1 p368]
[PARSE-Float p327]
[PARSE-Primary p326]
[$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 p326]
 [optional p359]
 [PARSE-TokTail p322]

— defun PARSE-PrimaryNoFloat —

```

(defun |PARSE-PrimaryNoFloat| ()
  (and (|PARSE-Primary1|) (optional (|PARSE-TokTail|))))

```

8.2.40 defun PARSE-Primary

[PARSE-Float p327]
 [PARSE-PrimaryNoFloat p326]

— defun PARSE-Primary —

```

(defun |PARSE-Primary| ()
  (or (|PARSE-Float|) (|PARSE-PrimaryNoFloat|)))

```

8.2.41 defun PARSE-Primary1

[PARSE-VarForm p332]
 [optional p359]
 [current-symbol p352]
 [PARSE-Primary1 p326]
 [must p358]
 [pop-stack-2 p369]
 [pop-stack-1 p368]
 [push-reduction p360]
 [PARSE-Quad p331]

[PARSE-String p331]
 [PARSE-IntegerTok p330]
 [PARSE-FormalParameter p331]
 [match-string p346]
 [PARSE-Data p334]
 [match-advance-string p347]
 [PARSE-Expr p318]
 [PARSE-Sequence p337]
 [PARSE-Enclosure p330]
 [\$boot p??]

— **defun PARSE-Primary1** —

```
(defun |PARSE-Primary1| ()
  (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 p328]
 [must p358]
 [PARSE-FloatExponent p329]
 [push-reduction p360]
 [make-float p??]
 [pop-stack-4 p369]
 [pop-stack-3 p369]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— **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 p352]
[char-eq p356]
[current-char p355]
[char-ne p356]
[next-char p355]
[PARSE-IntegerTok p330]
[must p358]
[PARSE-FloatBasePart p328]
[PARSE-IntegerTok p330]
[push-reduction p360]
[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 p347]
[must p358]
```

— defun PARSE-FloatBasePart —

```
[current-symbol p352]
[current-char p355]
[action p359]
[advance-token p354]
[PARSE-IntegerTok p330]
[match-advance-string p347]
[must p358]
[push-reduction p360]
[PARSE-FloatExponent identp (vol5)]
[floatexpid p357]
```

— 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 p347]
[must p358]
[PARSE-Expr p318]
[push-reduction p360]
[pop-stack-1 p368]

```

— 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 p366]

```

— defun PARSE-IntegerTok —

```

(defun |PARSE-IntegerTok| () (parse-number))

```

8.2.48 defun PARSE-FormalParameter

[PARSE-FormalParameterTok p331]

— defun PARSE-FormalParameter —

```
(defun |PARSE-FormalParameter| () (|PARSE-FormalParameterTok|))
```

8.2.49 defun PARSE-FormalParameterTok

[parse-argument-designator p367]

— defun PARSE-FormalParameterTok —

```
(defun |PARSE-FormalParameterTok| () (parse-argument-designator))
```

8.2.50 defun PARSE-Quad

[match-advance-string p347]

[push-reduction p360]

[PARSE-GlyphTok p336]

[\$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 p364]

— **defun PARSE-String** —

```
(defun |PARSE-String| () (parse-spadstring))
```

8.2.52 **defun PARSE-VarForm**

```
[PARSE-Name p333]
[optional p359]
[PARSE-Scripts p332]
[push-reduction p360]
[pop-stack-2 p369]
[pop-stack-1 p368]
```

— **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 p347]
[must p358]
[PARSE-ScriptItem p333]
```

— **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 p318]
 [optional p359]
 [star p359]
 [match-advance-string p347]
 [must p358]
 [PARSE-ScriptItem p333]
 [push-reduction p360]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— **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 p365]
 [push-reduction p360]
 [pop-stack-1 p368]

— **defun PARSE-Name** —

```
(defun |PARSE-Name| ()
  (and (parse-identifier) (push-reduction '|PARSE-Name| (pop-stack-1))))
```

—————

8.2.56 defun PARSE-Data

[action p359]
 [PARSE-Sexpr p334]
 [push-reduction p360]
 [translabel p361]
 [pop-stack-1 p368]
 [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 p334]

— **defun PARSE-Sexpr** —

```
(defun |PARSE-Sexpr| ()
  (and (action (advance-token)) (|PARSE-Sexpr1|)))
```

—————

8.2.58 defun PARSE-Sexpr1

[PARSE-AnyId p336]
 [optional p359]
 [PARSE-NBGlyphTok p335]
 [must p358]
 [PARSE-Sexpr1 p334]
 [action p359]
 [pop-stack-2 p369]
 [nth-stack p370]
 [match-advance-string p347]
 [push-reduction p360]
 [PARSE-IntegerTok p330]
 [pop-stack-1 p368]

[PARSE-String p331]
 [bang p??]
 [star p359]
 [PARSE-GlyphTok p336]

— 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 p351]
 [action p359]
 [advance-token p354]
 [tok p308]

— 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 p351]
[action p359]
[advance-token p354]
[tok p308]
```

— 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 p365]
[match-string p346]
[push-reduction p360]
[current-symbol p352]
[action p359]
[advance-token p354]
[parse-keyword p366]
```

— defun PARSE-AnyId —

```
(defun |PARSE-AnyId| ()
  (or (parse-identifier)
      (or (and (match-string "$")
                (push-reduction '|PARSE-AnyId| (current-symbol))
                (action (advance-token)))
          (parse-keyword)))))
```

8.2.62 defun PARSE-Sequence

[PARSE-OpenBracket p338]
 [must p358]
 [PARSE-Sequence1 p337]
 [match-advance-string p347]
 [PARSE-OpenBrace p338]
 [push-reduction p360]
 [pop-stack-1 p368]

— **defun PARSE-Sequence** —

```
(defun |PARSE-Sequence| ()
  (or (and (|PARSE-OpenBracket|) (must (|PARSE-Sequence1|))
    (must (match-advance-string "]")))
    (and (|PARSE-OpenBrace|) (must (|PARSE-Sequence1|))
    (must (match-advance-string "}")))
    (push-reduction '|PARSE-Sequence|
      (list '|brace| (pop-stack-1))))))
```

8.2.63 defun PARSE-Sequence1

[PARSE-Expression p317]
 [push-reduction p360]
 [pop-stack-2 p369]
 [pop-stack-1 p368]
 [optional p359]
 [PARSE-IteratorTail p339]

— **defun PARSE-Sequence1** —

```
(defun |PARSE-Sequence1| ()
  (and (or (and (|PARSE-Expression|)
    (push-reduction '|PARSE-Sequence1|
      (list (pop-stack-2) (pop-stack-1))))
    (push-reduction '|PARSE-Sequence1| (list (pop-stack-1))))
    (optional
      (and (|PARSE-IteratorTail|)
        (push-reduction '|PARSE-Sequence1|
          (cons 'collect
            (append (pop-stack-1)
              (list (pop-stack-1))))))))))
```

8.2.64 defun PARSE-OpenBracket

```
[getToken p350]
[current-symbol p352]
[eqcar p??]
[push-reduction p360]
[action p359]
[advance-token p354]
```

— 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 p350]
[current-symbol p352]
[eqcar p??]
[push-reduction p360]
[action p359]
[advance-token p354]
```

— 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 p347]
 [bang p??]
 [optional p359]
 [star p359]
 [PARSE-Iterator p339]

— **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 p347]
 [must p358]
 [PARSE-Primary p326]
 [PARSE-Expression p317]
 [PARSE-Expr p318]
 [pop-stack-3 p369]
 [pop-stack-2 p369]
 [pop-stack-1 p368]
 [optional p359]

— **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 p360]
[current-symbol p352]
[action p359]
[advance-token p354]
[optional p359]
[PARSE-TokTail p322]
[pop-stack-1 p368]

```

— 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 p347]
[must p358]
[PARSE-Expr p318]
[push-reduction p360]
[pop-stack-2 p369]
[pop-stack-1 p368]
```

— **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 p347]
[must p358]
[PARSE-Expression p317]
[push-reduction p360]
[pop-stack-1 p368]
```

— **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 p347]
[must p358]
[PARSE-Expression p317]
[push-reduction p360]
[pop-stack-1 p368]
```

— 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 p347]
 [PARSE-Expression p317]
 [must p358]
 [push-reduction p360]
 [PARSE-Label p325]
 [pop-stack-1 p368]

— 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 p336]
 [bang p??]
 [optional p359]
 [PARSE-Expression p317]
 [push-reduction p360]
 [pop-stack-2 p369]

[pop-stack-1 p368]

— **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 p347]
 [must p358]
 [PARSE-Expression p317]
 [bang p??]
 [optional p359]
 [PARSE-ElseClause p343]
 [push-reduction p360]
 [pop-stack-3 p369]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— **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 p352]
 [PARSE-Conditional p343]
 [PARSE-Expression p317]

— defun PARSE-ElseClause —

```
(defun |PARSE-ElseClause| ()
  (or (and (eq (current-symbol) '|if|) (|PARSE-Conditional|))
      (|PARSE-Expression|)))
```

8.2.77 defun PARSE-Loop

[star p359]
 [PARSE-Iterator p339]
 [must p358]
 [match-advance-string p347]
 [PARSE-Expr p318]
 [push-reduction p360]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— 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 p325]
 [must p358]
 [PARSE-Expr p318]
 [push-reduction p360]
 [pop-stack-2 p369]
 [pop-stack-1 p368]

— 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 p366]
[push-reduction p360]
[pop-stack-1 p368]
[bfp- p??]
[$boot p??]
```

— defun PARSE-FloatTok —

```
(defun |PARSE-FloatTok| ()
  (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 p350]
 [skip-blanks p346]
 [line-past-end-p p91]
 [current-char p355]
 [initial-substring-p p348]
 [subseq p??]
 [\$line p89]
 [line p89]

— 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 p355]
 [token-lookahead-type p347]
 [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 p348]

[current-token p353]
 [match-string p346]
 [line-current-index p??]
 [line-past-end-p p91]
 [line-current-char p??]
 [\$token p96]
 [\$line p89]

— **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 p350]
 [token-nonblank p??]
 [pack p??]
 [escape-keywords p349]
 [\$boot p??]
 [\$spad p395]

— 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

— defun escape-keywords —

```

(defun escape-keywords (pname id)
  (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 p352]

— 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 p348]
[line-current-segment p92]
[strconc p??]
[line-number p??]
[token-nonblank p??]
[line-new-line p92]
[line-number p??]
```

[valid-tokens p98]

— **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 p353]

[match-token p351]

— **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 p354]
[match-token p351]

— **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 p352]
[current-token p353]

— **defun current-symbol** —

```
(defun current-symbol ()
  (make-symbol-of (current-token)))
```

—————

8.3.18 **defun make-symbol-of**

[\$token p96]

— **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 p353]
 [valid-tokens p98]
 [current-token p353]

— 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 p355]
 [valid-tokens p98]

— 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 p353]

```
[current-token p353]
[valid-tokens p98]
[next-token p354]
```

— 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 p353]

```
[copy-token p??]
[try-get-token p353]
[valid-tokens p98]
[current-token p353]
```

— 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 p355]
 [XTokenReader p355]

— **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 p89]
 [current-line p90]

— **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 p90]

```
[line-next-char p91]
[current-line p90]
```

— **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 p356]
 [meta-error-handler p356]

— **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**

[floatexpid identp (vol5)]
 [floatexpid pname (vol5)]
 [spadreduce p??]
 [collect p271]
 [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 p357]

— 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 p360]
 [pop-stack-1 p368]

— 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 p94](#)]

— `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 p96](#)]
 [[make-reduction p??](#)]
 [[reduce-stack p360](#)]

— `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 translablel

[translablel p361]

— defun translablel —

```
(defun translablel (x al)
  (translablel x al) x)
```

—————

9.0.48 defun translablel1

[refvecp p??]
[maxindex p??]
[translablel p361]
[lassoc p??]

— defun translablel1 —

```
(defun translablel1 (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))
              (translablel1 (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??]
```

— defun displayPreCompilationErrors —

```
(defun |displayPreCompilationErrors| ()
  (let (n errors heading)
    (declare (special |$postStack| |$topOp|))
    (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)
  (unless |$InteractiveMode|
    (let ((index (case kind
                    (|syntax| 0)
                    (|precompilation| 1)
                    (|semantic| 2)
                    (t (error "BUMPERRORCOUNT")))))
      (setelt $spad_errors index (1+ (elt $spad_errors index))))))
```

9.0.51 defun parseTranCheckForRecord

```
;parseTranCheckForRecord(x,op) ==
; (x:= parseTran x) is ['Record,:l] =>
;   or/[y for y in l | y isnt [":",.,.]] =>
;   postError [" Constructor",:bright x,"has missing label"]
;   x
; x
```

```
[qcar p??]
[qcdr p??]
[postError p262]
[parseTran p101]
```

— defun parseTranCheckForRecord —

```
(defun |parseTranCheckForRecord| (x op)
  (declare (ignore op))
  (let (tmp3)
    (setq x (|parseTran| x))
    (cond
      ((and (pairp x) (eq (qcar x) '|Record|))
        (cond
          ((do ((z nil tmp3) (tmp4 (qcdr x) (cdr tmp4)) (y nil))
                ((or z (atom tmp4)) tmp3)
                (setq y (car tmp4))
                (cond
                  ((null (and (pairp y) (eq (qcar y) '|:|) (pairp (qcdr y))
                               (pairp (qcdr (qcdr y))) (eq (qcdr (qcdr 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 p257]

```

— 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 p351]
[token-symbol p??]
[push-reduction p360]
[advance-token p354]

```


— 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 p351]
 [token-symbol p??]
 [push-reduction p360]
 [advance-token p354]

— 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 p351]
 [token-symbol p??]
 [push-reduction p360]
 [advance-token p354]

— 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 p351]
[token-symbol p??]
[push-reduction p360]
[advance-token p354]

```

— 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 p351]
[token-symbol p??]
[push-reduction p360]
[advance-token p354]

```

— 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 p360]
 [match-current-token p351]
 [token-symbol p??]
 [advance-token p354]

— 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

— defun print-package —

```
(defun print-package (package)
  (format out-stream "~&~%(IN-PACKAGE ~S )~%~%" package))
```

9.0.61 defun checkWarning

[postError p262]
 [concat p??]

— defun checkWarning —

```
(defun |checkWarning| (msg)
  (|postError| (|concat| "Parsing error: " msg)))
```

9.0.62 defun tuple2List

```
[tuple2List p368]
[postTranSegment p276]
[postTran p258]
[$boot p??]
[$InteractiveMode p??]
```

— **defun tuple2List** —

```
(defun |tuple2List| (arg)
  (let (u p q)
    (declare (special |$InteractiveMode| $boot))
    (when (pairp arg)
      (setq u (|tuple2List| (qcdr arg)))
      (cond
        ((and (pairp (qcar arg)) (eq (qcar (qcar arg)) 'segment)
              (pairp (qcdr (qcar arg)))
              (pairp (qcdr (qcdr (qcar arg)))))
         (eq (qcdr (qcdr (qcdr (qcar arg)))) nil))
        (setq p (qcar (qcdr (qcar arg))))
        (setq q (qcar (qcdr (qcdr (qcar arg)))))
        (cond
          ((null u) (list '|construct| (|postTranSegment| p q)))
          ((and |$InteractiveMode| (null $boot))
           (cons '|append|
                 (cons (list '|construct| (|postTranSegment| p q))
                       (list (|tuple2List| (qcdr arg))))))
          (t
           (cons '|nconc|
                 (cons (list '|construct| (|postTranSegment| p q))
                       (list (|tuple2List| (qcdr arg)))))))
        ((null u) (list '|construct| (|postTran| (qcar arg))))
        (t (list '|cons| (|postTran| (qcar arg)) (|tuple2List| (qcdr arg)))))))
```

—————

9.0.63 defmacro pop-stack-1

```
[reduction-value p??]
[Pop-Reduction p370]
```

— **defmacro pop-stack-1** —

```
(defmacro pop-stack-1 () '(reduction-value (Pop-Reduction)))
```

—————

9.0.64 defmacro pop-stack-2

[stack-push p96]
 [reduction-value p??]
 [Pop-Reduction p370]

— 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 p96]
 [reduction-value p??]
 [Pop-Reduction p370]

— 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 p96]
 [reduction-value p??]
 [Pop-Reduction p370]

— 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 p96]
```

— 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 p371]

[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]
[$comblocklist p371]
[$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 p371]

— **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 p93]
 [preparseReadLine p85]
 [preparseReadLine1 p87]
 [skip-to-endif p374]

— **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 p380]
[compileSpadLispCmd p431]
[$newConlist p??]
[$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??]
[convertSpadToAsFile p??]
[compilerDoitWithScreenedLisplib p??]
[compilerDoit p384]
[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 p385]
[$byConstructors p434]
[$constructorsSeen p434]
```

— 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 p386]
[/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 p431]
[/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.nrlib/EQ.lsp")
<3 (OPEN #<input stream "/tmp/EQ.nrlib/EQ.lsp">)
3> (OPEN #p"/tmp/EQ.nrlib/EQ.data" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrlib/EQ.data">)
3> (OPEN #p"/tmp/EQ.nrlib/EQ.c" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrlib/EQ.c">)
3> (OPEN #p"/tmp/EQ.nrlib/EQ.h" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrlib/EQ.h">)
3> (OPEN #p"/tmp/EQ.nrlib/EQ.fn" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrlib/EQ.fn">)
3> (OPEN #p"/tmp/EQ.nrlib/EQ.o" :DIRECTION :OUTPUT :IF-EXISTS :APPEND)
<3 (OPEN #<output stream "/tmp/EQ.nrlib/EQ.o">)
3> (OPEN #p"/tmp/EQ.nrlib/EQ.data")
<3 (OPEN #<input stream "/tmp/EQ.nrlib/EQ.data">)
3> (OPEN "/tmp/EQ.nrlib/index.kaf")
<3 (OPEN #<input stream "/tmp/EQ.nrlib/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| (-> $ $))
          (|Signature| |rhs| (-> $ $))
          (|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)
          NIL)
    )
  )
)

```


[illegible]

```

(|;|
  (|;|
    (|;|
      (|;|
        (|:=| |Rep|
          (|Record| (|,| (|:=| |lhs| S) (|:=| |rhs| S))))
          (|,| |eq1| (|:=| |eq2| $)))
          (|:=| |s| S))
        (|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))))))
                  (|construct| |eq|)))
                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|)
(|;|
(|;|
(|;|
(|;|
(|;|
(|;|
  (== (- |eq|) (= (- (|lhs| |eq|)) (- (|rhs| |eq|))))
  (== (- |s| |eq2|) (- (|construct| (|,| |s| |s|)) |eq2|)))
  (== (- |eq1| |s|) (- |eq1| (|construct| (|,| |s| |s|)))))
  (== (|leftZero| |eq|) (= 0 (- (|rhs| |eq|) (|lhs| |eq|))))
  (== (|rightZero| |eq|) (= (- (|lhs| |eq|) (|rhs| |eq|)) 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| |eq|)
    (|;|
    (|;|

```

```

      (=> (|case| (|:=| |lh| (|recip| (|lhs| |eq|))) "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 p309]
[pop-stack-1 p368]
[s-process p396]
[ioclear p??]
[spad shut (vol5)]
[$noSubsumption p??]
[$InteractiveFrame p??]
[$InitialDomainsInScope p??]
[$InteractiveMode p??]
[line p89]
[echo-meta p??]
[/editfile p??]
[*comp370-apply* p??]
[*eof* p??]
[file-closed 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*
    file-closed |$noSubsumption| |$InteractiveFrame|
    |$InteractiveMode| |$InitialDomainsInScope|))
  ;; only rebind |$InteractiveFrame| if compiling
  (progv (if (not |$InteractiveMode|) '(|$InteractiveFrame|))
    (if (not |$InteractiveMode|)
      (list (|addBinding| '|$DomainsInScope|
        '((fluid . |true|)
          (special . ,(copy-tree |$InitialDomainsInScope|)))
        (|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 p399]
[new2OldLisp p364]
[parseTransform p101]
[postTransform p257]
[displayPreCompilationErrors p362]
[prettyprint p??]
[s-process processInteractive (vol5)]
[compTopLevel p400]
[def-process p??]
[displaySemanticErrors p??]
[terpri p??]
[get-internal-run-time p??]
[$Index p??]
[$macroassoc p??]

```



```

[$newspad p??]
[$PolyMode p??]
[$EmptyMode p??]
[$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??]
[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 $newspad |$PolyModel| |$EmptyModel|
|$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|))
(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 $boot)
  (if |$InteractiveModel|
    (|processInteractive| x nil)
    (when (setq u (|compTopLevel| x |$EmptyModel| |$InteractiveFrame|))
      (setq |$InteractiveFrame| (third u))))
  (def-process x))
(when |$semanticErrorStack| (|displaySemanticErrors|))
(terpri)))

```

10.1.8 defun print-defun

```
[is-console p373]
[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 p399]
```

— defun def-rename —

```
(defun def-rename (x)
  (def-rename1 x))
```

—————

10.1.10 defun def-rename1

```
[def-rename1 p399]
```

— 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 p401]
[$NRTderivedTargetIfTrue p??]
[$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 (pairp form) (eq (qcar form) 'def))
           (and (pairp form) (eq (qcar form) '|where|))
           (progn
             (setq t1 (qcdr form))
             (and (pairp t1)
                  (progn
                     (setq t2 (qcar t1))
                     (and (pairp t2) (eq (qcar 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 p402]

— 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 p403]
[compOrCroak1,compactify p431]
[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 p401]
```

— 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 '|compOrCroak1| (|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 p404]
[$compStack p??]
[$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 p405]
[compNoStacking1 p404]
[$compStack p??]
[$Representation p??]
[$EmptyMode p??]
```

— 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 p405]
[$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 p405]
[isDomainForm p248]
[isFunctor p188]
[insert p??]
[opOf p??]
[nequal p??]
[addDomain p186]
[$bootStrapMode p??]
[$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

```
;comp3(x,m,$e) ==
; --returns a Triple or %else nil to signalcan't do'
; $e:= addDomain(m,$e)
; e:= $e --for debugging purposes
; m is ["Mapping",:] => compWithMappingMode(x,m,e)
; m is ["QUOTE",a] => (x=a => [x,m,$e]; nil)
; STRINGP m => (atom x => (m=x or m=STRINGIMAGE x => [m,m,e]; nil); nil)
; ^x or atom x => compAtom(x,m,e)
; op:= first x
; getmode(op,e) is ["Mapping",:ml] and (u:= applyMapping(x,m,e,ml)) => u
; op is ["KAPPA",sig,varlist,body] => compApply(sig,varlist,body,rest x,m,e)
; op=":" => compColon(x,m,e)
; op="::" => compCoerce(x,m,e)
; not ($insideCompTypeOf=true) and stringPrefix?(' "TypeOf",PNAME op) =>
```

```

;   compTypeOf(x,m,e)
;   t:= compExpression(x,m,e)
;   t is [x',m',e'] and not MEMBER(m',getDomainsInScope e') =>
;   [x',m',addDomain(m',e')]
;   t

```

```

[addDomain p186]
[compWithMappingMode p419]
[compAtom p409]
[getmode p??]
[applyMapping p??]
[compApply p??]
[compColon p215]
[compCoerce p213]
[stringPrefix? p??]
[comp3 pname (vol5)]
[compTypeOf p408]
[compExpression p413]
[comp3 member (vol5)]
[getDomainsInScope p188]
[$e p??]
[$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 (pairp mode) (eq (qcar mode) '|Mapping|))
        (|compWithMappingMode| form mode env))
      ((and (pairp mode) (eq (qcar mode) '|quote|)
        (progn
          (setq tmp1 (qcdr mode))
          (and (pairp tmp1) (eq (qcdr tmp1) nil)
            (progn (setq a (qcar 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 (pairp tmp1)
            (eq (qcar tmp1) '|Mapping|)
            (progn (setq ml (qcdr tmp1)) t)))
      (setq u (|applyMapping| form mode env ml)))
  u)
((and (pairp op) (eq (qcar op) 'kappa)
  (progn
    (setq tmp1 (qcdr op))
    (and (pairp tmp1)
          (progn
            (setq sig (qcar tmp1))
            (setq tmp2 (qcdr tmp1))
            (and (pairp tmp2)
                  (progn
                    (setq varlist (qcar tmp2))
                    (setq tmp3 (qcdr tmp2))
                    (and (pairp tmp3)
                          (eq (qcdr tmp3) nil)
                          (progn
                            (setq body (qcar 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 (pairp tt)
          (progn
            (setq xprime (qcar tt))
            (setq tmp1 (qcdr tt))
            (and (pairp tmp1)
                  (progn
                    (setq mprime (qcar tmp1))
                    (setq tmp2 (qcdr tmp1))
                    (and (pairp tmp2)
                          (eq (qcdr tmp2) nil)
                          (progn
                            (setq eprime (qcar 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 p405]
[$insideCompTypeOf p??]
[$FormalMapVariableList p202]
```

— defun compTypeOf —

```
(defun |compTypeOf| (form mode env)
  (let (|$insideCompTypeOf| op argl newModemap)
    (declare (special |$insideCompTypeOf| |$FormalMapVariableList|))
    (setq op (car form))
    (setq argl (cdr form))
    (setq |$insideCompTypeOf| t)
    (setq newModemap
      (eqsubstlist argl |$FormalMapVariableList| (|get| op '|modemap| env)))
    (setq env (|put| op '|modemap| newModemap env))
    (|comp3| form mode env)))
```

10.1.20 defun compColonInside

```
[addDomain p186]
[comp p403]
[coerce p??]
[stackWarning p??]
[opOf p??]
[stackSemanticError p??]
[$newCompilerUnionFlag p??]
[$EmptyMode p??]
```

— 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

```

; compAtom(x,m,e) ==
;   T:= compAtomWithModemap(x,m,e,get(x,"modemap",e)) => T
;   x="nil" =>
;     T:=
;       modeIsAggregateOf('List,m,e) is [.,R]=> compList(x,['List,R],e)
;       modeIsAggregateOf('Vector,m,e) is [.,R]=> compVector(x,['Vector,R],e)
;       T => convert(T,m)
;   t:=
;     isSymbol x =>
;       compSymbol(x,m,e) or return nil
;     m = $Expression and primitiveType x => [x,m,e]
;     STRINGP x => [x,x,e]
;     [x,primitiveType x or return nil,e]
;   convert(t,m)

```

```

[compAtomWithModemap p??]
[get p??]
[modeIsAggregateOf p??]
[compList p413]
[compVector p254]
[convert p411]
[isSymbol p??]
[compSymbol p411]
[primitiveType p411]
[primitiveType p411]
[$Expression p??]

```

— defun compAtom —

```

(defun |compAtom| (form mode env)
  (prog (tmp1 tmp2 r td tt)
    (declare (special |$Expression|))
    (return
     (cond
      ((setq td
        (|compAtomWithModemap| form mode env (|get| form '|modemap| env))) td)
      ((eq form '|nil|)
        (setq td
          (cond
           ((progn
              (setq tmp1 (|modeIsAggregateOf| '|List| mode env))
              (and (pairp tmp1)
                (progn
                  (setq tmp2 (qcdr tmp1))
                  (and (pairp tmp2)
                    (eq (qcdr tmp2) nil)
                    (progn
                     (setq r (qcar tmp2)) t))))))
            (|compList| form (list '|List| r) env))
           ((progn
              (setq tmp1 (|modeIsAggregateOf| '|Vector| mode env))
              (and (pairp tmp1)
                (progn
                  (setq tmp2 (qcdr tmp1))
                  (and (pairp tmp2) (eq (qcdr tmp2) nil)
                    (progn
                     (setq r (qcar tmp2)) t))))))
            (|compVector| form (list '|Vector| r) env))))
        (when td (|convert| td mode)))
      (t
        (setq tt
          (cond
           ((|isSymbol| form) (or (|compSymbol| form mode env) (return nil)))
           ((and (equal mode |$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 p??]
[coerce p??]

— 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 p250]
[\$EmptyMode p??]

— defun primitiveType —

```
(defun |primitiveType| (form)
  (declare (special |$DoubleFloat| |$NegativeInteger| |$PositiveInteger|
                    |$NonNegativeInteger| |$String| |$EmptyMode|))
  (cond
    ((null form) |$EmptyMode|)
    ((stringp form) |$String|)
    ((integerp form)
     (cond
      ((= 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??]
[$FormalMapVariableList p202]
[$compForModeIfTrue p??]
[$formalArgList p??]
[$NoValueMode p??]
[$functorLocalParameters p??]
[$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

```

;compList(l,m is ["List",mUnder],e) ==
;  null l => [NIL,m,e]
;  Tl:= [[.,mUnder,e]:= comp(x,mUnder,e) or return "failed" for x in l]
;  Tl="failed" => nil
;  T:= [{"LIST",.:T.expr for T in Tl}],["List",mUnder],e]

```

[comp p403]

— 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 p414]

`[$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 (get1 (car form) 'special)))
        (funcall fn form mode env)
        (|compForm| form mode env))))
```

—————

10.1.27 defun compForm

`[compForm1 p414]`
`[compArgumentsAndTryAgain p418]`
`[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 p??]`
`[compOrCroak p401]`
`[compExpressionList p??]`
`[coerceable p??]`
`[comp p403]`
`[coerce p??]`
`[compForm2 p416]`
`[augModemapsFromDomain1 p191]`
`[getFormModemaps p??]`
`[nreverse0 p??]`
`[addDomain p186]`

```
[compToApply p??]
[$NumberOfArgsIfInteger p??]
[$Expression p??]
[$EmptyMode p??]
```

— **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 |$NumberOfArgsIfInteger| (|#| arg1))
    (cond
      ((eq op '|error|)
       (list
        (cons op
              (dolist (x arg1 (nreverse0 tmp4))
                (setq tmp2 (|outputComp| x env))
                (setq env (third tmp2))
                (push (car tmp2) tmp4)))
              mode env))
        ((and (pairp op) (eq (qcar op) '|elt|)
         (progn
          (setq tmp3 (qcdr op))
          (and (pairp tmp3)
               (progn
                (setq domain (qcar tmp3))
                (setq tmp1 (qcdr tmp3))
                (and (pairp tmp1)
                     (eq (qcdr tmp1) nil)
                     (progn
                      (setq opprime (qcar tmp1))
                      t))))))
          (cond
            ((eq domain '|Lisp|)
             (list
              (cons opprime
                    (dolist (x arg1 (nreverse tmp7))
                      (setq tmp2 (|compOrCroak| x |$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 (pairp domain) (eq (qcar domain) '|Mapping|)
  (setq ans
    (|compForm2| (cons opprime argl) mode
      (setq env (|augModemapsFromDomain1| domain domain env))
      (dolist (x (|getFormModemaps| (cons opprime argl) env)
        (nreverse0 tmp6))
        (when
          (and (pairp x)
            (and (pairp (qcar x)) (equal (qcar (qcar x)) domain)))
            (push x tmp6))))))
  ans)
(setq ans
  (|compForm2| (cons opprime argl) mode
    (setq env (|addDomain| domain env))
    (dolist (x (|getFormModemaps| (cons opprime argl) env)
      (nreverse0 tmp5))
      (when
        (and (pairp x)
          (and (pairp (qcar x)) (equal (qcar (qcar x)) domain)))
          (push x tmp5))))))
  ans)
((and (eq opprime '|construct|) (|coerceable| domain mode env))
  (when (setq td (|comp| (cons opprime argl) 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 argl mode env))))))

```

10.1.29 defun compForm2

```

[take p??]
[length p??]
[nreverse0 p??]
[sublis p??]
[assoc p??]
[PredImplies p??]
[isSimple p??]
[compUniquely p??]
[compFormPartiallyBottomUp p??]

```

```
[compForm3 p??]
[$EmptyMode p??]
[$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 (pairp u)
              (progn
                (setq tmp6 (qcar u))
                (and (pairp tmp6) (progn (setq dc (qcar tmp6)) t))))
          (progn
            (setq tmp7 (qcdr u))
            (and (pairp tmp7) (eq (qcdr tmp7) nil))
            (progn
              (setq tmp1 (qcar tmp7))
              (and (pairp tmp1)
                (progn
                  (setq cond (qcar tmp1))
                  (setq tmp2 (qcdr tmp1))
                  (and (pairp tmp2) (eq (qcdr tmp2) nil))
                  (progn
                    (setq tmp3 (qcar tmp2))
                    (and (pairp tmp3) (eq (qcar tmp3) '|Subsumed|')
                      (progn
                        (setq tmp4 (qcdr tmp3))
                        (and (pairp tmp4)
                          (progn
                            (setq tmp5 (qcdr tmp4))
                            (and (pairp tmp5)
                              (eq (qcdr tmp5) nil)
                                (progn
                                  (setq nsig (qcar tmp5))
                                  t))))))))))))))))
              (setq v (|assoc| (cons dc nsig) modemapList))
              (pairp v)
              (progn
```

```

      (setq tmp6 (qcdr v))
      (and (pairp tmp6) (eq (qcdr tmp6) nil)
        (progn
          (setq tmp7 (qcar tmp6))
          (and (pairp tmp7)
            (progn
              (setq ncond (qcar 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 argl
    while (and (|isSimple| x)
      (setq td (|compUniquely| x |$EmptyMode| env)))
    collect td
    do (setq env (third td))))
(cond
  ((some #'identity tl)
    (setq partialModelList (loop for x in tl collect (when x (second x))))
    (or
      (|compFormPartiallyBottomUp| form mode env modemapList partialModelList)
      (|compForm3| form mode env modemapList)))
  (t (|compForm3| form mode env modemapList))))

```

10.1.30 defun compArgumentsAndTryAgain

```

[comp p403]
[compForm1 p414]
[$EmptyMode p??]

```

— defun compArgumentsAndTryAgain —

```

(defun |compArgumentsAndTryAgain| (form mode env)
  (let (argl tmp1 a tmp2 tmp3 u)
    (declare (special |$EmptyMode|))
    (setq argl (cdr form))
    (cond
      ((and (pairp form) (eq (qcar form) '|elt|)
        (progn

```

```

      (setq tmp1 (qcdr form))
      (and (pairp tmp1)
        (progn
          (setq a (qcar tmp1))
          (setq tmp2 (qcdr tmp1))
          (and (pairp tmp2) (eq (qcdr 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))))))

```

10.1.31 defun compWithMappingMode

[compWithMappingMode1 p419]
 [\$formalArgList p??]

— defun compWithMappingMode —

```

(defun |compWithMappingMode| (form mode oldE)
  (declare (special |$formalArgList|))
  (|compWithMappingMode1| form mode oldE |$formalArgList|))

```

10.1.32 defun compWithMappingMode1

```

;compWithMappingMode1(x,m is ["Mapping",m',:sl],oldE,$formalArgList) ==
; $killOptimizeIfTrue: local:= true
; e:= oldE
; isFunctor x =>
;   if get(x,"modemap",$CategoryFrame) is [[[,target,:argModeList],.],:.] and
;   (and/[extendsCategoryForm("$",s,mode) for mode in argModeList for s in sl]
;   ) and extendsCategoryForm("$",target,m') then return [x,m,e]
; if STRINGP x then x:= INTERN x
; res:=nil
; old_style:=true

```

```

; if x is ["+>",vl,nx] then
;   old_style:=false
;   vl is [":",:] =>
;     ress:=compLambda(x,m,oldE)
;     ress
;   vl:=
;     vl is ["Tuple",:vl1] => vl1
;     vl
;   vl:=
;     SYMBOLP(vl) => [vl]
;     LISTP(vl) and (and/[SYMBOLP(v) for v in vl]) => vl
;     stackAndThrow ["bad +-> arguments:",vl]
;     $formatArgList:=[:vl,:$formalArgList]
;   x:=nx
; else
;   vl:=take(#sl,$FormalMapVariableList)
;   ress => ress
;   for m in sl for v in vl repeat
;     [.,.,e]:= compMakeDeclaration([":",v,m],$EmptyMode,e)
;   old_style and not null vl and not hasFormalMapVariable(x, vl) => return
;   [u.,.] := comp([x,:vl],m',e) or return nil
;   extractCodeAndConstructTriple(u, m, oldE)
;   null vl and (t := comp([x], m', e)) => return
;   [u.,.] := t
;   extractCodeAndConstructTriple(u, m, oldE)
;   [u.,.] := comp(x,m',e) or return nil
;   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.
;   $FUNNAME :local := nil
;   $FUNNAME__TAIL :local := [nil]
;   expandedFunction:=COMP_-TRAN CADR uu
;   frees:=freelist(expandedFunction,vl,nil,e)
;   where freelist(u,bound,free,e) ==
;     atom u =>
;       not IDENTP u => free
;       MEMQ(u,bound) => free
;       v:=ASSQ(u,free) =>
;         RPLACD(v,1+CDR v)
;         free
;       not getmode(u, e) => free
;       [[u,:1],:free]
;     op:=CAR u
;     MEMQ(op, '(QUOTE GO function)) => free
;     EQ(op,'LAMBDA) =>
;       bound:=UNIONQ(bound,CADR u)
;       for v in CDDR u repeat
;         free:=freelist(v,bound,free,e)

```



```

;      free
;      EQ(op,'PROG) =>
;        bound:=UNIONQ(bound,CADR u)
;        for v in CDDR u | NOT ATOM v repeat
;          free:=freelist(v,bound,free,e)
;      free
;      EQ(op,'SEQ) =>
;        for v in CDR u | NOT ATOM v repeat
;          free:=freelist(v,bound,free,e)
;      free
;      EQ(op,'COND) =>
;        for v in CDR u repeat
;          for vv in v repeat
;            free:=freelist(vv,bound,free,e)
;      free
;      if ATOM op then u:=CDR u --Atomic functions aren't descended
;      for v in u repeat
;        free:=freelist(v,bound,free,e)
;      free
;      expandedFunction :=
;        --One free can go by itself, more than one needs a vector
;        --An A-list name . number of times used
;        #frees = 0 => ['LAMBDA,[:v1,"$$"], :CDDR expandedFunction]
;        #frees = 1 =>
;          vec:=first first frees
;          ['LAMBDA,[:v1,vec], :CDDR expandedFunction]
;      scode:=nil
;      vec:=nil
;      locals:=nil
;      i:=-1
;      for v in frees repeat
;        i:=i+1
;        vec:=[first v,:vec]
;        scode:=[['SETQ,first v,[(($QuickCode => 'QREFELT;'ELT),"$$",i]],:scode]
;        locals:=[first v,:locals]
;      body:=CDDR expandedFunction
;      if locals then
;        if body is [['DECLARE,..],:] then
;          body:=[CAR body,['PROG,locals,:scode,['RETURN,['PROGN,:CDR body]]]]
;        else body:=[['PROG,locals,:scode,['RETURN,['PROGN,:body]]]]
;      vec:=['VECTOR,:NREVERSE vec]
;      ['LAMBDA,[:v1,"$$"],:body]
;      fname:=['CLOSEDFN,expandedFunction]
;      --Like QUOTE, but gets compiled
;      uu:=
;        frees => ['CONS,fname,vec]
;        ['LIST,fname]
;      [uu,m,oldE]

```

```

[isFunctor p188]
[get p??]
[qcar p??]
[qcdr p??]
[extendsCategoryForm p??]
[compLambda p233]
[stackAndThrow p??]
[take p??]
[compMakeDeclaration p429]
[hasFormalMapVariable p427]
[comp p403]
[extractCodeAndConstructTriple p427]
[optimizeFunctionDef p??]
[comp-tran p??]
[freelist p430]
[$formalArgList p??]
[$killOptimizeIfTrue p??]
[$funname p??]
[$funnameTail p??]
[$QuickCode p??]
[$EmptyMode p??]
[$FormalMapVariableList p202]
[$CategoryFrame p??]
[$formatArgList p??]

```

— **defun compWithMappingModel** —

```

(defun |compWithMappingModel| (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 (pairp tmp1)

```

```

      (progn
        (setq tmp2 (qcar tmp1))
        (and (pairp tmp2)
          (progn
            (setq tmp3 (qcar tmp2))
            (and (pairp tmp3)
              (progn
                (setq tmp4 (qcdr tmp3))
                (and (pairp tmp4)
                  (progn
                    (setq target (qcar tmp4))
                    (setq argModeList (qcdr tmp4))
                    t))))))
            (progn
              (setq tmp5 (qcdr tmp2))
              (and (pairp tmp5) (eq (qcdr tmp5) nil))))))
    (prog (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 (pairp form)
        (eq (qcar form) '+->))
        (progn
          (setq tmp1 (qcdr form))
          (and (pairp tmp1)
            (progn
              (setq v1 (qcar tmp1))
              (setq tmp2 (qcdr tmp1))
              (and (pairp tmp2)
                (eq (qcdr tmp2) nil)

```

```

                                (progn (setq nx (qcar tmp2)) t))))))
(setq oldstyle nil)
(cond
  ((and (pairp v1) (eq (qcar v1) '|:|))
    (setq ress (|compLambda| form mode oldE))
    ress)
  (t
    (setq v1
      (cond
        ((and (pairp v1)
          (eq (qcar v1) '|@Tuple|)
          (progn (setq v11 (qcdr 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 (|#| sl) |$FormalMapVariableList|)))
(cond
  (ress ress)
  (t
    (do ((t8 sl (cdr t8)) (m nil) (t9 v1 (cdr t9)) (v nil))
      ((or (atom t8)
        (progn (setq m (car t8)) nil)
        (atom t9)
        (progn (setq v (car t9)) nil))
        nil)
      (seq (exit (progn
        (setq tmp6
          (|compMakeDeclaration| (list '|:| v m ) |$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 '$$ (cons i nil)))
               nil)))
        scode))
    (setq locals (cons (car v) locals))))))
(setq body (cddr expandedFunction))
(cond
 (locals
  (cond
   ((and (pairp body)
        (progn
         (setq tmp1 (qcar body))
         (and (pairp tmp1)
              (eq (qcar tmp1) 'declare))))
   (setq body
    (cons (car body)
          (cons
           (cons 'prog
                (cons locals
                     (append scode
                           (cons
                            (cons 'return
                                (cons
                                 (cons 'progn
                                     (cdr body))
                                 nil))
                                nil))))
           nil))))
   (t
    (setq body
     (cons
      (cons 'prog
            (cons locals
                  (append scode
                        (cons
                         (cons 'return
                             (cons
                              (cons 'progn body)
                              nil))
                             nil))))
      nil))))))
    (setq vec (cons 'vector (nreverse vec)))
    (cons 'lambda (cons (append vl (list '$$) body))))))
(setq fname (list 'closedfn expandedFunction))
(setq uu

```

```

      (cond
        (frees (list 'cons fname vec))
        (t (list 'list fname))))
      (list uu mode oldE)))))))))

```

10.1.33 defun extractCodeAndConstructTriple

— defun extractCodeAndConstructTriple —

```

(defun |extractCodeAndConstructTriple| (form mode oldE)
  (let (tmp1 a fn op env)
    (cond
      ((and (pairp form) (eq (qcar form) '|call|))
        (progn
          (setq tmp1 (qcdr form))
          (and (pairp tmp1)
            (progn (setq fn (qcar tmp1)) t))))
      (cond
        ((and (pairp fn) (eq (qcar fn) '|applyFun|))
          (progn
            (setq tmp1 (qcdr fn))
            (and (pairp tmp1) (eq (qcdr tmp1) nil)
              (progn (setq a (qcar 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))))

```

10.1.34 defun hasFormalMapVariable

```

[hasFormalMapVariable ScanOrPairVec (vol5)]
[$formalMapVariables p??]

```

— defun hasFormalMapVariable —

```

(defun |hasFormalMapVariable| (x vl)
  (let (|$formalMapVariables|)
    (declare (special |$formalMapVariables|))

```

```
(when (setq |$formalMapVariables| v1)
  (|ScanOrPairVec| #'(lambda (y) (member y |$formalMapVariables|)) x)))
```

10.1.35 defun argsToSig

— defun argsToSig —

```
(defun |argsToSig| (args)
  (let (tmp1 v tmp2 tt sig1 arg1 bad)
    (cond
      ((and (pairp args) (eq (qcar args) '|:|))
        (progn
          (setq tmp1 (qcdr args))
          (and (pairp tmp1)
            (progn
              (setq v (qcar tmp1))
              (setq tmp2 (qcdr tmp1))
              (and (pairp tmp2)
                (eq (qcdr tmp2) nil)
                (progn
                  (setq tt (qcar tmp2))
                  t))))))
        (list (list v) (list tt)))
      (t
        (setq sig1 nil)
        (setq arg1 nil)
        (setq bad nil)
        (dolist (arg args)
          (cond
            ((and (pairp arg) (eq (qcar arg) '|:|))
              (progn
                (setq tmp1 (qcdr arg))
                (and (pairp tmp1)
                  (progn
                    (setq v (qcar tmp1))
                    (setq tmp2 (qcdr tmp1))
                    (and (pairp tmp2) (eq (qcdr tmp2) nil)
                      (progn
                        (setq tt (qcar tmp2))
                        t))))))
                (setq sig1 (cons tt sig1))
                (setq arg1 (cons v arg1))
                (t (setq bad t))))
            (t
              (setq bad t))))
        (cond
          (bad (list nil nil ))
```



```
(t (list (reverse arg1) (reverse sig1)))))))))
```

10.1.36 defun compMakeDeclaration

```
[compColon p215]
[$insideExpressionIfTrue p??]
```

— defun compMakeDeclaration —

```
(defun |compMakeDeclaration| (form mode env)
  (let (|$insideExpressionIfTrue|)
    (declare (special |$insideExpressionIfTrue|))
    (setq |$insideExpressionIfTrue| nil)
    (|compColon| form mode env)))
```

10.1.37 defun modifyModeStack

```
[say p??]
[copy p??]
[setelt p??]
[resolve p??]
[$reportExitModeStack p??]
[$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)))))
```

10.1.38 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 p430]

```
[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))))
```

10.1.39 defun compOrCroak1,compactify

```
[compOrCroak1,compactify p431]
[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))))))
```

10.1.40 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)))
```

10.1.41 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 p433]
[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|))))))

```

10.1.42 defun recompile-lib-file-if-necessary

```

[compile-lib-file p433]
[*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))))
    (unless (and ldate bdate (> bdate ldate))
      (compile-lib-file lfile)
      (list bfile))))

```

10.1.43 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)

```

10.1.44 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)))
```

10.1.45 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)))
```

10.1.46 defvar \$byConstructors

— initvars —

```
(defvar |$byConstructors| () "list of constructors to be compiled")
```

10.1.47 defvar \$constructorsSeen

— initvars —

```
(defvar |$constructorsSeen| () "list of constructors found")
```

— 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 addclose}
\getchunk{defun addConstructorModemaps}
\getchunk{defun addDomain}
\getchunk{defun addEltModemap}
\getchunk{defun addEmptyCapsuleIfNecessary}
\getchunk{defun addModemapKnown}
\getchunk{defun addModemap0}
\getchunk{defun addModemap1}
\getchunk{defun addNewDomain}
\getchunk{defun add-parens-and-semis-to-line}
\getchunk{defun Advance-Char}
\getchunk{defun advance-token}
\getchunk{defun aplTran}
\getchunk{defun aplTran1}
\getchunk{defun aplTranList}
\getchunk{defun argsToSig}
\getchunk{defun augLisplibModemapsFromCategory}
\getchunk{defun augmentLisplibModemapsFromFunctor}
\getchunk{defun augModemapsFromCategory}
\getchunk{defun augModemapsFromCategoryRep}
```

```

\getchunk{defun augModemapsFromDomain}
\getchunk{defun augModemapsFromDomain1}

\getchunk{defun blankp}
\getchunk{defun bumperrorcount}

\getchunk{defun char-eq}
\getchunk{defun char-ne}
\getchunk{defun checkWarning}
\getchunk{defun comma2Tuple}
\getchunk{defun comp}
\getchunk{defun comp2}
\getchunk{defun comp3}
\getchunk{defun compAdd}
\getchunk{defun compArgumentsAndTryAgain}
\getchunk{defun compAtom}
\getchunk{defun compAtSign}
\getchunk{defun compCapsule}
\getchunk{defun compCapsuleInner}
\getchunk{defun compCase}
\getchunk{defun compCase1}
\getchunk{defun compCat}
\getchunk{defun compCategory}
\getchunk{defun compDefineCategory1}
\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 compDefineCategory}
\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 compForm}
\getchunk{defun compForm1}
\getchunk{defun compForm2}
\getchunk{defun compFunctorBody}
\getchunk{defun compHas}
\getchunk{defun compIf}

```



```

\getchunk{defun compileFileQuietly}
\getchunk{defun compile-lib-file}
\getchunk{defun compiler}
\getchunk{defun compileDocumentation}
\getchunk{defun compilerDoit}
\getchunk{defun compileSpad2Cmd}
\getchunk{defun compileSpadLispCmd}
\getchunk{defun compImport}
\getchunk{defun compIs}
\getchunk{defun compJoin}
\getchunk{defun compLambda}
\getchunk{defun compLeave}
\getchunk{defun compList}
\getchunk{defun compMacro}
\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 setqSetelt}
\getchunk{defun setqSingle}
\getchunk{defun compSetq}
\getchunk{defun compSetq1}
\getchunk{defun compString}
\getchunk{defun compSubDomain}
\getchunk{defun compSubDomain1}
\getchunk{defun compSymbol}
\getchunk{defun compSubsetCategory}
\getchunk{defun compSuchthat}
\getchunk{defun compTopLevel}
\getchunk{defun compTypeOf}
\getchunk{defun compVector}
\getchunk{defun compWhere}
\getchunk{defun compWithMappingMode}
\getchunk{defun compWithMappingModel}
\getchunk{defun containsBang}
\getchunk{defun convert}
\getchunk{defun convertOpAlist2compilerInfo}
\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 dollarTran}
\getchunk{defun domainMember}
\getchunk{defun drop}

\getchunk{defun errhuh}
\getchunk{defun escape-keywords}
\getchunk{defun escaped}
\getchunk{defun evalAndRwriteLispForm}
\getchunk{defun evalAndSub}
\getchunk{defun extractCodeAndConstructTriple}

\getchunk{defun finalizeLisplib}
\getchunk{defun fincomblock}
\getchunk{defun floatexpid}
\getchunk{defun freelist}

\getchunk{defun get-a-line}
\getchunk{defun getCategoryOpsAndAtts}
\getchunk{defun getConstructorOpsAndAtts}
\getchunk{defun getDomainsInScope}
\getchunk{defun getFunctorOpsAndAtts}
\getchunk{defun getModemap}
\getchunk{defun getModemapList}
\getchunk{defun getModemapListFromDomain}
\getchunk{defun getOperationAlist}
\getchunk{defun getScriptName}
\getchunk{defun getSlotFromCategoryForm}
\getchunk{defun getSlotFromFunctor}
\getchunk{defun getTargetFromRhs}
\getchunk{defun get-token}
\getchunk{defun getToken}
\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 indent-pos}
\getchunk{defun infixtok}
\getchunk{defun initialize-preparse}
\getchunk{defun initial-substring}
\getchunk{defun initial-substring-p}
\getchunk{defun initializeLisplib}
\getchunk{defun is-console}
\getchunk{defun isDomainConstructorForm}
\getchunk{defun isDomainForm}
\getchunk{defun isFunctor}
\getchunk{defun isListConstructor}
\getchunk{defun isSuperDomain}
\getchunk{defun isTokenDelimiter}

\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 lisplibDoRename}
\getchunk{defun lisplibWrite}
\getchunk{defun loadIfNecessary}
\getchunk{defun loadLibIfNecessary}

\getchunk{defun macroExpand}
\getchunk{defun macroExpandInPlace}
\getchunk{defun macroExpandList}
\getchunk{defun makeCategoryPredicates}
\getchunk{defun makeFunctorArgumentParameters}
\getchunk{defun makeSimplePredicateOrNil}
\getchunk{defun make-string-adjustable}
\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 mergeModemap}
\getchunk{defun mergeSignatureAndLocalVarAlists}
\getchunk{defun meta-syntax-error}
\getchunk{defun mkCategoryPackage}
\getchunk{defun mkConstructor}

```

```

\getchunk{defun mkEvaluableCategoryForm}
\getchunk{defun mkNewModemapList}
\getchunk{defun mkOpVec}
\getchunk{defun modifyModeStack}

\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 optional}

\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 parseDollarGreaterThen}
\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 parseGreaterThen}
\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 push-reduction}
\getchunk{defun putDomainsInScope}

\getchunk{defun quote-if-string}

\getchunk{defun read-a-line}
\getchunk{defun recompile-lib-file-if-necessary}
\getchunk{defun /rf-1}
\getchunk{defun removeSuperfluousMapping}
\getchunk{defun reportOnFunctorCompilation}
\getchunk{defun /RQ,LIB}
\getchunk{defun rwriteLispForm}

\getchunk{defun setDefOp}
\getchunk{defun skip-blanks}
\getchunk{defun skip-ifblock}
\getchunk{defun skip-to-endif}
\getchunk{defun spad}
\getchunk{defun spad-fixed-arg}
\getchunk{defun stack-clear}
\getchunk{defun stack-load}
\getchunk{defun stack-pop}
\getchunk{defun stack-push}
\getchunk{defun storeblanks}
\getchunk{defun substituteCategoryArguments}
\getchunk{defun substNames}
\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 translabel}
\getchunk{defun translabel1}
\getchunk{defun try-get-token}
\getchunk{defun tuple2List}

\getchunk{defun underscore}
\getchunk{defun unget-tokens}
\getchunk{defun unTuple}
\getchunk{defun updateCategoryFrameForCategory}
\getchunk{defun updateCategoryFrameForConstructor}

```



```
\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 11

Index

Index

- `+- >`, 233
 - defplist, 233
- `- >`, 282
 - defplist, 282
- `<=`, 129
 - defplist, 129
- `==>`, 283
 - defplist, 283
- `=>`, 279
 - defplist, 279
- `>`, 116
 - defplist, 116
- `>=`, 114, 115
 - defplist, 114, 115
- `*comp370-apply*`
 - usedby spad, 395
- `*eof*`
 - usedby read-a-line, 88
 - usedby spad, 395
- `*lisp-bin-filetype*`
 - usedby recompile-lib-file-if-necessary, 433
- `*standard-output*`
 - usedby compileFileQuietly, 434
- `*terminal-io*`
 - usedby is-console, 373
- `., 274`
 - defplist, 274
- `/`, 289
 - defplist, 289
- `/RQ,LIB`, 385
 - calledby compilerDoit, 384
 - calls /rf-1, 385
 - calls echo-meta[5], 385
 - uses \$lisplib, 385
 - defun, 385
- `/editfile`
 - usedby /rf-1, 386
 - usedby compAdd, 205
 - usedby compFunctorBody, 176
 - usedby compileSpad2Cmd, 382
 - usedby compiler, 379
 - usedby initializeLisplib, 159
 - usedby spad, 395
- `/major-version`
 - usedby initializeLisplib, 159
- `/rf-1`, 386
 - calledby /RQ,LIB, 385
 - calls makeInputFilename[5], 386
 - calls ncINTERPFILE, 386
 - calls spad[5], 386
 - uses /editfile, 386
 - uses echo-meta, 386
 - defun, 386
- `/rf[5]`
 - called by compilerDoit, 384
- `/rq[5]`
 - called by compilerDoit, 384
- `:, 108, 215, 272`
 - defplist, 108, 215, 272
- `::, 107, 213, 273`
 - defplist, 107, 213, 273
- `:BF:, 267`
 - defplist, 267
- `;;, 287`
 - defplist, 287
- `==, 276`
 - defplist, 276
- `$/editfile`
 - local ref finalizeLisplib, 160
- `$Boolean`
 - usedby compCase1, 210
 - usedby compIf, 229
 - usedby compIs, 230
 - usedby compReduce1, 238

- usedby compRepeatOrCollect, 240
- usedby compSubDomain1, 251
- usedby compSuchthat, 253
- usedby compSymbol, 412
- \$CapsuleDomainsInScope
 - local def putDomainsInScope, 189
 - local ref getDomainsInScope, 189
- \$CapsuleModemapFrame
 - local def addModemapKnown, 196
- \$CategoryFrame
 - local def updateCategoryFrameForCategory, 121
 - local def updateCategoryFrameForConstructor, 120
 - local ref isFunctor, 188
 - local ref loadLibIfNecessary, 119
 - local ref mkEvalableCategoryForm, 148
 - local ref updateCategoryFrameForCategory, 121
 - local ref updateCategoryFrameForConstructor, 120
- usedby compDefineFunctor1, 168
- usedby compSubDomain1, 251
- usedby compWithMappingMode1, 422
- usedby parseHasRhs, 118
- usedby parseHas, 117
- \$CategoryNames
 - local ref mkEvalableCategoryForm, 148
- \$Category
 - local ref augModemapsFromDomain, 190
 - local ref mkEvalableCategoryForm, 148
 - usedby compConstructorCategory, 222
 - usedby compDefine1, 224
 - usedby compJoin, 231
- \$CheckVectorList
 - usedby compDefineFunctor1, 168
 - usedby displayMissingFunctions, 178
- \$ConditionalOperators
 - usedby genDomainOps, 182
 - usedby makeFunctorArgumentParameters, 179
- \$ConstructorNames
 - usedby compDefine1, 224
- \$DomainFrame
 - usedby s-process, 397
- \$DoubleFloat
 - usedby primitiveType, 411
- \$DummyFunctorNames
 - local ref augModemapsFromDomain, 190
- \$EchoLineStack
 - usedby fincomblock, 372
 - usedby preparse-echo, 88
 - usedby preparseReadLine1, 87
- \$EchoLines
 - usedby ncINTERPFILE, 431
- \$EmptyEnvironment
 - local ref augLisplibModemapsFromCategory, 153
 - usedby genDomainViewList, 181
 - usedby s-process, 397
- \$EmptyMode
 - local ref compileDocumentation, 158
 - local ref mkEvalableCategoryForm, 148
 - usedby compAdd, 205
 - usedby compArgumentsAndTryAgain, 418
 - usedby compCase1, 210
 - usedby compColonInside, 408
 - usedby compCons1, 219
 - usedby compDefine1, 224
 - usedby compDefineAddSignature, 141
 - usedby compDefineCategory1, 145
 - usedby compForm1, 415
 - usedby compForm2, 417
 - usedby compIs, 230
 - usedby compMacro, 235
 - usedby compNoStacking, 404
 - usedby compPretend, 236
 - usedby compSetq1, 246
 - usedby compSubDomain1, 251
 - usedby compWhere, 255
 - usedby compWithMappingMode1, 422
 - usedby primitiveType, 411
 - usedby s-process, 397
 - usedby setqSingle, 247
- \$EmptyVector
 - usedby compVector, 254
- \$Expression
 - usedby compAtom, 410
 - usedby compForm1, 415
 - usedby compSymbol, 412
- \$FormalMapVariableList, 202
 - local ref finalizeLisplib, 160

- local ref getSlotFromCategoryForm, 163
- local ref isDomainConstructorForm, 249
- usedby compColon, 216
- usedby compDefineCategory2, 150
- usedby compDefineFunctor1, 168
- usedby compSymbol, 412
- usedby compTypeOf, 408
- usedby compWithMappingMode1, 422
- usedby makeCategoryPredicates, 146
- usedby mkCategoryPackage, 147
- usedby mkOpVec, 183
- usedby substNames, 202
- defvar, 202
- \$Index
 - usedby s-process, 396
- \$InitialDomainsInScope
 - usedby spad, 395
- \$InteractiveFrame
 - usedby spad, 395
- \$InteractiveMode
 - local def addConstructorModemaps, 192
 - local ref isFunctor, 188
 - local ref loadLibIfNecessary, 119
 - local ref mkNewModemapList, 199
 - usedby bumperrorcount, 363
 - usedby compileFileQuietly, 434
 - usedby compileSpad2Cmd, 382
 - usedby dollarTran, 357
 - usedby parseAnd, 105
 - usedby parseAtSign, 106
 - usedby parseCoerce, 108
 - usedby parseColon, 108
 - usedby parseHas, 117
 - usedby parseIf, ifTran, 122
 - usedby parseNot, 132
 - usedby postBigFloat, 268
 - usedby postDef, 277
 - usedby postError, 262
 - usedby postMDef, 283
 - usedby postReduce, 285
 - usedby spad, 395
 - usedby tuple2List, 368
- \$LocalDomainAlist
 - usedby compDefineFunctor1, 168
- \$LocalFrame
 - usedby s-process, 397
- \$NRTaddForm
 - usedby compAdd, 205
 - usedby compDefineFunctor1, 168
 - usedby compFunctorBody, 176
 - usedby compSubDomain, 250
- \$NRTaddList
 - usedby compDefineFunctor1, 168
- \$NRTattributeAlist
 - usedby compDefineFunctor1, 168
- \$NRTbase
 - usedby compDefineFunctor1, 168
- \$NRTdeltaLength
 - usedby compDefineFunctor1, 168
- \$NRTdeltaListComp
 - usedby compDefineFunctor1, 168
- \$NRTdeltaList
 - usedby compDefineFunctor1, 168
- \$NRTderivedTargetIfTrue
 - usedby compTopLevel, 400
- \$NRTdomainFormList
 - usedby compDefineFunctor1, 168
- \$NRTloadTimeAlist
 - usedby compDefineFunctor1, 168
- \$NRTslot1Info
 - usedby compDefineFunctor1, 168
- \$NRTslot1PredicateList
 - local def finalizeLisplib, 161
 - usedby compDefineFunctor1, 168
- \$NegativeInteger
 - usedby primitiveType, 411
- \$NoValueMode
 - usedby compDefine1, 224
 - usedby compImport, 230
 - usedby compMacro, 235
 - usedby compRepeatOrCollect, 240
 - usedby compSeq1, 244
 - usedby compSymbol, 412
 - usedby setqSingle, 247
- \$NoValue
 - usedby compSymbol, 412
 - usedby parseAtom, 102
- \$NonNegativeInteger
 - usedby primitiveType, 411
- \$NumberOfArgsIfInteger
 - usedby compForm1, 415
- \$One

- usedby compElt, 226
- \$PatternVariableList
 - local ref augLisplibModemapsFromCategory, 153
 - local ref augmentLisplibModemapsFromFunctor, 174
- \$PolyMode
 - usedby s-process, 397
- \$PositiveInteger
 - usedby primitiveType, 411
- \$PrettyPrint
 - usedby print-defun, 399
- \$QuickCode
 - usedby compDefineFunctor1, 169
 - usedby compWithMappingMode1, 422
 - usedby compileSpad2Cmd, 382
- \$QuickLet
 - usedby compileSpad2Cmd, 382
 - usedby setqSingle, 247
- \$ReadingFile
 - usedby ncINTERPFILE, 431
- \$Representation
 - usedby compDefineFunctor1, 168
 - usedby compNoStacking, 404
- \$SpecialDomainNames
 - local ref isDomainForm, 249
 - usedby addEmptyCapsuleIfNecessary, 142
- \$StringCategory
 - usedby compString, 250
- \$String
 - usedby primitiveType, 411
- \$Symbol
 - usedby compSymbol, 412
- \$Translation
 - usedby s-process, 397
- \$TriangleVariableList
 - usedby compDefineCategory2, 150
 - usedby compForm2, 417
 - usedby makeCategoryPredicates, 146
- \$VariableCount
 - usedby s-process, 397
- \$Zero
 - usedby compElt, 226
- \$addFormLhs
 - usedby compAdd, 205
 - usedby compSubDomain, 250
- \$addForm
 - usedby compAdd, 205
 - usedby compCapsuleInner, 208
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 168
 - usedby compSubDomain, 250
- \$algebraOutputStream
 - local ref compDefineLisplib, 156
- \$alternateViewList
 - usedby makeFunctorArgumentParameters, 179
- \$attributesName
 - usedby compDefineFunctor1, 168
- \$base
 - local def augModemapsFromCategoryRep, 203
 - local def augModemapsFromCategory, 195
- \$bootStrapMode
 - usedby comp2, 405
 - usedby compAdd, 205
 - usedby compCapsule, 208
 - usedby compColon, 216
 - usedby compDefineCategory1, 145
 - usedby compDefineFunctor1, 168
 - usedby compFunctorBody, 176
- \$boot
 - usedby PARSE-FloatTok, 345
 - usedby PARSE-Primary1, 327
 - usedby PARSE-Quad, 331
 - usedby PARSE-Selector, 325
 - usedby PARSE-TokTail, 322
 - usedby aplTran1, 293
 - usedby aplTran, 293
 - usedby postAtom, 259
 - usedby postBigFloat, 268
 - usedby postColonColon, 273
 - usedby postDef, 277
 - usedby postForm, 262
 - usedby postIf, 279
 - usedby postMDef, 283
 - usedby quote-if-string, 348
 - usedby tuple2List, 368
- \$byConstructors, 434
 - usedby compilerDoit, 384
 - usedby prepare1, 81
- defvar, 434

- \$byteAddress
 - usedby compDefineFunctor1, 168
- \$byteVec
 - usedby compDefineFunctor1, 168
- \$categoryPredicateList
 - usedby compDefineCategory1, 145
 - usedby mkCategoryPackage, 147
- \$comblocklist, 371
 - usedby fincomblock, 372
 - usedby preparse, 77
 - defvar, 371
- \$compErrorMessageStack
 - usedby compOrCroak1, 402
- \$compForModeIfTrue
 - usedby compSymbol, 412
- \$compStack
 - local def compOrCroak1, 402
 - local ref compNoStacking1, 404
 - local ref compNoStacking, 404
 - local ref comp, 403
- \$compTimeSum
 - usedby compTopLevel, 400
- \$compUniquelyIfTrue
 - usedby s-process, 397
- \$compileDocumentation
 - local ref compDefineLisplib, 155
- \$compileOnlyCertainItems
 - usedby compDefineFunctor1, 168
 - usedby compileSpad2Cmd, 382
- \$condAlist
 - usedby compDefineFunctor1, 168
- \$constructorLineNumber
 - usedby preparse, 77
- \$constructorsSeen, 434
 - usedby compilerDoit, 384
 - usedby preparse1, 81
 - defvar, 434
- \$currentFunction
 - usedby s-process, 397
- \$defOp
 - usedby parseTransform, 101
 - usedby postError, 262
 - usedby postTransformCheck, 261
 - usedby setDefOp, 292
- \$definition
 - usedby compDefineCategory2, 150
- \$defstack, 299
 - defvar, 299
- \$docList
 - usedby postDef, 277
 - usedby preparse, 77
- \$domainShell
 - local ref augLisplibModemapsFromCategory, 153
 - usedby compDefineCategory2, 150
 - usedby compDefineCategory, 155
 - usedby compDefineFunctor1, 168
 - usedby compDefineFunctor, 166
 - usedby getOperationAlist, 201
- \$echolinestack, 72
 - usedby initialize-preparse, 73
 - usedby preparse1, 81
 - defvar, 72
- \$endTestList
 - usedby compReduce1, 238
- \$envHashTable
 - usedby compTopLevel, 400
- \$env
 - usedby displayMissingFunctions, 178
- \$erase
 - local ref initializeLisplib, 159
- \$exitModeStack
 - usedby compExit, 227
 - usedby compLeave, 234
 - usedby compOrCroak1, 402
 - usedby compRepeatOrCollect, 240
 - usedby compReturn, 243
 - usedby compSeq1, 244
 - usedby compSeq, 244
 - usedby comp, 403
 - usedby modifyModeStack, 429
 - usedby s-process, 397
- \$exitMode
 - usedby s-process, 397
- \$extraParms
 - usedby compDefineCategory2, 150
- \$e
 - local def addEltModemap, 197
 - local def augmentLisplibModemapsFromFunctor, 174
 - local def mkEvaluableCategoryForm, 148
 - local ref addModemapKnown, 196

- local ref augmentLisplibModemapsFrom-Functor, 174
 - local ref compileDocumentation, 158
 - local ref getSlotFromFunctor, 165
 - local ref mkEvaluableCategoryForm, 148
 - usedby comp3, 406
 - usedby compHas, 228
 - usedby compReduce1, 238
 - usedby genDomainOps, 182
 - usedby genDomainView, 181
 - usedby getOperationAlist, 201
 - usedby mkCategoryPackage, 147
 - usedby s-process, 397
- \$fcopy
 - local ref compileDocumentation, 158
- \$filep
 - local ref compDefineLisplib, 155
- \$finalEnv
 - usedby compSeq1, 244
- \$forceAdd
 - local ref mergeModemap, 199
 - local ref mkNewModemapList, 199
 - usedby compTopLevel, 400
 - usedby makeFunctorArgumentParameters, 179
- \$formalArgList
 - usedby compDefine1, 223
 - usedby compDefineCategory2, 150
 - usedby compReduce, 238
 - usedby compRepeatOrCollect, 241
 - usedby compSymbol, 412
 - usedby compWithMappingMode1, 422
 - usedby compWithMappingMode, 419
 - usedby displayMissingFunctions, 178
- \$formalMapVariables
 - local def hasFormalMapVariable, 427
- \$formatArgList
 - usedby compWithMappingMode1, 422
- \$form
 - usedby compCapsuleInner, 208
 - usedby compDefine1, 224
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 168
 - usedby s-process, 397
 - usedby setqSingle, 247
- \$frontier
 - usedby compDefineCategory2, 150
- \$functionLocations
 - local ref transformOperationAlist, 164
 - usedby compDefineFunctor1, 168
- \$functionStats
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 168
 - usedby reportOnFunctorCompilation, 177
- \$functorForm
 - local ref addModemap0, 196
 - usedby compAdd, 205
 - usedby compCapsule, 208
 - usedby compDefineFunctor1, 168
 - usedby compFunctorBody, 176
 - usedby getOperationAlist, 201
- \$functorLocalParameters
 - usedby compCapsuleInner, 209
 - usedby compDefineFunctor1, 168
 - usedby compSymbol, 412
- \$functorSpecialCases
 - usedby compDefineFunctor1, 168
- \$functorStats
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 168
 - usedby reportOnFunctorCompilation, 177
- \$functorTarget
 - usedby compDefineFunctor1, 168
- \$functorsUsed
 - usedby compDefineFunctor1, 168
- \$funnameTail
 - usedby compWithMappingMode1, 422
- \$funname
 - usedby compWithMappingMode1, 422
- \$f
 - usedby compileSpad2Cmd, 382
- \$genFVar
 - usedby compDefineFunctor1, 168
 - usedby s-process, 397
- \$genSDVar
 - usedby compDefineFunctor1, 168
 - usedby s-process, 397
- \$genno
 - local def aplTran, 293
- \$getDomainCode
 - usedby compCapsuleInner, 208
 - usedby compDefineCategory2, 150

- usedby compDefineFunctor1, 168
 - usedby genDomainOps, 182
 - usedby genDomainView, 181
- \$goGetList
 - usedby compDefineFunctor1, 168
- \$headerDocumentation
 - usedby postDef, 277
 - usedby preparse, 77
- \$index, 72
 - usedby initialize-preparse, 73
 - usedby preparseReadLine1, 87
 - usedby preparse, 77
 - defvar, 72
- \$initList
 - usedby compReduce1, 238
- \$insideCapsuleFunctionIfTrue
 - local ref CapsuleModemapFrame, 196
 - local ref addEltModemap, 197
 - local ref getDomainsInScope, 189
 - local ref putDomainsInScope, 189
 - usedby compDefine1, 224
 - usedby s-process, 397
- \$insideCategoryIfTrue
 - usedby compCapsuleInner, 208
 - usedby compColon, 216
 - usedby compDefine1, 224
 - usedby compDefineCategory2, 150
 - usedby s-process, 397
- \$insideCategoryPackageIfTrue
 - usedby compCapsuleInner, 208
 - usedby compDefineCategory1, 145
 - usedby compDefineFunctor1, 168
- \$insideCoerceInteractiveHardIfTrue
 - usedby s-process, 397
- \$insideCompTypeOf
 - usedby comp3, 406
 - usedby compTypeOf, 408
- \$insideConstructIfTrue
 - usedby parseColon, 108
 - usedby parseConstruct, 103
- \$insideExpressionIfTrue
 - usedby compCapsule, 208
 - usedby compColon, 216
 - usedby compDefine1, 223, 224
 - usedby compExpression, 414
 - usedby compMakeDeclaration, 429
 - usedby compSeq1, 244
 - usedby compWhere, 255
 - usedby s-process, 397
- \$insideFunctorIfTrue
 - usedby compColon, 216
 - usedby compDefine1, 224
 - usedby compDefineCategory, 155
 - usedby compDefineFunctor1, 168
 - usedby getOperationAlist, 201
 - usedby s-process, 397
- \$insidePostCategoryIfTrue
 - usedby postCategory, 269
 - usedby postWith, 292
- \$insideSetqSingleIfTrue
 - usedby setqSingle, 247
- \$insideWhereIfTrue
 - usedby compDefine1, 224
 - usedby compWhere, 255
 - usedby s-process, 397
- \$is-eqlist, 300
 - defvar, 300
- \$is-gensymlist, 300
 - defvar, 300
- \$is-spill-list, 299
 - defvar, 299
- \$is-spill, 299
 - defvar, 299
- \$isOpPackageName
 - usedby compDefineFunctor1, 168
- \$killOptimizeIfTrue
 - usedby compTopLevel, 400
 - usedby compWithMappingMode1, 422
- \$leaveLevelStack
 - usedby compLeave, 234
 - usedby compRepeatOrCollect, 241
 - usedby s-process, 397
- \$leaveMode
 - usedby s-process, 397
- \$level
 - usedby compOrCroak1, 402
- \$lhsOfColon
 - local def evalAndSub, 201
 - usedby compColon, 216
 - usedby compSubsetCategory, 252
- \$lhs
 - usedby parseDEF, 109

- usedby parseMDEF, 131
- \$libFile
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref finalizeLisplib, 160
 - local ref initializeLisplib, 159
 - local ref rwriteLispForm, 154
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 168
- \$linelist, 72
 - usedby initialize-prepare, 73
 - usedby prepare1, 81
 - usedby prepareReadLine1, 87
 - defvar, 72
- \$line
 - usedby Advance-Char, 93
 - usedby current-char, 355
 - usedby line-advance-char, 91
 - usedby line-at-end-p, 90
 - usedby line-clear, 90
 - usedby line-new-line, 92
 - usedby line-next-char, 91
 - usedby line-past-end-p, 91
 - usedby line-print, 90, 92
 - usedby match-advance-string, 347
 - usedby match-string, 346
- \$lisplibAbbreviation
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref finalizeLisplib, 161
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 168
- \$lisplibAncestors
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref finalizeLisplib, 161
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 168
- \$lisplibAttributes
 - local ref finalizeLisplib, 161
- \$lisplibCategoriesExtended
 - local def compDefineLisplib, 156
 - usedby compDefineFunctor1, 168
- \$lisplibCategory
 - local def compDefineLisplib, 156
 - local def finalizeLisplib, 161
- local ref finalizeLisplib, 160
- usedby compDefineCategory1, 145
- usedby compDefineCategory2, 150
- usedby compDefineCategory, 155
- usedby compDefineFunctor1, 168
- \$lisplibForm
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref finalizeLisplib, 160
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 169
- \$lisplibFunctionLocations
 - usedby compDefineFunctor1, 169
- \$lisplibKind
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref compDefineLisplib, 156
 - local ref finalizeLisplib, 160
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 169
- \$lisplibMissingFunctions
 - usedby compDefineFunctor1, 169
- \$lisplibModemapAlist
 - local def augLisplibModemapsFromCategory, 153
 - local def augmentLisplibModemapsFromFunctor, 174
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref augLisplibModemapsFromCategory, 153
 - local ref augmentLisplibModemapsFromFunctor, 174
 - local ref finalizeLisplib, 160
- \$lisplibModemap
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref finalizeLisplib, 160
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 169
- \$lisplibOpAlist
 - local def initializeLisplib, 159
- \$lisplibOperationAlist
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref getSlotFromFunctor, 165

- usedby compDefineFunctor1, 169
- \$lisplibParents
 - local def compDefineLisplib, 156
 - local ref finalizeLisplib, 161
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 169
- \$lisplibPredicates
 - local def compDefineLisplib, 156
 - local ref finalizeLisplib, 161
- \$lisplibSignatureAlist
 - local def initializeLisplib, 159
 - local ref finalizeLisplib, 160
- \$lisplibSlot1
 - local def compDefineLisplib, 156
 - local ref finalizeLisplib, 161
 - usedby compDefineFunctor1, 169
- \$lisplibSuperDomain
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref finalizeLisplib, 160
 - usedby compSubDomain1, 251
- \$lisplibVariableAlist
 - local def compDefineLisplib, 156
 - local def initializeLisplib, 159
 - local ref finalizeLisplib, 160
- \$lisplib
 - local def compDefineLisplib, 156
 - local ref lisplibWrite, 166
 - local ref rwriteLispForm, 154
 - usedby /RQ,LIB, 385
 - usedby comp2, 405
 - usedby compDefineCategory2, 150
 - usedby compDefineCategory, 155
 - usedby compDefineFunctor1, 167
 - usedby compDefineFunctor, 166
- \$lookupFunction
 - usedby compDefineFunctor1, 169
- \$macroIfTrue
 - usedby compDefine, 223
 - usedby compMacro, 235
- \$macroassoc
 - usedby s-process, 397
- \$maxSignatureLineNumber
 - usedby postDef, 277
 - usedby preparse, 77
- \$mutableDomains
 - usedby compDefineFunctor1, 169
- \$mutableDomain
 - usedby compDefineFunctor1, 169
- \$mvl
 - usedby makeCategoryPredicates, 146
- \$myFunctorBody
 - usedby compDefineFunctor1, 169
- \$m
 - usedby compileSpad2Cmd, 382
- \$newCompilerUnionFlag
 - usedby compColonInside, 408
 - usedby compPretend, 236
- \$newComp
 - usedby compileSpad2Cmd, 382
- \$newConlist
 - local def compDefineLisplib, 156
 - local ref compDefineLisplib, 156
 - usedby compileSpad2Cmd, 382
 - usedby compiler, 379
- \$newspad
 - usedby s-process, 397
- \$noEnv
 - usedby compColon, 216
- \$noParseCommands
 - usedby PARSE-SpecialCommand, 311
- \$noSubsumption
 - usedby spad, 395
- \$options
 - usedby compileSpad2Cmd, 382
 - usedby compileSpadLispCmd, 432
 - usedby compiler, 379
 - usedby mkCategoryPackage, 147
- \$op
 - local def compDefineLisplib, 156
 - usedby compDefine1, 224
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 169
 - usedby compSubDomain1, 251
 - usedby parseDollarGreaterEqual, 112
 - usedby parseDollarGreaterThan, 112
 - usedby parseDollarLessEqual, 113
 - usedby parseDollarNotEqual, 114
 - usedby parseGreaterEqual, 115
 - usedby parseGreaterThan, 116
 - usedby parseLessEqual, 130
 - usedby parseNotEqual, 133

- usedby parseTran, 101
- usedby reportOnFunctorCompilation, 177
- \$packagesUsed
 - usedby comp2, 405
 - usedby compAdd, 205
 - usedby compDefine, 223
 - usedby compTopLevel, 400
- \$pairlis
 - local def finalizeLisplib, 161
 - usedby compDefineFunctor1, 169
- \$postStack
 - usedby displayPreCompilationErrors, 362
 - usedby postError, 262
 - usedby s-process, 397
- \$predAlist
 - usedby compDefWhereClause, 184
- \$prefix
 - usedby compDefine1, 224
 - usedby compDefineCategory2, 150
- \$preparse-last-line, 72
 - usedby initialize-preparse, 73
 - usedby preparse1, 81
 - usedby preparseReadLine1, 87
 - usedby preparse, 77
 - defvar, 72
- \$preparseReportIfTrue
 - usedby preparse, 77
- \$previousTime
 - usedby s-process, 397
- \$profileAlist
 - usedby compDefineFunctor, 166
- \$profileCompiler
 - local ref finalizeLisplib, 161
 - usedby compDefineFunctor, 166
 - usedby setqSingle, 247
- \$reportExitModeStack
 - usedby modifyModeStack, 429
- \$resolveTimeSum
 - usedby compTopLevel, 400
- \$returnMode
 - usedby compReturn, 243
 - usedby s-process, 397
- \$scanIfTrue
 - usedby compOrCroak1, 402
 - usedby compileSpad2Cmd, 382
- \$semanticErrorStack
 - usedby reportOnFunctorCompilation, 177
 - usedby s-process, 397
- \$setelt
 - usedby compDefineFunctor1, 169
- \$sideEffectsList
 - usedby compReduce1, 238
- \$sigAlist
 - usedby compDefWhereClause, 184
- \$signature
 - usedby compCapsuleInner, 208
 - usedby compDefineFunctor1, 169
- \$skipme
 - usedby preparse1, 81
 - usedby preparse, 77
- \$sourceFileTypes
 - usedby compileSpad2Cmd, 382
- \$spad-errors
 - usedby bumperrorcount, 363
- \$spadLibFT
 - local ref compDefineLisplib, 156
 - local ref compileDocumentation, 158
 - local ref finalizeLisplib, 161
 - local ref lisplibDoRename, 158
- \$spad
 - usedby quote-if-string, 348
- \$stack
 - usedby reduce-stack, 360
 - usedby stack-/empty, 95
 - usedby stack-clear, 95
 - usedby stack-load, 95
 - usedby stack-pop, 96
 - usedby stack-push, 96
- \$s
 - usedby compOrCroak1, 402
- \$template
 - usedby compDefineFunctor1, 169
- \$tokenCommands
 - usedby PARSE-SpecialCommand, 311
- \$token
 - usedby current-token, 97
 - usedby make-symbol-of, 352
 - usedby match-advance-string, 347
 - usedby next-token, 97
 - usedby prior-token, 97
 - usedby token-install, 98
 - usedby token-print, 98

- usedby valid-tokens, 98
- \$top-level
 - usedby compDefineCategory2, 150
 - usedby compDefineFunctor1, 168
 - usedby s-process, 397
- \$topOp
 - usedby displayPreCompilationErrors, 362
 - usedby s-process, 397
 - usedby setDefOp, 292
- \$tripleCache
 - usedby compDefine, 223
- \$tripleHits
 - usedby compDefine, 223
- \$tv1
 - usedby makeCategoryPredicates, 146
- \$uncondAlist
 - usedby compDefineFunctor1, 169
- \$until
 - usedby compReduce1, 238
 - usedby compRepeatOrCollect, 240
- \$viewNames
 - usedby compDefineFunctor1, 169
- \$v1, 300
 - defvar, 300
- \$warningStack
 - usedby reportOnFunctorCompilation, 177
 - usedby s-process, 397
- , 31
- abbreviation?
 - calledby parseHasRhs, 118
- abbreviationsSpad2Cmd
 - calledby mkCategoryPackage, 146
- action, 359
 - calledby PARSE-AnyId, 336
 - calledby PARSE-Category, 316
 - calledby PARSE-CommandTail, 313
 - calledby PARSE-Data, 334
 - calledby PARSE-FloatExponent, 329
 - calledby PARSE-GlyphTok, 336
 - calledby PARSE-Infix, 321
 - calledby PARSE-NBGlyphTok, 335
 - calledby PARSE-NewExpr, 309
 - calledby PARSE-OpenBrace, 338
 - calledby PARSE-OpenBracket, 338
 - calledby PARSE-Operation, 319
 - calledby PARSE-Prefix, 320
 - calledby PARSE-ReductionOp, 323
 - calledby PARSE-Sexpr1, 334
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-SpecialKeyWord, 310
 - calledby PARSE-Suffix, 340
 - calledby PARSE-TokTail, 322
 - calledby PARSE-TokenCommandTail, 312
 - calledby PARSE-TokenList, 312
 - defun, 359
- add, 264
 - defplist, 264
- add-parens-and-semis-to-line, 84
 - calledby parsepiles, 84
 - calls addclose, 84
 - calls drop, 84
 - calls infixtok, 84
 - calls nonblankloc, 84
 - defun, 84
- addBinding
 - calledby addModemap1, 198
 - calledby compDefineCategory2, 149
- addBinding[5]
 - called by setqSingle, 247
 - called by spad, 395
- addclose, 370
 - calledby add-parens-and-semis-to-line, 84
 - calls suffix, 370
 - defun, 370
- addConstructorModemaps, 192
 - calledby augModemapsFromDomain1, 191
 - calls addModemap, 192
 - calls get1, 192
 - calls msubst, 192
 - calls pairp, 192
 - calls putDomainsInScope, 192
 - calls qcar, 192
 - calls qcdr, 192
 - local def \$InteractiveMode, 192
 - defun, 192
- addContour
 - calledby compWhere, 255
- addDomain, 186
 - calledby comp2, 405
 - calledby comp3, 406

- calledby compAtSign, 207
- calledby compCapsule, 208
- calledby compCase, 209
- calledby compCoerce, 213
- calledby compColonInside, 408
- calledby compColon, 216
- calledby compElt, 225
- calledby compForm1, 415
- calledby compImport, 230
- calledby compPretend, 236
- calledby compSubDomain1, 251
- calls addNewDomain, 187
- calls constructor?, 187
- calls domainMember, 187
- calls getDomainsInScope, 186
- calls getmode, 187
- calls identp, 186
- calls isCategoryForm, 187
- calls isFunctor, 187
- calls isLiteral, 187
- calls member, 187
- calls pairp, 187
- calls qslessp, 186
- calls unknownTypeError, 187
- defun, 186
- addEltModemap, 197
 - calledby addModemap0, 196
 - calls addModemap1, 197
 - calls makeLiteral, 197
 - calls pairp, 197
 - calls qcar, 197
 - calls qcdr, 197
 - calls systemErrorHere, 197
 - local def \$e, 197
 - local ref \$insideCapsuleFunctionIfTrue, 197
 - defun, 197
- addEmptyCapsuleIfNecessary, 142
 - calledby compDefine1, 223
 - calls kar, 142
 - uses \$SpecialDomainNames, 142
 - defun, 142
- addInformation
 - calledby compCapsuleInner, 208
- addModemap
 - calledby addConstructorModemaps, 192
 - calledby augModemapsFromCategoryRep, 203
 - calledby genDomainOps, 182
 - calledby updateCategoryFrameForCategory, 121
 - calledby updateCategoryFrameForConstructor, 120
- addModemap0, 196
 - calledby addModemapKnown, 196
 - calls addEltModemap, 196
 - calls addModemap1, 196
 - calls pairp, 196
 - calls qcar, 196
 - local ref \$functorForm, 196
 - defun, 196
- addModemap1, 198
 - calledby addEltModemap, 197
 - calledby addModemap0, 196
 - calls addBinding, 198
 - calls augProplist, 198
 - calls getProplist, 198
 - calls lassoc, 198
 - calls mkNewModemapList, 198
 - calls msubst, 198
 - calls unErrorRef, 198
 - defun, 198
- addModemapKnown, 196
 - calledby augModemapsFromCategory, 195
 - calls addModemap0, 196
 - local def \$CapsuleModemapFrame, 196
 - local ref \$e, 196
 - defun, 196
- addNewDomain, 190
 - calledby addDomain, 187
 - calledby augModemapsFromDomain, 190
 - calls augModemapsFromDomain, 190
 - defun, 190
- addoptions
 - calledby initializeLisplib, 159
- addStats
 - calledby reportOnFunctorCompilation, 177
- Advance-Char, 93
 - calls Line-Advance-Char, 93
 - calls Line-At-End-P, 93
 - calls current-char, 93
 - calls next-line, 93

- uses \$line, 93
 - defun, 93
- advance-char
 - calledby skip-blanks, 346
- advance-token, 354
 - calledby PARSE-AnyId, 336
 - calledby PARSE-FloatExponent, 329
 - calledby PARSE-GlyphTok, 336
 - calledby PARSE-Infix, 321
 - calledby PARSE-NBGlyphTok, 335
 - calledby PARSE-OpenBrace, 338
 - calledby PARSE-OpenBracket, 338
 - calledby PARSE-Prefix, 321
 - calledby PARSE-ReductionOp, 323
 - calledby PARSE-Suffix, 340
 - calledby PARSE-TokenList, 312
 - calledby parse-argument-designator, 367
 - calledby parse-identifier, 365
 - calledby parse-keyword, 366
 - calledby parse-number, 366
 - calledby parse-spadstring, 365
 - calledby parse-string, 365
 - calls copy-token, 354
 - calls current-token, 354
 - calls try-get-token, 354
 - uses current-token, 354
 - uses valid-tokens, 354
 - defun, 354
- allLASSOCs
 - calledby augmentLisplibModemapsFrom-Functor, 174
- and, 105
 - defplist, 105
- aplTran, 293
 - calledby postTransform, 257
 - calls aplTran1, 293
 - calls containsBang, 293
 - local def \$genno, 293
 - uses \$boot, 293
 - defun, 293
- aplTran1, 293
 - calledby aplTran1, 293
 - calledby aplTranList, 295
 - calledby aplTran, 293
 - calledby hasAplExtension, 295
 - calls aplTran1, 293
 - calls aplTranList, 293
 - calls hasAplExtension, 293
 - calls nreverse0, 293
 - calls , 293
 - uses \$boot, 293
 - defun, 293
- aplTranList, 295
 - calledby aplTran1, 293
 - calledby aplTranList, 295
 - calls aplTran1, 295
 - calls aplTranList, 295
 - defun, 295
- applyMapping
 - calledby comp3, 406
- argsToSig, 428
 - calledby compLambda, 233
 - defun, 428
- assignError
 - calledby setqSingle, 247
- assoc
 - calledby augModemapsFromCategoryRep, 203
 - calledby compColon, 215
 - calledby compDefineAddSignature, 141
 - calledby compForm2, 416
 - calledby mkCategoryPackage, 146
 - calledby mkNewModemapList, 198
 - calledby mkOpVec, 183
 - calledby transformOperationAlist, 164
- AssocBarGensym
 - calledby mkOpVec, 183
- assocleft
 - calledby compDefWhereClause, 184
- assocright
 - calledby compDefWhereClause, 184
- assq
 - calledby makeFunctorArgumentParameters, 178
 - calledby mkOpVec, 183
- assq[5]
 - called by freelist, 430
- atEndOfLine
 - calledby PARSE-TokenCommandTail, 312
- augLisplibModemapsFromCategory, 152
 - calledby compDefineCategory2, 149
 - calls interactiveModemapForm, 153

- calls isCategoryForm, 152
- calls lassoc, 153
- calls member, 153
- calls mkAlistOfExplicitCategoryOps, 152
- calls mkpf, 153
- calls sublis, 152
- local def \$lisplibModemapAlist, 153
- local ref \$EmptyEnvironment, 153
- local ref \$PatternVariableList, 153
- local ref \$domainShell, 153
- local ref \$lisplibModemapAlist, 153
- defun, 152
- augmentLisplibModemapsFromFunctor, 174
 - calledby compDefineFunctor1, 167
 - calls allLASSOCs, 174
 - calls formal2Pattern, 174
 - calls interactiveModemapForm, 174
 - calls listOfPatternIds, 174
 - calls member, 174
 - calls mkAlistOfExplicitCategoryOps, 174
 - calls mkDatabasePred, 174
 - calls mkpf, 174
 - calls msubst, 174
 - local def \$e, 174
 - local def \$lisplibModemapAlist, 174
 - local ref \$PatternVariableList, 174
 - local ref \$e, 174
 - local ref \$lisplibModemapAlist, 174
 - defun, 174
- augModemapsFromCategory, 195
 - calledby augModemapsFromDomain1, 191
 - calledby compDefineFunctor1, 167
 - calledby genDomainView, 181
 - calls addModemapKnown, 195
 - calls compilerMessage, 195
 - calls evalAndSub, 195
 - calls putDomainsInScope, 195
 - local def \$base, 195
 - defun, 195
- augModemapsFromCategoryRep, 203
 - calledby compDefineFunctor1, 167
 - calls addModemap, 203
 - calls assoc, 203
 - calls compilerMessage, 203
 - calls evalAndSub, 203
 - calls isCategory, 203
 - calls msubst, 203
 - calls putDomainsInScope, 203
 - local def \$base, 203
 - defun, 203
- augModemapsFromDomain, 190
 - calledby addNewDomain, 190
 - calls addNewDomain, 190
 - calls augModemapsFromDomain1, 190
 - calls getDomainsInScope, 190
 - calls getdatabase, 190
 - calls kar, 190
 - calls listOrVectorElementNode, 190
 - calls member, 190
 - calls opOf, 190
 - calls stripUnionTags, 190
 - local ref \$Category, 190
 - local ref \$DummyFunctorNames, 190
 - defun, 190
- augModemapsFromDomain1, 191
 - calledby augModemapsFromDomain, 190
 - calledby compForm1, 414
 - calledby setqSingle, 247
 - calls addConstructorModemaps, 191
 - calls augModemapsFromCategory, 191
 - calls getl, 191
 - calls getmodeOrMapping, 191
 - calls getmode, 191
 - calls kar, 191
 - calls stackMessage, 191
 - calls substituteCategoryArguments, 191
 - defun, 191
- augProplist
 - calledby addModemap1, 198
- Bang, 358
 - defmacro, 358
- bang
 - calledby PARSE-Category, 315
 - calledby PARSE-CommandTail, 313
 - calledby PARSE-Conditional, 343
 - calledby PARSE-Form, 323
 - calledby PARSE-Import, 317
 - calledby PARSE-IteratorTail, 339
 - calledby PARSE-Seg, 342
 - calledby PARSE-Sexpr1, 335
 - calledby PARSE-SpecialCommand, 311

- calledby PARSE-TokenCommandTail, 311
- bf-
 - calledby PARSE-FloatTok, 345
- blankp, 371
 - calledby nonblankloc, 374
 - defun, 371
- Block, 268
 - defplist, 268
- bootStrapError
 - calledby compCapsule, 208
 - calledby compFunctorBody, 176
- bright
 - calledby compDefineLisplib, 155
 - calledby displayMissingFunctions, 177
 - calledby parseInBy, 126
 - calledby postForm, 262
- browserAutoloadOnceTrigger
 - calledby compileSpad2Cmd, 382
- bumperrorcount, 363
 - calledby postError, 262
 - uses \$InteractiveMode, 363
 - uses \$spad-errors, 363
 - defun, 363
- canReturn
 - calledby compIf, 228
- capsule, 207
 - defplist, 207
- CapsuleModemapFrame
 - local ref \$insideCapsuleFunctionIfTrue, 196
- case, 209
 - defplist, 209
- catches
 - compOrCroak1, 402
 - preparse1, 81
 - spad, 395
- category, 106, 212, 269
 - defplist, 106, 212, 269
- char-eq, 356
 - calledby PARSE-FloatBase, 328
 - calledby PARSE-TokTail, 322
 - defun, 356
- char-ne, 356
 - calledby PARSE-FloatBase, 328
 - calledby PARSE-Selector, 325
- defun, 356
- chaseInferences
 - calledby compHas, 228
- checkWarning, 367
 - calledby postCapsule, 265
 - calls concat, 367
 - calls postError, 367
 - defun, 367
- coerce
 - calledby compAtSign, 207
 - calledby compCase, 209
 - calledby compCoerce1, 214
 - calledby compCoerce, 213
 - calledby compColonInside, 408
 - calledby compForm1, 414
 - calledby compHas, 228
 - calledby compIf, 229
 - calledby compIs, 230
 - calledby convert, 411
- coerceable
 - calledby compForm1, 414
- coerceByModemap
 - calledby compCoerce1, 214
- coerceExit
 - calledby compRepeatOrCollect, 240
- collect, 240, 271
 - calledby floatexpid, 357
 - defplist, 240, 271
- comma2Tuple, 274
 - calledby postComma, 274
 - calledby postConstruct, 275
 - calls postFlatten, 274
 - defun, 274
- comp, 403
 - calledby compAdd, 205
 - calledby compArgumentsAndTryAgain, 418
 - calledby compAtSign, 207
 - calledby compCase1, 210
 - calledby compCoerce1, 214
 - calledby compColonInside, 408
 - calledby compCons1, 219
 - calledby compDefWhereClause, 184
 - calledby compDefineAddSignature, 141
 - calledby compExit, 227
 - calledby compForm1, 414
 - calledby compIs, 230

- calledby compLeave, 234
- calledby compList, 413
- calledby compOrCroak1, 402
- calledby compPretend, 236
- calledby compReduce1, 238
- calledby compRepeatOrCollect, 240
- calledby compReturn, 242
- calledby compSeqItem, 245
- calledby compSubsetCategory, 252
- calledby compSuchthat, 253
- calledby compVector, 254
- calledby compWhere, 255
- calledby compWithMappingMode1, 422
- calledby setqSetelt, 246
- calledby setqSingle, 247
- calls compNoStacking, 403
- local ref \$compStack, 403
- uses \$exitModeStack, 403
- defun, 403
- comp-tran
 - calledby compWithMappingMode1, 422
- comp2, 405
 - calledby compNoStacking1, 404
 - calledby compNoStacking, 404
 - calls addDomain, 405
 - calls comp3, 405
 - calls insert, 405
 - calls isDomainForm, 405
 - calls isFunctor, 405
 - calls nequal, 405
 - calls opOf, 405
 - uses \$bootStrapMode, 405
 - uses \$lisplib, 405
 - uses \$packagesUsed, 405
 - defun, 405
- comp3, 405
 - calledby comp2, 405
 - calledby compTypeOf, 408
 - calls addDomain, 406
 - calls applyMapping, 406
 - calls compApply, 406
 - calls compAtom, 406
 - calls compCoerce, 406
 - calls compColon, 406
 - calls compExpression, 406
 - calls compTypeOf, 406
 - calls compWithMappingMode, 406
 - calls getDomainsInScope, 406
 - calls getmode, 406
 - calls member[5], 406
 - calls pname[5], 406
 - calls stringPrefix?, 406
 - uses \$e, 406
 - uses \$insideCompTypeOf, 406
 - defun, 405
- compAdd, 205
 - calls NRTgetLocalIndex, 205
 - calls compCapsule, 205
 - calls compOrCroak, 205
 - calls compSubDomain1, 205
 - calls compTuple2Record, 205
 - calls comp, 205
 - calls nreverse0, 205
 - calls pairp, 205
 - calls qcar, 205
 - calls qcdr, 205
 - uses /editfile, 205
 - uses \$EmptyMode, 205
 - uses \$NRTaddForm, 205
 - uses \$addFormLhs, 205
 - uses \$addForm, 205
 - uses \$bootStrapMode, 205
 - uses \$functorForm, 205
 - uses \$packagesUsed, 205
 - defun, 205
- compApply
 - calledby comp3, 406
- compApplyModemap
 - calledby getModemap, 193
- compArgumentsAndTryAgain, 418
 - calledby compForm, 414
 - calls compForm1, 418
 - calls comp, 418
 - uses \$EmptyMode, 418
 - defun, 418
- compAtom, 409
 - calledby comp3, 406
 - calls compAtomWithModemap, 409
 - calls compList, 409
 - calls compSymbol, 409
 - calls compVector, 409
 - calls convert, 409

- calls get, 409
- calls isSymbol, 409
- calls modeIsAggregateOf, 409
- calls primitiveType, 409
- uses \$Expression, 410
- defun, 409
- compAtomWithModemap
 - calledby compAtom, 409
- compAtSign, 207
 - calledby compLambda, 233
 - calls addDomain, 207
 - calls coerce, 207
 - calls comp, 207
 - defun, 207
- compBoolean
 - calledby compIf, 229
- compCapsule, 208
 - calledby compAdd, 205
 - calledby compSubDomain, 250
 - calls addDomain, 208
 - calls bootStrapError, 208
 - calls compCapsuleInner, 208
 - uses \$bootStrapMode, 208
 - uses \$functorForm, 208
 - uses \$insideExpressionIfTrue, 208
 - uses editfile, 208
 - defun, 208
- compCapsuleInner, 208
 - calledby compCapsule, 208
 - calls addInformation, 208
 - calls compCapsuleItems, 208
 - calls mkpf, 208
 - calls processFunctorOrPackage, 208
 - uses \$addForm, 208
 - uses \$form, 208
 - uses \$functorLocalParameters, 209
 - uses \$getDomainCode, 208
 - uses \$insideCategoryIfTrue, 208
 - uses \$insideCategoryPackageIfTrue, 208
 - uses \$signature, 208
 - defun, 208
- compCapsuleItems
 - calledby compCapsuleInner, 208
- compCase, 209
 - calls addDomain, 209
 - calls coerce, 209
 - calls compCase1, 209
 - defun, 209
- compCase1, 210
 - calledby compCase, 209
 - calls comp, 210
 - calls getModemapList, 210
 - calls modeEqual, 210
 - calls nreverse0, 210
 - uses \$Boolean, 210
 - uses \$EmptyMode, 210
 - defun, 210
- compCat, 211
 - calls get1, 211
 - defun, 211
- compCategory, 212
 - calls compCategoryItem, 212
 - calls mkExplicitCategoryFunction, 212
 - calls qcar, 212
 - calls qcdr, 212
 - calls resolve, 212
 - calls systemErrorHere, 212
 - defun, 212
- compCategoryItem
 - calledby compCategory, 212
- compCoerce, 213
 - calledby comp3, 406
 - calls addDomain, 213
 - calls coerce, 213
 - calls compCoerce1, 213
 - calls getmode, 213
 - defun, 213
- compCoerce1, 214
 - calledby compCoerce, 213
 - calls coerceByModemap, 214
 - calls coerce, 214
 - calls comp, 214
 - calls mkq, 214
 - calls msubst, 214
 - calls resolve, 214
 - defun, 214
- compColon, 215
 - calledby comp3, 406
 - calledby compColon, 216
 - calledby compMakeDeclaration, 429
 - calls addDomain, 216
 - calls assoc, 215

- calls compColonInside, 215
- calls compColon, 216
- calls eqsubstlist, 216
- calls genSomeVariable, 216
- calls getDomainsInScope, 215
- calls getmode, 216
- calls isCategoryForm, 216
- calls isDomainForm, 215, 216
- calls length, 216
- calls makeCategoryForm, 216
- calls member[5], 216
- calls nreverse0, 216
- calls put, 216
- calls systemErrorHere, 216
- calls take, 216
- calls unknownTypeError, 216
- uses \$FormalMapVariableList, 216
- uses \$bootStrapMode, 216
- uses \$insideCategoryIfTrue, 216
- uses \$insideExpressionIfTrue, 216
- uses \$insideFunctorIfTrue, 216
- uses \$lhsOfColon, 216
- uses \$noEnv, 216
- defun, 215
- compColonInside, 408
 - calledby compColon, 215
 - calls addDomain, 408
 - calls coerce, 408
 - calls comp, 408
 - calls opOf, 408
 - calls stackSemanticError, 408
 - calls stackWarning, 408
 - uses \$EmptyMode, 408
 - uses \$newCompilerUnionFlag, 408
 - defun, 408
- compCons, 219
 - calls compCons1, 219
 - calls compForm, 219
 - defun, 219
- compCons1, 219
 - calledby compCons, 219
 - calls comp, 219
 - calls convert, 219
 - calls pairp, 219
 - calls qcar, 219
 - calls qcdr, 219
 - uses \$EmptyMode, 219
 - defun, 219
- compConstruct, 220
 - calls compForm, 220
 - calls compList, 220
 - calls compVector, 220
 - calls convert, 220
 - calls getDomainsInScope, 220
 - calls modelsAggregateOf, 220
 - defun, 220
- compConstructorCategory, 222
 - calls resolve, 222
 - uses \$Category, 222
 - defun, 222
- compDefine, 223
 - calls compDefine1, 223
 - uses \$macroIfTrue, 223
 - uses \$packagesUsed, 223
 - uses \$tripleCache, 223
 - uses \$tripleHits, 223
 - defun, 223
- compDefine1, 223
 - calledby compDefine1, 223
 - calledby compDefineCategory1, 145
 - calledby compDefine, 223
 - calls addEmptyCapsuleIfNecessary, 223
 - calls compDefWhereClause, 223
 - calls compDefine1, 223
 - calls compDefineAddSignature, 223
 - calls compDefineCapsuleFunction, 223
 - calls compDefineCategory, 223
 - calls compDefineFunctor, 223
 - calls compInternalFunction, 223
 - calls getAbbreviation, 223
 - calls getSignatureFromMode, 223
 - calls getTargetFromRhs, 223
 - calls giveFormalParametersValues, 223
 - calls isDomainForm, 223
 - calls isMacro, 223
 - calls length, 223
 - calls macroExpand, 223
 - calls stackAndThrow, 223
 - calls strconc, 223
 - uses \$Category, 224
 - uses \$ConstructorNames, 224
 - uses \$EmptyMode, 224

- uses \$NoValueMode, 224
 - uses \$formalArgList, 223
 - uses \$form, 224
 - uses \$insideCapsuleFunctionIfTrue, 224
 - uses \$insideCategoryIfTrue, 224
 - uses \$insideExpressionIfTrue, 223, 224
 - uses \$insideFunctorIfTrue, 224
 - uses \$insideWhereIfTrue, 224
 - uses \$op, 224
 - uses \$prefix, 224
 - defun, 223
- compDefineAddSignature, 141
 - calledby compDefine1, 223
 - calls assoc, 141
 - calls comp, 141
 - calls getProplist, 141
 - calls hasFullSignature, 141
 - calls lassoc, 141
 - uses \$EmptyMode, 141
 - defun, 141
- compDefineCapsuleFunction
 - calledby compDefine1, 223
- compDefineCategory, 155
 - calledby compDefine1, 223
 - calls compDefineCategory1, 155
 - calls compDefineLisplib, 155
 - uses \$domainShell, 155
 - uses \$insideFunctorIfTrue, 155
 - uses \$lisplibCategory, 155
 - uses \$lisplib, 155
 - defun, 155
- compDefineCategory1, 145
 - calledby compDefineCategory, 155
 - calls compDefine1, 145
 - calls compDefineCategory2, 145
 - calls makeCategoryPredicates, 145
 - calls mkCategoryPackage, 145
 - uses \$EmptyMode, 145
 - uses \$bootStrapMode, 145
 - uses \$categoryPredicateList, 145
 - uses \$insideCategoryPackageIfTrue, 145
 - uses \$lisplibCategory, 145
 - defun, 145
- compDefineCategory2, 149
 - calledby compDefineCategory1, 145
 - calls addBinding, 149
 - calls augLisplibModemapsFromCategory, 149
 - calls compMakeDeclaration, 149
 - calls compOrCroak, 149
 - calls compile, 149
 - calls computeAncestorsOf, 149
 - calls constructor?, 149
 - calls evalAndRwriteLispForm, 149
 - calls eval, 149
 - calls getArgumentModeOrMoan, 149
 - calls getParentsFor, 149
 - calls giveFormalParametersValues, 149
 - calls lisplibWrite, 149
 - calls mkConstructor, 149
 - calls mkq, 149
 - calls nequal, 149
 - calls opOf, 149
 - calls optFunctorBody, 149
 - calls removeZeroOne, 149
 - calls sublis, 149
 - calls take, 149
 - uses \$FormalMapVariableList, 150
 - uses \$TriangleVariableList, 150
 - uses \$addForm, 150
 - uses \$definition, 150
 - uses \$domainShell, 150
 - uses \$extraParms, 150
 - uses \$formalArgList, 150
 - uses \$form, 150
 - uses \$frontier, 150
 - uses \$functionStats, 150
 - uses \$functorStats, 150
 - uses \$getDomainCode, 150
 - uses \$insideCategoryIfTrue, 150
 - uses \$libFile, 150
 - uses \$lisplibAbbreviation, 150
 - uses \$lisplibAncestors, 150
 - uses \$lisplibCategory, 150
 - uses \$lisplibForm, 150
 - uses \$lisplibKind, 150
 - uses \$lisplibModemap, 150
 - uses \$lisplibParents, 150
 - uses \$lisplib, 150
 - uses \$op, 150
 - uses \$prefix, 150
 - uses \$top-level, 150

- defun, 149
- compDefineFunctor, 166
 - calledby compDefine1, 223
 - calls compDefineFunctor1, 166
 - calls compDefineLisplib, 166
 - uses \$domainShell, 166
 - uses \$lisplib, 166
 - uses \$profileAlist, 166
 - uses \$profileCompiler, 166
 - defun, 166
- compDefineFunctor1, 167
 - calledby compDefineFunctor, 166
 - calls NRTgenInitialAttributeAlist, 167
 - calls NRTgetLocalIndex, 167
 - calls NRTgetLookupFunction, 167
 - calls NRTmakeSlot1Info, 167
 - calls augModemapsFromCategoryRep, 167
 - calls augModemapsFromCategory, 167
 - calls augmentLisplibModemapsFromFunctor, 167
 - calls compFunctorBody, 167
 - calls compMakeCategoryObject, 167
 - calls compMakeDeclaration, 167
 - calls compile, 167
 - calls computeAncestorsOf, 167
 - calls constructor?, 167
 - calls disallowNilAttribute, 167
 - calls evalAndRwriteLispForm, 167
 - calls getArgumentModeOrMoan, 167
 - calls getModemap, 167
 - calls getParentsFor, 167
 - calls getdatabase, 167
 - calls giveFormalParametersValues, 167
 - calls isCategoryPackageName, 167
 - calls isPackageFunction, 167
 - calls lisplibWrite, 167
 - calls makeFunctorArgumentParameters, 167
 - calls maxindex, 167
 - calls mkq, 167
 - calls modemap2Signature, 167
 - calls nequal, 167
 - calls pairp, 167
 - calls pname, 167
 - calls pp, 167
 - calls qcar, 167
 - calls qcdr, 167
 - calls remdup, 167
 - calls removeZeroOne, 167
 - calls reportOnFunctorCompilation, 167
 - calls sayBrightly, 167
 - calls simpBool, 167
 - calls strconc, 167
 - calls sublis, 167
 - uses \$CategoryFrame, 168
 - uses \$CheckVectorList, 168
 - uses \$FormalMapVariableList, 168
 - uses \$LocalDomainAlist, 168
 - uses \$NRTaddForm, 168
 - uses \$NRTaddList, 168
 - uses \$NRTattributeAlist, 168
 - uses \$NRTbase, 168
 - uses \$NRTdeltaLength, 168
 - uses \$NRTdeltaListComp, 168
 - uses \$NRTdeltaList, 168
 - uses \$NRTdomainFormList, 168
 - uses \$NRTloadTimeAlist, 168
 - uses \$NRTslot1Info, 168
 - uses \$NRTslot1PredicateList, 168
 - uses \$QuickCode, 169
 - uses \$Representation, 168
 - uses \$addForm, 168
 - uses \$attributesName, 168
 - uses \$bootStrapMode, 168
 - uses \$byteAddress, 168
 - uses \$byteVec, 168
 - uses \$compileOnlyCertainItems, 168
 - uses \$condAlist, 168
 - uses \$domainShell, 168
 - uses \$form, 168
 - uses \$functionLocations, 168
 - uses \$functionStats, 168
 - uses \$functorForm, 168
 - uses \$functorLocalParameters, 168
 - uses \$functorSpecialCases, 168
 - uses \$functorStats, 168
 - uses \$functorTarget, 168
 - uses \$functorsUsed, 168
 - uses \$genFVar, 168
 - uses \$genSDVar, 168
 - uses \$getDomainCode, 168
 - uses \$goGetList, 168

- uses \$insideCategoryPackageIfTrue, 168
- uses \$insideFunctorIfTrue, 168
- uses \$isOpPackageName, 168
- uses \$libFile, 168
- uses \$lisplibAbbreviation, 168
- uses \$lisplibAncestors, 168
- uses \$lisplibCategoriesExtended, 168
- uses \$lisplibCategory, 168
- uses \$lisplibForm, 169
- uses \$lisplibFunctionLocations, 169
- uses \$lisplibKind, 169
- uses \$lisplibMissingFunctions, 169
- uses \$lisplibModemap, 169
- uses \$lisplibOperationAlist, 169
- uses \$lisplibParents, 169
- uses \$lisplibSlot1, 169
- uses \$lisplib, 167
- uses \$lookupFunction, 169
- uses \$mutableDomains, 169
- uses \$mutableDomain, 169
- uses \$myFunctorBody, 169
- uses \$op, 169
- uses \$pairlis, 169
- uses \$setelt, 169
- uses \$signature, 169
- uses \$template, 169
- uses \$top-level, 168
- uses \$uncondAlist, 169
- uses \$viewNames, 169
- defun, 167
- compDefineLisplib, 155
 - calledby compDefineCategory, 155
 - calledby compDefineFunctor, 166
 - calls bright, 155
 - calls compileDocumentation, 155
 - calls filep, 155
 - calls fillerSpaces, 155
 - calls finalizeLisplib, 155
 - calls getConstructorAbbreviation, 155
 - calls getdatabase, 155
 - calls lisplibDoRename, 155
 - calls localdatabase, 155
 - calls rpackfile, 155
 - calls rshut, 155
 - calls sayMSG, 155
 - calls unloadOneConstructor, 155
 - calls updateCategoryFrameForCategory, 155
 - calls updateCategoryFrameForConstructor, 155
 - local def \$libFile, 156
 - local def \$lisplibAbbreviation, 156
 - local def \$lisplibAncestors, 156
 - local def \$lisplibCategoriesExtended, 156
 - local def \$lisplibCategory, 156
 - local def \$lisplibForm, 156
 - local def \$lisplibKind, 156
 - local def \$lisplibModemapAlist, 156
 - local def \$lisplibModemap, 156
 - local def \$lisplibOperationAlist, 156
 - local def \$lisplibParents, 156
 - local def \$lisplibPredicates, 156
 - local def \$lisplibSlot1, 156
 - local def \$lisplibSuperDomain, 156
 - local def \$lisplibVariableAlist, 156
 - local def \$lisplib, 156
 - local def \$newConlist, 156
 - local def \$op, 156
 - local ref \$algebraOutputStream, 156
 - local ref \$compileDocumentation, 155
 - local ref \$filep, 155
 - local ref \$lisplibKind, 156
 - local ref \$newConlist, 156
 - local ref \$spadLibFT, 156
 - defun, 155
- compDefWhereClause, 184
 - calledby compDefine1, 223
 - calls assocleft, 184
 - calls assocright, 184
 - calls comp, 184
 - calls concat, 184
 - calls delete, 184
 - calls getmode, 184
 - calls lassoc, 184
 - calls listOffidentifiersIn, 184
 - calls orderByDependency, 184
 - calls pairList, 184
 - calls pairp, 184
 - calls qcar, 184
 - calls qcdr, 184
 - calls union, 184
 - calls userError, 184

- uses \$predAlist, 184
 - uses \$sigAlist, 184
 - defun, 184
- compElt, 225
 - calls addDomain, 225
 - calls compForm, 225
 - calls convert, 225
 - calls getDeltaEntry, 225
 - calls getModemapListFromDomain, 225
 - calls isDomainForm, 225
 - calls length, 225
 - calls nequal, 226
 - calls opOf, 225
 - calls stackMessage, 225
 - calls stackWarning, 225
 - uses \$One, 226
 - uses \$Zero, 226
 - defun, 225
- compExit, 227
 - calls comp, 227
 - calls modifyModeStack, 227
 - calls stackMessageIfNone, 227
 - uses \$exitModeStack, 227
 - defun, 227
- compExpression, 413
 - calledby comp3, 406
 - calls compForm, 414
 - calls getl, 413
 - uses \$insideExpressionIfTrue, 414
 - defun, 413
- compExpressionList
 - calledby compForm1, 414
- compForm, 414
 - calledby compConstruct, 220
 - calledby compCons, 219
 - calledby compElt, 225
 - calledby compExpression, 414
 - calls compArgumentsAndTryAgain, 414
 - calls compForm1, 414
 - calls stackMessageIfNone, 414
 - defun, 414
- compForm1, 414
 - calledby compArgumentsAndTryAgain, 418
 - calledby compForm, 414
 - calls addDomain, 415
 - calls augModemapsFromDomain1, 414
 - calls coerceable, 414
 - calls coerce, 414
 - calls compExpressionList, 414
 - calls compForm2, 414
 - calls compOrCroak, 414
 - calls compToApply, 415
 - calls comp, 414
 - calls getFormModemaps, 414
 - calls length, 414
 - calls nreverse0, 414
 - calls outputComp, 414
 - uses \$EmptyMode, 415
 - uses \$Expression, 415
 - uses \$NumberOfArgsIfInteger, 415
 - defun, 414
- compForm2, 416
 - calledby compForm1, 414
 - calls PredImplies, 416
 - calls assoc, 416
 - calls compForm3, 417
 - calls compFormPartiallyBottomUp, 417
 - calls compUniquely, 416
 - calls isSimple, 416
 - calls length, 416
 - calls nreverse0, 416
 - calls sublis, 416
 - calls take, 416
 - uses \$EmptyMode, 417
 - uses \$TriangleVariableList, 417
 - defun, 416
- compForm3
 - calledby compForm2, 417
- compForMode
 - calledby compJoin, 231
- compFormPartiallyBottomUp
 - calledby compForm2, 417
- compFromIf
 - calledby compIf, 229
- compFunctorBody, 176
 - calledby compDefineFunctor1, 167
 - calls bootStrapError, 176
 - calls compOrCroak, 176
 - uses /editfile, 176
 - uses \$NRTaddForm, 176
 - uses \$bootStrapMode, 176
 - uses \$functorForm, 176

- defun, 176
- compHas, 228
 - calls chaseInferences, 228
 - calls coerce, 228
 - calls compHasFormat, 228
 - uses \$e, 228
 - defun, 228
- compHasFormat
 - calledby compHas, 228
- compIf, 228
 - calls canReturn, 228
 - calls coerce, 229
 - calls compBoolean, 229
 - calls compFromIf, 229
 - calls intersectionEnvironment, 229
 - calls quotify, 229
 - calls resolve, 229
 - uses \$Boolean, 229
 - defun, 228
- compile
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
- compile-lib-file, 433
 - calledby recompile-lib-file-if-necessary, 433
 - defun, 433
- compileDocumentation, 158
 - calledby compDefineLisplib, 155
 - calls finalizeDocumentation, 158
 - calls lisplibWrite, 158
 - calls make-input-filename, 158
 - calls rdefiostream, 158
 - calls replaceFile, 158
 - calls rpackfile, 158
 - calls rshut, 158
 - local ref \$EmptyMode, 158
 - local ref \$e, 158
 - local ref \$fcopy, 158
 - local ref \$spadLibFT, 158
 - defun, 158
- compileFileQuietly, 434
 - uses *standard-output*, 434
 - uses \$InteractiveMode, 434
 - defun, 434
- compiler, 378
 - calls compileSpad2Cmd, 379
 - calls compileSpadLispCmd, 379
 - calls findfile, 379
 - calls helpSpad2Cmd[5], 379
 - calls mergePathnames[5], 379
 - calls namestring[5], 379
 - calls pathnameType[5], 379
 - calls pathname[5], 379
 - calls selectOptionLC[5], 379
 - calls throwKeyedMsg, 379
 - uses /editfile, 379
 - uses \$newConlist, 379
 - uses \$options, 379
 - defun, 378
- compilerDoit, 384
 - calledby compileSpad2Cmd, 382
 - calls /RQ.LIB, 384
 - calls /rf[5], 384
 - calls /rq[5], 384
 - calls member[5], 384
 - calls opOf, 384
 - calls sayBrightly, 384
 - uses \$byConstructors, 384
 - uses \$constructorsSeen, 384
 - defun, 384
- compilerDoitWithScreenedLisplib
 - calledby compileSpad2Cmd, 382
- compilerMessage
 - calledby augModemapsFromCategoryRep, 203
 - calledby augModemapsFromCategory, 195
- compileSpad2Cmd, 380
 - calledby compiler, 379
 - calls browserAutoloadOnceTrigger, 382
 - calls compilerDoitWithScreenedLisplib, 382
 - calls compilerDoit, 382
 - calls convertSpadToAsFile, 382
 - calls error, 382
 - calls extendLocalLibdb, 382
 - calls namestring[5], 382
 - calls nequal, 382
 - calls object2String, 382
 - calls pathnameType[5], 382
 - calls pathname[5], 382
 - calls sayKeyedMsg[5], 382
 - calls selectOptionLC[5], 382
 - calls spad2AsTranslatorAutoloadOnceTrigger, 382

- calls spadPrompt, 382
- calls strconc, 382
- calls terminateSystemCommand[5], 382
- calls throwKeyedMsg, 382
- calls updateSourceFiles[5], 382
- uses /editfile, 382
- uses \$InteractiveMode, 382
- uses \$QuickCode, 382
- uses \$QuickLet, 382
- uses \$compileOnlyCertainItems, 382
- uses \$f, 382
- uses \$m, 382
- uses \$newComp, 382
- uses \$newConlist, 382
- uses \$options, 382
- uses \$scanIfTrue, 382
- uses \$sourceFileTypes, 382
- defun, 380
- compileSpadLispCmd, 431
 - calledby compiler, 379
 - calls fnameMake[5], 431
 - calls fnameReadable?[5], 432
 - calls localdatabase[5], 432
 - calls namestring[5], 431
 - calls object2String, 432
 - calls pathnameDirectory[5], 432
 - calls pathnameName[5], 432
 - calls pathnameType[5], 431
 - calls pathname[5], 431
 - calls recompile-lib-file-if-necessary, 432
 - calls sayKeyedMsg[5], 432
 - calls selectOptionLC[5], 431
 - calls spadPrompt, 432
 - calls terminateSystemCommand[5], 431
 - calls throwKeyedMsg, 432
 - uses \$options, 432
 - defun, 431
- compImport, 230
 - calls addDomain, 230
 - uses \$NoValueMode, 230
 - defun, 230
- compInternalFunction
 - calledby compDefine1, 223
- compIs, 230
 - calls coerce, 230
 - calls comp, 230
 - uses \$Boolean, 230
 - uses \$EmptyMode, 230
 - defun, 230
- compIterator
 - calledby compReduce1, 238
 - calledby compRepeatOrCollect, 240
- compJoin, 231
 - calls compForMode, 231
 - calls compJoin, getParms, 231
 - calls convert, 231
 - calls isCategoryForm, 231
 - calls nreverse0, 231
 - calls pairp, 231
 - calls qcar, 231
 - calls qcdr, 231
 - calls stackSemanticError, 231
 - calls union, 231
 - calls wrapDomainSub, 231
 - uses \$Category, 231
 - defun, 231
- compJoin, getParms
 - calledby compJoin, 231
- compLambda, 233
 - calledby compWithMappingMode1, 422
 - calls argsToSig, 233
 - calls compAtSign, 233
 - calls qcar, 233
 - calls qcdr, 233
 - calls stackAndThrow, 233
 - defun, 233
- compLeave, 234
 - calls comp, 234
 - calls modifyModeStack, 234
 - uses \$exitModeStack, 234
 - uses \$leaveLevelStack, 234
 - defun, 234
- compList, 413
 - calledby compAtom, 409
 - calledby compConstruct, 220
 - calls comp, 413
 - defun, 413
- compMacro, 235
 - calls formatUnabbreviated, 235
 - calls macroExpand, 235
 - calls put, 235
 - calls qcar, 235

- calls sayBrightly, 235
- uses \$EmptyMode, 235
- uses \$NoValueMode, 235
- uses \$macroIfTrue, 235
- defun, 235
- compMakeCategoryObject
 - calledby compDefineFunctor1, 167
 - calledby getOperationAlist, 201
 - calledby getSlotFromFunctor, 165
- compMakeDeclaration, 429
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
 - calledby compSetq1, 246
 - calledby compSubDomain1, 251
 - calledby compWithMappingMode1, 422
 - calls compColon, 429
 - uses \$insideExpressionIfTrue, 429
 - defun, 429
- compNoStacking, 404
 - calledby comp, 403
 - calls comp2, 404
 - calls compNoStacking1, 404
 - local ref \$compStack, 404
 - uses \$EmptyMode, 404
 - uses \$Representation, 404
 - defun, 404
- compNoStacking1, 404
 - calledby compNoStacking, 404
 - calls comp2, 404
 - calls get, 404
 - local ref \$compStack, 404
 - defun, 404
- compOrCroak, 401
 - calledby compAdd, 205
 - calledby compDefineCategory2, 149
 - calledby compForm1, 414
 - calledby compFunctorBody, 176
 - calledby compRepeatOrCollect, 240
 - calledby compSubDomain1, 251
 - calledby compTopLevel, 400
 - calledby getTargetFromRhs, 142
 - calledby mkEvalableCategoryForm, 148
 - calls compOrCroak1, 401
 - defun, 401
- compOrCroak1, 402
 - calledby compOrCroak, 401
- calls compOrCroak1, compactify, 402
- calls comp, 402
- calls displayComp, 402
- calls displaySemanticErrors, 402
- calls mkErrorExpr, 402
- calls say, 402
- calls stackSemanticError, 402
- calls userError, 402
- local def \$compStack, 402
- uses \$compErrorMessageStack, 402
- uses \$exitModeStack, 402
- uses \$level, 402
- uses \$scanIfTrue, 402
- uses \$s, 402
- catches, 402
- defun, 402
- compOrCroak1, compactify, 431
 - calledby compOrCroak1, compactify, 431
 - calledby compOrCroak1, 402
 - calls compOrCroak1, compactify, 431
 - calls lassoc, 431
 - defun, 431
- compPretend, 236
 - calls addDomain, 236
 - calls comp, 236
 - calls nequal, 236
 - calls opOf, 236
 - calls stackSemanticError, 236
 - calls stackWarning, 236
 - uses \$EmptyMode, 236
 - uses \$newCompilerUnionFlag, 236
 - defun, 236
- compQuote, 237
 - defun, 237
- compReduce, 238
 - calls compReduce1, 238
 - uses \$formalArgList, 238
 - defun, 238
- compReduce1, 238
 - calledby compReduce, 238
 - calls compIterator, 238
 - calls comp, 238
 - calls getIdentity, 238
 - calls msubst, 238
 - calls nreverse0, 238
 - calls parseTran, 238

- calls `systemError`, 238
- uses `$Boolean`, 238
- uses `$endTestList`, 238
- uses `$e`, 238
- uses `$initList`, 238
- uses `$sideEffectsList`, 238
- uses `$until`, 238
- defun, 238
- `compRepeatOrCollect`, 240
 - calls `coerceExit`, 240
 - calls `compIterator`, 240
 - calls `compOrCroak`, 240
 - calls `comp`, 240
 - calls `length`, 240
 - calls `modeIsAggregateOf`, 240
 - calls `msubst`, 240
 - calls `stackMessage`, 240
 - calls `,`, 240
 - uses `$Boolean`, 240
 - uses `$NoValueMode`, 240
 - uses `$exitModeStack`, 240
 - uses `$formalArgList`, 241
 - uses `$leaveLevelStack`, 241
 - uses `$until`, 240
 - defun, 240
- `compReturn`, 242
 - calls `comp`, 242
 - calls `modifyModeStack`, 243
 - calls `nequal`, 242
 - calls `resolve`, 242
 - calls `stackSemanticError`, 242
 - calls `userError`, 242
 - uses `$exitModeStack`, 243
 - uses `$returnMode`, 243
 - defun, 242
- `compSeq`, 244
 - calls `compSeq1`, 244
 - uses `$exitModeStack`, 244
 - defun, 244
- `compSeq1`, 244
 - calledby `compSeq`, 244
 - calls `compSeqItem`, 244
 - calls `mkq`, 244
 - calls `nreverse0`, 244
 - calls `replaceExitEtc`, 244
 - uses `$NoValueMode`, 244
 - uses `$exitModeStack`, 244
 - uses `$finalEnv`, 244
 - uses `$insideExpressionIfTrue`, 244
 - defun, 244
- `compSeqItem`, 245
 - calledby `compSeq1`, 244
 - calls `comp`, 245
 - calls `macroExpand`, 245
 - defun, 245
- `compSetq`, 245
 - calledby `compSetq1`, 246
 - calls `compSetq1`, 245
 - defun, 245
- `compSetq1`, 246
 - calledby `compSetq`, 245
 - calls `compMakeDeclaration`, 246
 - calls `compSetq`, 246
 - calls `identp[5]`, 246
 - calls `qcar`, 246
 - calls `qcdr`, 246
 - calls `setqMultiple`, 246
 - calls `setqSetelt`, 246
 - calls `setqSingle`, 246
 - uses `$EmptyMode`, 246
 - defun, 246
- `compString`, 250
 - calls `resolve`, 250
 - uses `$StringCategory`, 250
 - defun, 250
- `compSubDomain`, 250
 - calls `compCapsule`, 250
 - calls `compSubDomain1`, 250
 - uses `$NRTaddForm`, 250
 - uses `$addFormLhs`, 250
 - uses `$addForm`, 250
 - defun, 250
- `compSubDomain1`, 251
 - calledby `compAdd`, 205
 - calledby `compSubDomain`, 250
 - calls `addDomain`, 251
 - calls `compMakeDeclaration`, 251
 - calls `compOrCroak`, 251
 - calls `evalAndRwriteLispForm`, 251
 - calls `lispize`, 251
 - calls `stackSemanticError`, 251
 - uses `$Boolean`, 251

- uses `$CategoryFrame`, 251
 - uses `$EmptyMode`, 251
 - uses `$lisplibSuperDomain`, 251
 - uses `$op`, 251
 - defun, 251
- `compSubsetCategory`, 252
 - calls `comp`, 252
 - calls `msubst`, 252
 - calls `put`, 252
 - uses `$lhsOfColon`, 252
 - defun, 252
- `compSuchthat`, 253
 - calls `comp`, 253
 - calls `put`, 253
 - uses `$Boolean`, 253
 - defun, 253
- `compSymbol`, 411
 - calledby `compAtom`, 409
 - calls `NRTgetLocalIndex`, 412
 - calls `errorRef`, 412
 - calls `getmode`, 411
 - calls `get`, 412
 - calls `isFunction`, 412
 - calls `member[5]`, 412
 - calls `stackMessage`, 412
 - uses `$Boolean`, 412
 - uses `$Expression`, 412
 - uses `$FormalMapVariableList`, 412
 - uses `$NoValueMode`, 412
 - uses `$NoValue`, 412
 - uses `$Symbol`, 412
 - uses `$compForModeIfTrue`, 412
 - uses `$formalArgList`, 412
 - uses `$functorLocalParameters`, 412
 - defun, 411
- `compToApply`
 - calledby `compForm1`, 415
- `compTopLevel`, 400
 - calledby `s-process`, 396
 - calls `compOrCroak`, 400
 - calls `newComp`, 400
 - uses `$NRTderivedTargetIfTrue`, 400
 - uses `$compTimeSum`, 400
 - uses `$envHashTable`, 400
 - uses `$forceAdd`, 400
 - uses `$killOptimizeIfTrue`, 400
 - uses `$packagesUsed`, 400
 - uses `$resolveTimeSum`, 400
 - defun, 400
- `compTuple2Record`
 - calledby `compAdd`, 205
- `compTypeOf`, 408
 - calledby `comp3`, 406
 - calls `comp3`, 408
 - calls `eqsubstlist`, 408
 - calls `get`, 408
 - calls `put`, 408
 - uses `$FormalMapVariableList`, 408
 - uses `$insideCompTypeOf`, 408
 - defun, 408
- `compUniquely`
 - calledby `compForm2`, 416
- `computeAncestorsOf`
 - calledby `compDefineCategory2`, 149
 - calledby `compDefineFunctor1`, 167
- `compVector`, 254
 - calledby `compAtom`, 409
 - calledby `compConstruct`, 220
 - calls `comp`, 254
 - uses `$EmptyVector`, 254
 - defun, 254
- `compWhere`, 255
 - calls `addContour`, 255
 - calls `comp`, 255
 - calls `deltaContour`, 255
 - calls `macroExpand`, 255
 - uses `$EmptyMode`, 255
 - uses `$insideExpressionIfTrue`, 255
 - uses `$insideWhereIfTrue`, 255
 - defun, 255
- `compWithMappingMode`, 419
 - calledby `comp3`, 406
 - calls `compWithMappingMode1`, 419
 - uses `$formalArgList`, 419
 - defun, 419
- `compWithMappingMode1`, 419
 - calledby `compWithMappingMode`, 419
 - calls `comp-tran`, 422
 - calls `compLambda`, 422
 - calls `compMakeDeclaration`, 422
 - calls `comp`, 422
 - calls `extendsCategoryForm`, 422

- calls extractCodeAndConstructTriple, 422
- calls freelist, 422
- calls get, 422
- calls hasFormalMapVariable, 422
- calls isFunctor, 422
- calls optimizeFunctionDef, 422
- calls qcar, 422
- calls qcdr, 422
- calls stackAndThrow, 422
- calls take, 422
- uses \$CategoryFrame, 422
- uses \$EmptyMode, 422
- uses \$FormalMapVariableList, 422
- uses \$QuickCode, 422
- uses \$formalArgList, 422
- uses \$formatArgList, 422
- uses \$funnameTail, 422
- uses \$funname, 422
- uses \$killOptimizeIfTrue, 422
- defun, 419
- concat
 - calledby checkWarning, 367
 - calledby compDefWhereClause, 184
- cons, 218
 - defplist, 218
- consProplistOf
 - calledby setqSingle, 247
- construct, 103, 220, 275
 - defplist, 103, 220, 275
- constructor?
 - calledby addDomain, 187
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
 - calledby isFunctor, 188
- contained
 - calledby evalAndSub, 201
 - calledby parseCategory, 107
- containsBang, 296
 - calledby aplTran, 293
 - calledby containsBang, 296
 - calls containsBang, 296
 - defun, 296
- convert, 411
 - calledby compAtom, 409
 - calledby compCons1, 219
 - calledby compConstruct, 220
 - calledby compElt, 225
 - calledby compJoin, 231
 - calledby setqSingle, 247
 - calls coerce, 411
 - calls resolve, 411
 - defun, 411
- convertOpAlist2compilerInfo, 120
 - calledby updateCategoryFrameForConstructor, 120
 - defun, 120
- convertSpadToAsFile
 - calledby compileSpad2Cmd, 382
- copy
 - calledby modifyModeStack, 429
- copy-token
 - calledby PARSE-TokTail, 322
 - calledby advance-token, 354
- croak
 - calledby drop, 371
- curoutstream
 - usedby s-process, 397
- current-char, 355
 - calledby Advance-Char, 93
 - calledby PARSE-FloatBasePart, 329
 - calledby PARSE-FloatBase, 328
 - calledby PARSE-FloatExponent, 329
 - calledby PARSE-Selector, 325
 - calledby PARSE-TokTail, 322
 - calledby match-string, 346
 - calledby skip-blanks, 346
 - uses \$line, 355
 - uses current-line, 355
 - defun, 355
- current-fragment, 88
 - defvar, 88
- current-line, 90
 - usedby PARSE-Category, 316
 - usedby current-char, 355
 - usedby next-char, 356
 - defvar, 90
- current-symbol, 352
 - calledby PARSE-AnyId, 336
 - calledby PARSE-ElseClause, 343
 - calledby PARSE-FloatBase, 328
 - calledby PARSE-FloatExponent, 329
 - calledby PARSE-Infix, 321

- calledby PARSE-NewExpr, 309
- calledby PARSE-OpenBrace, 338
- calledby PARSE-OpenBracket, 338
- calledby PARSE-Operation, 319
- calledby PARSE-Prefix, 320
- calledby PARSE-Primary1, 326
- calledby PARSE-ReductionOp, 323
- calledby PARSE-Selector, 325
- calledby PARSE-SpecialCommand, 311
- calledby PARSE-SpecialKeyWord, 310
- calledby PARSE-Suffix, 340
- calledby PARSE-TokTail, 322
- calledby PARSE-TokenList, 312
- calledby isTokenDelimiter, 349
- calls current-token, 352
- calls make-symbol-of, 352
- defun, 352
- current-token, 97, 353
 - calledby PARSE-FloatBasePart, 329
 - calledby PARSE-SpecialKeyWord, 310
 - calledby advance-token, 354
 - calledby current-symbol, 352
 - calledby match-advance-string, 347
 - calledby match-current-token, 351
 - calledby next-token, 354
 - calls try-get-token, 353
 - usedby advance-token, 354
 - usedby current-token, 353
 - uses \$token, 97
 - uses current-token, 353
 - uses valid-tokens, 353
 - defun, 353
 - defvar, 97
- curstrm
 - calledby s-process, 396
- dcq
 - calledby preparseReadLine, 85
- decodeScripts, 297
 - calledby decodeScripts, 297
 - calledby getScriptName, 297
 - calls decodeScripts, 297
 - calls qcar, 297
 - calls qcdr, 297
 - calls strconc, 297
 - defun, 297
- deepestExpression, 296
 - calledby deepestExpression, 296
 - calledby hasAplExtension, 295
 - calls deepestExpression, 296
 - defun, 296
- def, 109, 222
 - defplist, 109, 222
- def-process
 - calledby s-process, 396
- def-rename, 399
 - calledby s-process, 396
 - calls def-rename1, 399
 - defun, 399
- def-rename1, 399
 - calledby def-rename1, 399
 - calledby def-rename, 399
 - calls def-rename1, 399
 - defun, 399
- definition-name, 309
 - usedby PARSE-NewExpr, 309
 - defvar, 309
- defmacro
 - Bang, 358
 - line-clear, 90
 - must, 358
 - nth-stack, 370
 - pop-stack-1, 368
 - pop-stack-2, 369
 - pop-stack-3, 369
 - pop-stack-4, 369
 - reduce-stack-clear, 360
 - stack-/empty, 95
 - star, 359
- defplist, 105, 204, 207, 253, 266
 - + - >, 233
 - >, 282
 - <=, 129
 - =>, 283
 - =>, 279
 - >, 116
 - >=, 114, 115
 - „, 274
 - /, 289
 - ., 108, 215, 272
 - ::, 107, 213, 273
 - :BF:, 267

- ;;, 287
- ==, 276
- add, 264
- and, 105
- Block, 268
- capsule, 207
- case, 209
- category, 106, 212, 269
- collect, 240, 271
- cons, 218
- construct, 103, 220, 275
- def, 109, 222
- dollargreaterequal, 112
- dollargreaterthan, 112
- dollarnotequal, 113
- elt, 225
- eqv, 114
- exit, 227
- has, 116, 228
- if, 121, 228, 279
- implies, 124
- import, 229
- In, 281
- in, 125, 280
- inby, 126
- is, 127, 230
- isnt, 127
- Join, 128, 231, 281
- leave, 129, 234
- let, 130, 245
- letd, 131
- ListCategory, 221
- Mapping, 211
- mdef, 131, 235
- not, 132
- notequal, 133
- or, 133
- pretend, 134, 236, 284
- quote, 237, 285
- Record, 211
- RecordCategory, 221
- reduce, 237, 285
- repeat, 240, 286
- return, 135, 242
- Scripts, 286
- segment, 135, 136
- seq, 243
- setq, 245
- Signature, 288
- String, 250
- SubDomain, 250
- SubsetCategory, 252
- TupleCollect, 290
- Union, 211
- UnionCategory, 222
- vcons, 137
- vector, 254
- VectorCategory, 222
- where, 137, 255, 291
- with, 292
- defstruct
 - line, 89
 - reduction, 98
 - stack, 94
 - token, 96
- defun
 - /RQ,LIB, 385
 - /rf-1, 386
 - action, 359
 - add-parens-and-semis-to-line, 84
 - addclose, 370
 - addConstructorModemaps, 192
 - addDomain, 186
 - addEltModemap, 197
 - addEmptyCapsuleIfNecessary, 142
 - addModemap0, 196
 - addModemap1, 198
 - addModemapKnown, 196
 - addNewDomain, 190
 - Advance-Char, 93
 - advance-token, 354
 - aplTran, 293
 - aplTran1, 293
 - aplTranList, 295
 - argsToSig, 428
 - augLisplibModemapsFromCategory, 152
 - augmentLisplibModemapsFromFunctor, 174
 - augModemapsFromCategory, 195
 - augModemapsFromCategoryRep, 203
 - augModemapsFromDomain, 190
 - augModemapsFromDomain1, 191
 - blankp, 371

- bumperrorcount, 363
- char-eq, 356
- char-ne, 356
- checkWarning, 367
- comma2Tuple, 274
- comp, 403
- comp2, 405
- comp3, 405
- compAdd, 205
- compArgumentsAndTryAgain, 418
- compAtom, 409
- compAtSign, 207
- compCapsule, 208
- compCapsuleInner, 208
- compCase, 209
- compCase1, 210
- compCat, 211
- compCategory, 212
- compCoerce, 213
- compCoerce1, 214
- compColon, 215
- compColonInside, 408
- compCons, 219
- compCons1, 219
- compConstruct, 220
- compConstructorCategory, 222
- compDefine, 223
- compDefine1, 223
- compDefineAddSignature, 141
- compDefineCategory, 155
- compDefineCategory1, 145
- compDefineCategory2, 149
- compDefineFunctor, 166
- compDefineFunctor1, 167
- compDefineLisplib, 155
- compDefWhereClause, 184
- compElt, 225
- compExit, 227
- compExpression, 413
- compForm, 414
- compForm1, 414
- compForm2, 416
- compFunctorBody, 176
- compHas, 228
- compIf, 228
- compile-lib-file, 433
- compileDocumentation, 158
- compileFileQuietly, 434
- compiler, 378
- compilerDoit, 384
- compileSpad2Cmd, 380
- compileSpadLispCmd, 431
- compImport, 230
- compIs, 230
- compJoin, 231
- compLambda, 233
- compLeave, 234
- compList, 413
- compMacro, 235
- compMakeDeclaration, 429
- compNoStacking, 404
- compNoStacking1, 404
- compOrCroak, 401
- compOrCroak1, 402
- compOrCroak1.compactify, 431
- compPretend, 236
- compQuote, 237
- compReduce, 238
- compReduce1, 238
- compRepeatOrCollect, 240
- compReturn, 242
- compSeq, 244
- compSeq1, 244
- compSeqItem, 245
- compSetq, 245
- compSetq1, 246
- compString, 250
- compSubDomain, 250
- compSubDomain1, 251
- compSubsetCategory, 252
- compSuchthat, 253
- compSymbol, 411
- compTopLevel, 400
- compTypeOf, 408
- compVector, 254
- compWhere, 255
- compWithMappingMode, 419
- compWithMappingMode1, 419
- containsBang, 296
- convert, 411
- convertOpAlist2compilerInfo, 120
- current-char, 355

- current-symbol, 352
- current-token, 353
- decodeScripts, 297
- deepestExpression, 296
- def-rename, 399
- def-rename1, 399
- disallowNilAttribute, 176
- displayMissingFunctions, 177
- displayPreCompilationErrors, 362
- dollarTran, 357
- domainMember, 195
- drop, 371
- errhuh, 301
- escape-keywords, 349
- escaped, 371
- evalAndRwriteLispForm, 154
- evalAndSub, 201
- extractCodeAndConstructTriple, 427
- finalizeLisplib, 160
- fincomblock, 372
- floatexpid, 357
- freelist, 430
- genDomainOps, 182
- genDomainView, 181
- genDomainViewList, 181
- genDomainViewList0, 180
- get-a-line, 94
- get-token, 355
- getCategoryOpsAndAtts, 162
- getConstructorOpsAndAtts, 162
- getDomainsInScope, 188
- getFunctorOpsAndAtts, 165
- getModemap, 193
- getModemapList, 194
- getModemapListFromDomain, 195
- getOperationAlist, 201
- getScriptName, 297
- getSlotFromCategoryForm, 163
- getSlotFromFunctor, 165
- getTargetFromRhs, 142
- getToken, 350
- getUniqueModemap, 194
- getUniqueSignature, 193
- giveFormalParametersValues, 143
- hackforis, 300
- hackforis1, 301
- hasAplExtension, 295
- hasFormalMapVariable, 427
- hasFullSignature, 141
- indent-pos, 372
- infixtok, 373
- initial-substring, 93
- initial-substring-p, 348
- initialize-prepare, 73
- initializeLisplib, 159
- is-console, 373
- isDomainConstructorForm, 249
- isDomainForm, 248
- isFunctor, 188
- isListConstructor, 111
- isSuperDomain, 189
- isTokenDelimiter, 349
- killColons, 289
- line-advance-char, 91
- line-at-end-p, 90
- line-current-segment, 92
- line-new-line, 92
- line-next-char, 91
- line-past-end-p, 91
- line-print, 90
- lisplibDoRename, 158
- lisplibWrite, 166
- loadIfNecessary, 119
- loadLibIfNecessary, 119
- macroExpand, 144
- macroExpandInPlace, 143
- macroExpandList, 144
- make-string-adjustable, 94
- make-symbol-of, 352
- makeCategoryPredicates, 146
- makeFunctorArgumentParameters, 178
- makeSimplePredicateOrNil, 364
- match-advance-string, 347
- match-current-token, 351
- match-next-token, 352
- match-string, 346
- match-token, 351
- mergeModemap, 199
- mergeSignatureAndLocalVarAlists, 165
- meta-syntax-error, 357
- mkCategoryPackage, 146
- mkConstructor, 154

- mkEvalableCategoryForm, 148
- mkNewModemapList, 198
- mkOpVec, 183
- modifyModeStack, 429
- ncINTERPFILE, 431
- new2OldLisp, 364
- next-char, 355
- next-line, 92
- next-tab-loc, 373
- next-token, 354
- nonblankloc, 374
- optional, 359
- PARSE-AnyId, 336
- PARSE-Application, 324
- parse-argument-designator, 367
- PARSE-Category, 315
- PARSE-Command, 310
- PARSE-CommandTail, 313
- PARSE-Conditional, 343
- PARSE-Data, 334
- PARSE-ElseClause, 343
- PARSE-Enclosure, 330
- PARSE-Exit, 341
- PARSE-Expr, 318
- PARSE-Expression, 317
- PARSE-Float, 327
- PARSE-FloatBase, 328
- PARSE-FloatBasePart, 328
- PARSE-FloatExponent, 329
- PARSE-FloatTok, 345
- PARSE-Form, 323
- PARSE-FormalParameter, 331
- PARSE-FormalParameterTok, 331
- PARSE-getSemanticForm, 320
- PARSE-GlyphTok, 336
- parse-identifier, 365
- PARSE-Import, 317
- PARSE-Infix, 321
- PARSE-InfixWith, 315
- PARSE-IntegerTok, 330
- PARSE-Iterator, 339
- PARSE-IteratorTail, 339
- parse-keyword, 366
- PARSE-Label, 325
- PARSE-LabelExpr, 344
- PARSE-Leave, 342
- PARSE-LedPart, 318
- PARSE-leftBindingPowerOf, 319
- PARSE-Loop, 344
- PARSE-Name, 333
- PARSE-NBGlyphTok, 335
- PARSE-NewExpr, 309
- PARSE-NudPart, 318
- parse-number, 366
- PARSE-OpenBrace, 338
- PARSE-OpenBracket, 338
- PARSE-Operation, 319
- PARSE-Option, 314
- PARSE-Prefix, 320
- PARSE-Primary, 326
- PARSE-Primary1, 326
- PARSE-PrimaryNoFloat, 326
- PARSE-PrimaryOrQM, 313
- PARSE-Quad, 331
- PARSE-Qualification, 322
- PARSE-Reduction, 323
- PARSE-ReductionOp, 323
- PARSE-Return, 341
- PARSE-rightBindingPowerOf, 320
- PARSE-ScriptItem, 333
- PARSE-Scripts, 332
- PARSE-Seg, 342
- PARSE-Selector, 325
- PARSE-SemiColon, 341
- PARSE-Sequence, 337
- PARSE-Sequence1, 337
- PARSE-Sexpr, 334
- PARSE-Sexpr1, 334
- parse-spadstring, 364
- PARSE-SpecialCommand, 311
- PARSE-SpecialKeyword, 310
- PARSE-Statement, 314
- PARSE-String, 331
- parse-string, 365
- PARSE-Suffix, 340
- PARSE-TokenCommandTail, 311
- PARSE-TokenList, 312
- PARSE-TokenOption, 312
- PARSE-TokTail, 322
- PARSE-VarForm, 332
- PARSE-With, 315
- parseAnd, 105

parseAtom, 102
parseAtSign, 106
parseCategory, 107
parseCoerce, 108
parseColon, 108
parseConstruct, 103
parseDEF, 109
parseDollarGreaterEqual, 112
parseDollarGreaterThan, 112
parseDollarLessEqual, 113
parseDollarNotEqual, 113
parseDropAssertions, 107
parseEquivalence, 114
parseExit, 115
parseGreaterEqual, 115
parseGreaterThan, 116
parseHas, 116
parseHasRhs, 118
parseIf, 122
parseIf,ifTran, 122
parseImplies, 124
parseIn, 125
parseInBy, 126
parseIs, 127
parseIsnt, 128
parseJoin, 128
parseLeave, 129
parseLessEqual, 130
parseLET, 130
parseLETD, 131
parseLhs, 110
parseMDEF, 131
parseNot, 132
parseNotEqual, 133
parseOr, 133
parsepiles, 84
parsePretend, 134
parseprint, 374
parseReturn, 135
parseSegment, 136
parseSeq, 136
parseTran, 101
parseTranCheckForRecord, 363
parseTranList, 103
parseTransform, 101
parseType, 106
parseVCONS, 137
parseWhere, 137
Pop-Reduction, 370
postAdd, 264
postAtom, 259
postAtSign, 267
postBigFloat, 268
postBlock, 268
postBlockItem, 266
postBlockItemList, 265
postCapsule, 265
postCategory, 269
postcheck, 261
postCollect, 271
postCollect,finish, 270
postColon, 273
postColonColon, 273
postComma, 274
postConstruct, 275
postDef, 276
postDefArgs, 278
postError, 262
postExit, 279
postFlatten, 274
postFlattenLeft, 287
postForm, 262
postIf, 279
postIn, 281
postin, 280
postInSeq, 280
postIteratorList, 272
postJoin, 282
postMakeCons, 270
postMapping, 282
postMDef, 283
postOp, 259
postPretend, 284
postQUOTE, 285
postReduce, 285
postRepeat, 286
postScripts, 287
postScriptsForm, 260
postSemiColon, 287
postSignature, 288
postSlash, 289
postTran, 258

- postTranList, 260
- postTranScripts, 260
- postTranSegment, 276
- postTransform, 257
- postTransformCheck, 261
- postTuple, 290
- postTupleCollect, 291
- postType, 267
- postWhere, 291
- postWith, 292
- preparse, 76
- preparse-echo, 88
- preparse1, 81
- preparseReadLine, 85
- preparseReadLine1, 87
- primitiveType, 411
- print-defun, 399
- print-package, 367
- push-reduction, 360
- putDomainsInScope, 189
- quote-if-string, 348
- read-a-line, 88
- recompile-lib-file-if-necessary, 433
- removeSuperfluousMapping, 289
- reportOnFunctorCompilation, 177
- rwriteLispForm, 154
- s-process, 396
- setDefOp, 292
- setqSetelt, 246
- setqSingle, 247
- skip-blanks, 346
- skip-ifblock, 86
- skip-to-endif, 374
- spad, 395
- spad-fixed-arg, 433
- stack-clear, 95
- stack-load, 95
- stack-pop, 96
- stack-push, 96
- storeblanks, 93
- substituteCategoryArguments, 192
- substNames, 202
- token-install, 98
- token-lookahead-type, 347
- token-print, 98
- transformOperationAlist, 163
- transIs, 110
- transIs1, 110
- translabel, 361
- translabel1, 361
- try-get-token, 353
- tuple2List, 368
- underscore, 350
- unget-tokens, 350
- unTuple, 301
- updateCategoryFrameForCategory, 121
- updateCategoryFrameForConstructor, 120
- writeLib1, 160
- defvar
 - \$FormalMapVariableList, 202
 - \$byConstructors, 434
 - \$comblocklist, 371
 - \$constructorsSeen, 434
 - \$defstack, 299
 - \$echolinestack, 72
 - \$index, 72
 - \$is-eqlist, 300
 - \$is-gensymlist, 300
 - \$is-spill-list, 299
 - \$is-spill, 299
 - \$linelist, 72
 - \$preparse-last-line, 72
 - \$v1, 300
 - current-fragment, 88
 - current-line, 90
 - current-token, 97
 - definition-name, 309
 - initial-gensym, 300
 - lablasoc, 309
 - meta-error-handler, 356
 - next-token, 97
 - nonblank, 97
 - ParseMode, 309
 - prior-token, 97
 - reduce-stack, 360
 - tmptok, 308
 - tok, 308
 - valid-tokens, 98
 - XTokenReader, 355
- delete
 - calledby compDefWhereClause, 184
 - calledby putDomainsInScope, 189

- deltaContour
 - calledby compWhere, 255
- digitp[5]
 - called by PARSE-FloatBasePart, 329
 - called by PARSE-FloatBase, 328
 - called by floatexpid, 357
- disallowNilAttribute, 176
 - calledby compDefineFunctor1, 167
 - defun, 176
- displayComp
 - calledby compOrCroak1, 402
- displayMissingFunctions, 177
 - calledby reportOnFunctorCompilation, 177
 - calls bright, 177
 - calls formatUnabbreviatedSig, 177
 - calls getmode, 177
 - calls member, 177
 - calls sayBrightly, 177
 - uses \$CheckVectorList, 178
 - uses \$env, 178
 - uses \$formalArgList, 178
 - defun, 177
- displayPreCompilationErrors, 362
 - calledby s-process, 396
 - calls length, 362
 - calls nequal, 362
 - calls remdup, 362
 - calls sayBrightly, 362
 - calls sayMath, 362
 - uses \$postStack, 362
 - uses \$stopOp, 362
 - defun, 362
- displaySemanticErrors
 - calledby compOrCroak1, 402
 - calledby reportOnFunctorCompilation, 177
 - calledby s-process, 396
- displayWarnings
 - calledby reportOnFunctorCompilation, 177
- dollargreaterequal, 112
 - defplist, 112
- dollargreaterthan, 112
 - defplist, 112
- dollarnotequal, 113
 - defplist, 113
- dollarTran, 357
 - calledby PARSE-Qualification, 322
- uses \$InteractiveMode, 357
 - defun, 357
- domainMember, 195
 - calledby addDomain, 187
 - calls modeEqual, 195
 - defun, 195
- doSystemCommand[5]
 - called by preparse1, 81
- drop, 371
 - calledby add-parens-and-semis-to-line, 84
 - calledby drop, 371
 - calls croak, 371
 - calls drop, 371
 - calls take, 371
 - defun, 371
- Echo-Meta
 - usedby preparse-echo, 88
- echo-meta
 - usedby /rf-1, 386
 - usedby spad, 395
- echo-meta[5]
 - called by /RQ,LIB, 385
- editfile
 - usedby compCapsule, 208
- elemn
 - calledby PARSE-Operation, 319
 - calledby PARSE-leftBindingPowerOf, 320
 - calledby PARSE-rightBindingPowerOf, 320
- elt, 225
 - defplist, 225
- eqcar
 - calledby PARSE-OpenBrace, 338
 - calledby PARSE-OpenBracket, 338
 - calledby getToken, 350
 - calledby hackforis1, 301
- eqsubstlist
 - calledby compColon, 216
 - calledby compTypeOf, 408
 - calledby isDomainConstructorForm, 249
 - calledby substNames, 202
- eqv, 114
 - defplist, 114
- erase
 - calledby initializeLisplib, 159
- errhuh, 301

- calls systemError, 301
- defun, 301
- error
 - calledby compileSpad2Cmd, 382
- errorRef
 - calledby compSymbol, 412
- errors
 - usedby initializeLisplib, 159
- Escape-Character
 - usedby token-lookahead-type, 347
- escape-keywords, 349
 - calledby quote-if-string, 348
- defun, 349
- escaped, 371
 - calledby preparse1, 81
- defun, 371
- eval
 - calledby compDefineCategory2, 149
 - calledby evalAndRwriteLispForm, 154
 - calledby getSlotFromCategoryForm, 163
- evalAndRwriteLispForm, 154
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
 - calledby compSubDomain1, 251
 - calls eval, 154
 - calls rwriteLispForm, 154
- defun, 154
- evalAndSub, 201
 - calledby augModemapsFromCategoryRep, 203
 - calledby augModemapsFromCategory, 195
 - calls contained, 201
 - calls getOperationAlist, 201
 - calls get, 201
 - calls isCategory, 201
 - calls put, 201
 - calls substNames, 201
 - local def \$lhsOfColon, 201
- defun, 201
- exit, 227
 - defplist, 227
- expand-tabs
 - calledby preparseReadLine1, 87
- extendLocalLibdb
 - calledby compileSpad2Cmd, 382
- extendsCategoryForm
 - calledby compWithMappingMode1, 422
- extractCodeAndConstructTriple, 427
 - calledby compWithMappingMode1, 422
- defun, 427
- file-closed
 - usedby spad, 395
- filep
 - calledby compDefineLisplib, 155
- fillerSpaces
 - calledby compDefineLisplib, 155
- finalizeDocumentation
 - calledby compileDocumentation, 158
 - calledby finalizeLisplib, 160
- finalizeLisplib, 160
 - calledby compDefineLisplib, 155
 - calls NRTgenInitialAttributeAlist, 160
 - calls finalizeDocumentation, 160
 - calls getConstructorOpsAndAtts, 160
 - calls lisplibWrite, 160
 - calls makeprop, 160
 - calls mergeSignatureAndLocalVarAlists, 160
 - calls namestring, 160
 - calls profileWrite, 160
 - calls removeZeroOne, 160
 - calls sayMSG, 160
 - local def \$NRTslot1PredicateList, 161
 - local def \$lisplibCategory, 161
 - local def \$pairlis, 161
 - local ref \$/editfile, 160
 - local ref \$FormalMapVariableList, 160
 - local ref \$libFile, 160
 - local ref \$lisplibAbbreviation, 161
 - local ref \$lisplibAncestors, 161
 - local ref \$lisplibAttributes, 161
 - local ref \$lisplibCategory, 160
 - local ref \$lisplibForm, 160
 - local ref \$lisplibKind, 160
 - local ref \$lisplibModemapAlist, 160
 - local ref \$lisplibModemap, 160
 - local ref \$lisplibParents, 161
 - local ref \$lisplibPredicates, 161
 - local ref \$lisplibSignatureAlist, 160
 - local ref \$lisplibSlot1, 161
 - local ref \$lisplibSuperDomain, 160

- local ref \$lisplibVariableAlist, 160
 - local ref \$profileCompiler, 161
 - local ref \$spadLibFT, 161
 - defun, 160
- fincomblock, 372
 - calledby prepare1, 81
 - calls prepare-echo, 372
 - uses \$EchoLineStack, 372
 - uses \$comblocklist, 372
 - defun, 372
- findfile
 - calledby compiler, 379
- floatexpid, 357
 - calledby PARSE-FloatExponent, 329
 - calls collect, 357
 - calls digitp[5], 357
 - calls identp[5], 357
 - calls maxindex, 357
 - calls pname[5], 357
 - calls spadreduce, 357
 - calls step, 357
 - defun, 357
- fnameMake[5]
 - called by compileSpadLispCmd, 431
- fnameReadable?[5]
 - called by compileSpadLispCmd, 432
- formal2Pattern
 - calledby augmentLisplibModemapsFrom-
Functor, 174
- formatUnabbreviated
 - calledby compMacro, 235
- formatUnabbreviatedSig
 - calledby displayMissingFunctions, 177
- fp-output-stream
 - calledby is-console, 373
- freelist, 430
 - calledby compWithMappingMode1, 422
 - calledby freelist, 430
 - calls assq[5], 430
 - calls freelist, 430
 - calls getmode, 430
 - calls identp[5], 430
 - calls unionq, 430
 - defun, 430
- functionp
 - calledby loadLibIfNecessary, 119
- genDomainOps, 182
 - calledby genDomainView, 181
 - calls addModemap, 182
 - calls getOperationAlist, 182
 - calls mkDomainConstructor, 182
 - calls mkq, 182
 - calls substNames, 182
 - uses \$ConditionalOperators, 182
 - uses \$e, 182
 - uses \$getDomainCode, 182
 - defun, 182
- genDomainView, 181
 - calledby genDomainViewList, 181
 - calls augModemapsFromCategory, 181
 - calls genDomainOps, 181
 - calls member, 181
 - calls mkDomainConstructor, 181
 - calls pairp, 181
 - calls qcar, 181
 - calls qcdr, 181
 - uses \$e, 181
 - uses \$getDomainCode, 181
 - defun, 181
- genDomainViewList, 181
 - calledby genDomainViewList, 181
 - calls genDomainViewList, 181
 - calls genDomainView, 181
 - calls isCategoryForm, 181
 - calls pairp, 181
 - calls qcdr, 181
 - uses \$EmptyEnvironment, 181
 - defun, 181
- genDomainViewList0, 180
 - calledby makeFunctorArgumentParamet-
ters, 178
 - calls getDomainViewList, 180
 - defun, 180
- genSomeVariable
 - calledby compColon, 216
- genvar
 - calledby hasAplExtension, 295
- get
 - calledby compAtom, 409
 - calledby compNoStacking1, 404
 - calledby compSymbol, 412
 - calledby compTypeOf, 408

- calledby compWithMappingModel, 422
- calledby evalAndSub, 201
- calledby getDomainsInScope, 189
- calledby getModemapListFromDomain, 195
- calledby getModemapList, 194
- calledby getModemap, 193
- calledby giveFormalParametersValues, 143
- calledby hasFullSignature, 141
- calledby isFunctor, 188
- calledby isSuperDomain, 190
- calledby mkEvalableCategoryForm, 148
- calledby parseHasRhs, 118
- calledby setqSingle, 247
- get-a-line, 94
 - calledby initialize-prepare, 73
 - calledby prepareReadLine1, 87
 - calls is-console, 94
 - calls make-string-adjustable, 94
 - calls mkprompt[5], 94
 - calls read-a-line, 94
 - defun, 94
- get-internal-run-time
 - calledby s-process, 396
- get-token, 355
 - calledby try-get-token, 353
 - calls XTokenReader, 355
 - uses XTokenReader, 355
 - defun, 355
- getAbbreviation
 - calledby compDefine1, 223
- getArgumentModeOrMoan
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
- getCategoryOpsAndAtts, 162
 - calledby getConstructorOpsAndAtts, 162
 - calls getSlotFromCategoryForm, 162
 - calls transformOperationAlist, 162
 - defun, 162
- getConstructorAbbreviation
 - calledby compDefineLisplib, 155
- getConstructorOpsAndAtts, 162
 - calledby finalizeLisplib, 160
 - calls getCategoryOpsAndAtts, 162
 - calls getFunctorOpsAndAtts, 162
 - defun, 162
- getdatabase
 - calledby augModemapsFromDomain, 190
 - calledby compDefineFunctor1, 167
 - calledby compDefineLisplib, 155
 - calledby getOperationAlist, 201
 - calledby isFunctor, 188
 - calledby loadLibIfNecessary, 119
 - calledby macroExpandList, 144
 - calledby mkCategoryPackage, 146
 - calledby mkEvalableCategoryForm, 148
 - calledby parseHas, 116
 - calledby updateCategoryFrameForCategory, 121
 - calledby updateCategoryFrameForConstructor, 120
- getDeltaEntry
 - calledby compElt, 225
- getDomainsInScope, 188
 - calledby addDomain, 186
 - calledby augModemapsFromDomain, 190
 - calledby comp3, 406
 - calledby compColon, 215
 - calledby compConstruct, 220
 - calledby putDomainsInScope, 189
 - calls get, 189
 - local ref \$CapsuleDomainsInScope, 189
 - local ref \$insideCapsuleFunctionIfTrue, 189
 - defun, 188
- getDomainViewList
 - calledby genDomainViewList0, 180
- getFormModemaps
 - calledby compForm1, 414
- getFunctorOpsAndAtts, 165
 - calledby getConstructorOpsAndAtts, 162
 - calls getSlotFromFunctor, 165
 - calls transformOperationAlist, 165
 - defun, 165
- getIdentity
 - calledby compReduce1, 238
- getl
 - calledby PARSE-Operation, 319
 - calledby PARSE-ReductionOp, 323
 - calledby PARSE-leftBindingPowerOf, 319
 - calledby PARSE-rightBindingPowerOf, 320
 - calledby addConstructorModemaps, 192
 - calledby augModemapsFromDomain1, 191

- calledby compCat, 211
- calledby compExpression, 413
- calledby loadLibIfNecessary, 119
- calledby parseTran, 101
- getmode
 - calledby addDomain, 187
 - calledby augModemapsFromDomain1, 191
 - calledby comp3, 406
 - calledby compCoerce, 213
 - calledby compColon, 216
 - calledby compDefWhereClause, 184
 - calledby compSymbol, 411
 - calledby displayMissingFunctions, 177
 - calledby freelist, 430
 - calledby setqSingle, 247
- getModemap, 193
 - calledby compDefineFunctor1, 167
 - calls compApplyModemap, 193
 - calls get, 193
 - calls sublis, 193
 - defun, 193
- getModemapList, 194
 - calledby compCase1, 210
 - calledby getUniqueModemap, 194
 - calls getModemapListFromDomain, 194
 - calls get, 194
 - calls nreverse0, 194
 - calls pairp, 194
 - calls qcar, 194
 - calls qcdr, 194
 - defun, 194
- getModemapListFromDomain, 195
 - calledby compElt, 225
 - calledby getModemapList, 194
 - calls get, 195
 - defun, 195
- getmodeOrMapping
 - calledby augModemapsFromDomain1, 191
- getOperationAlist, 201
 - calledby evalAndSub, 201
 - calledby genDomainOps, 182
 - calls compMakeCategoryObject, 201
 - calls getdatabase, 201
 - calls isFunctor, 201
 - calls stackMessage, 201
 - calls systemError, 201
 - uses \$domainShell, 201
 - uses \$e, 201
 - uses \$functorForm, 201
 - uses \$insideFunctorIfTrue, 201
 - defun, 201
- getOperationAlistFromLisplib
 - calledby mkOpVec, 183
- getParentsFor
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
- getPrincipalView
 - calledby mkOpVec, 183
- getProplist
 - calledby addModemap1, 198
 - calledby compDefineAddSignature, 141
 - calledby loadLibIfNecessary, 119
- getProplist[5]
 - called by setqSingle, 247
- getScriptName, 297
 - calledby postScriptsForm, 260
 - calledby postScripts, 287
 - calls decodeScripts, 297
 - calls identp[5], 297
 - calls internl, 297
 - calls pname[5], 297
 - calls postError, 297
 - defun, 297
- getSignatureFromMode
 - calledby compDefine1, 223
- getSlotFromCategoryForm, 163
 - calledby getCategoryOpsAndAtts, 162
 - calls eval, 163
 - calls systemErrorHere, 163
 - calls take, 163
 - local ref \$FormalMapVariableList, 163
 - defun, 163
- getSlotFromFunctor, 165
 - calledby getFunctorOpsAndAtts, 165
 - calls compMakeCategoryObject, 165
 - calls systemErrorHere, 165
 - local ref \$e, 165
 - local ref \$lisplibOperationAlist, 165
 - defun, 165
- getTargetFromRhs, 142
 - calledby compDefine1, 223
 - calledby getTargetFromRhs, 142

- calls compOrCroak, 142
- calls getTargetFromRhs, 142
- calls stackSemanticError, 142
- defun, 142
- getToken, 350
 - calledby PARSE-OpenBrace, 338
 - calledby PARSE-OpenBracket, 338
 - calls eqcar, 350
 - defun, 350
- getUniqueModemap, 194
 - calledby getUniqueSignature, 193
 - calls getModemapList, 194
 - calls qslessp, 194
 - calls stackWarning, 194
 - defun, 194
- getUniqueSignature, 193
 - calls getUniqueModemap, 193
 - defun, 193
- giveFormalParametersValues, 143
 - calledby compDefine1, 223
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
 - calls get, 143
 - calls put, 143
 - defun, 143
- hackforis, 300
 - calls hackforis1, 300
 - defun, 300
- hackforis1, 301
 - calledby hackforis, 300
 - calls eqcar, 301
 - calls kar, 301
 - defun, 301
- has, 116, 228
 - defplist, 116, 228
- hasAplExtension, 295
 - calledby aplTran1, 293
 - calls aplTran1, 295
 - calls deepestExpression, 295
 - calls genvar, 295
 - calls msubst, 295
 - calls nreverse0, 295
 - defun, 295
- hasFormalMapVariable, 427
 - calledby compWithMappingMode1, 422
- calls ScanOrPairVec[5], 427
- local def \$formalMapVariables, 427
- defun, 427
- hasFullSignature, 141
 - calledby compDefineAddSignature, 141
 - calls get, 141
 - defun, 141
- helpSpad2Cmd[5]
 - called by compiler, 379
- identp
 - calledby addDomain, 186
 - calledby isFunctor, 188
- identp[5]
 - called by PARSE-FloatExponent, 329
 - called by compSetq1, 246
 - called by floatexpid, 357
 - called by freelist, 430
 - called by getScriptName, 297
 - called by postTransform, 257
 - called by setqSingle, 247
- if, 121, 228, 279
 - defplist, 121, 228, 279
- ifcar
 - calledby preparsed, 76
- implies, 124
 - defplist, 124
- import, 229
 - defplist, 229
- In, 281
 - defplist, 281
- in, 125, 280
 - defplist, 125, 280
- inby, 126
 - defplist, 126
- incExitLevel
 - calledby parseIf,ifTran, 122
- indent-pos, 372
 - calledby preparsed1, 81
 - defun, 372
- infixtok, 373
 - calledby add-parens-and-semis-to-line, 84
 - calls string2id-n, 373
 - defun, 373
- init-boot/spad-reader[5]
 - called by spad, 395

- initial-gensym, 300
 - defvar, 300
- initial-substring, 93
 - calledby prepareReadLine, 85
 - calledby skip-ifblock, 86
 - calledby skip-to-endif, 374
 - calls mismatch, 93
 - defun, 93
- initial-substring-p, 348
 - calledby match-string, 346
 - calls string-not-greaterp, 348
 - defun, 348
- initialize-prepare, 73
 - calledby spad, 395
 - calls get-a-line, 73
 - uses \$echolinestack, 73
 - uses \$index, 73
 - uses \$linelist, 73
 - uses \$prepare-last-line, 73
 - defun, 73
- initializeLisplib, 159
 - calls LAM,FILEACTQ, 159
 - calls adoptions, 159
 - calls erase, 159
 - calls pathnameTypeId, 159
 - calls writeLib1, 159
 - local def \$libFile, 159
 - local def \$lisplibAbbreviation, 159
 - local def \$lisplibAncestors, 159
 - local def \$lisplibForm, 159
 - local def \$lisplibKind, 159
 - local def \$lisplibModemapAlist, 159
 - local def \$lisplibModemap, 159
 - local def \$lisplibOpAlist, 159
 - local def \$lisplibOperationAlist, 159
 - local def \$lisplibSignatureAlist, 159
 - local def \$lisplibSuperDomain, 159
 - local def \$lisplibVariableAlist, 159
 - local ref \$erase, 159
 - local ref \$libFile, 159
 - uses /editfile, 159
 - uses /major-version, 159
 - uses errors, 159
 - defun, 159
- insert
 - calledby comp2, 405
- insertAlist
 - calledby transformOperationAlist, 164
- insertModemap
 - calledby mkNewModemapList, 199
- interactiveModemapForm
 - calledby augLisplibModemapsFromCategory, 153
 - calledby augmentLisplibModemapsFromFunctor, 174
- internl
 - calledby getScriptName, 297
 - calledby postForm, 262
 - calledby substituteCategoryArguments, 192
- intersectionEnvironment
 - calledby compIf, 229
- ioclear
 - calledby spad, 395
- is, 127, 230
 - defplist, 127, 230
- is-console, 373
 - calledby get-a-line, 94
 - calledby prepare1, 81
 - calledby print-defun, 399
 - calls fp-output-stream, 373
 - uses *terminal-io*, 373
 - defun, 373
- isAlmostSimple
 - calledby makeSimplePredicateOrNil, 364
- isCategory
 - calledby augModemapsFromCategoryRep, 203
 - calledby evalAndSub, 201
- isCategoryForm
 - calledby addDomain, 187
 - calledby augLisplibModemapsFromCategory, 152
 - calledby compColon, 216
 - calledby compJoin, 231
 - calledby genDomainViewList, 181
 - calledby isDomainConstructorForm, 249
 - calledby isDomainForm, 249
 - calledby makeFunctorArgumentParameters, 178
- isCategoryPackageName
 - calledby compDefineFunctor1, 167
 - calledby substNames, 202

- isDomainConstructorForm, 249
 - calledby isDomainForm, 249
 - calls eqsubstlist, 249
 - calls isCategoryForm, 249
 - calls pairp, 249
 - calls qcar, 249
 - calls qcdr, 249
 - local ref \$FormalMapVariableList, 249
 - defun, 249
- isDomainForm, 248
 - calledby comp2, 405
 - calledby compColon, 215, 216
 - calledby compDefine1, 223
 - calledby compElt, 225
 - calledby setqSingle, 247
 - calls isCategoryForm, 249
 - calls isDomainConstructorForm, 249
 - calls isFunctor, 249
 - calls kar, 248
 - calls pairp, 248
 - calls qcar, 248
 - calls qcdr, 249
 - local ref \$SpecialDomainNames, 249
 - defun, 248
- isDomainInScope
 - calledby setqSingle, 247
- isFunction
 - calledby compSymbol, 412
- isFunctor, 188
 - calledby addDomain, 187
 - calledby comp2, 405
 - calledby compWithMappingMode1, 422
 - calledby getOperationAlist, 201
 - calledby isDomainForm, 249
 - calls constructor?, 188
 - calls getdatabase, 188
 - calls get, 188
 - calls identp, 188
 - calls opOf, 188
 - calls updateCategoryFrameForCategory, 188
 - calls updateCategoryFrameForConstructor, 188
 - local ref \$CategoryFrame, 188
 - local ref \$InteractiveMode, 188
 - defun, 188
- isListConstructor, 111
 - calledby transIs, 110
 - calls member, 111
 - defun, 111
- isLiteral
 - calledby addDomain, 187
- isMacro
 - calledby compDefine1, 223
- isnt, 127
 - defplist, 127
- isPackageFunction
 - calledby compDefineFunctor1, 167
- isSimple
 - calledby compForm2, 416
 - calledby makeSimplePredicateOrNil, 364
- isSubset
 - calledby isSuperDomain, 189
- isSuperDomain, 189
 - calledby mergeModemap, 199
 - calls get, 190
 - calls isSubset, 189
 - calls lassoc, 190
 - calls opOf, 190
 - defun, 189
- isSymbol
 - calledby compAtom, 409
- isTokenDelimiter, 349
 - calledby PARSE-TokenList, 312
 - calls current-symbol, 349
 - defun, 349
- Join, 128, 231, 281
 - defplist, 128, 231, 281
- JoinInner
 - calledby mkCategoryPackage, 146
- kar
 - calledby addEmptyCapsuleIfNecessary, 142
 - calledby augModemapsFromDomain1, 191
 - calledby augModemapsFromDomain, 190
 - calledby hackforis1, 301
 - calledby isDomainForm, 248
- keyedSystemError
 - calledby transformOperationAlist, 164
- killColons, 289
 - calledby killColons, 289

- calledby postSignature, 288
 - calls killColons, 289
 - defun, 289
- labasoc
 - usedby PARSE-Data, 334
- lablasoc, 309
 - defvar, 309
- LAM,FILEACTQ
 - calledby initializeLisplib, 159
- lassoc
 - calledby addModemap1, 198
 - calledby augLisplibModemapsFromCategory, 153
 - calledby compDefWhereClause, 184
 - calledby compDefineAddSignature, 141
 - calledby compOrCroak1,compactify, 431
 - calledby isSuperDomain, 190
 - calledby loadLibIfNecessary, 119
 - calledby mergeSignatureAndLocalVarAlists, 165
 - calledby translable1, 361
- lassq
 - calledby transformOperationAlist, 164
- last
 - calledby parseSeq, 136
- leave, 129, 234
 - defplist, 129, 234
- length
 - calledby compColon, 216
 - calledby compDefine1, 223
 - calledby compElt, 225
 - calledby compForm1, 414
 - calledby compForm2, 416
 - calledby compRepeatOrCollect, 240
 - calledby displayPreCompilationErrors, 362
 - calledby mkOpVec, 183
 - calledby postScriptsForm, 260
- let, 130, 245
 - defplist, 130, 245
- letd, 131
 - defplist, 131
- line, 89
 - usedby match-string, 346
 - usedby spad, 395
 - defstruct, 89
- Line-Advance-Char
 - calledby Advance-Char, 93
- line-advance-char, 91
 - uses \$line, 91
 - defun, 91
- Line-At-End-P
 - calledby Advance-Char, 93
- line-at-end-p, 90
 - calledby next-char, 356
 - uses \$line, 90
 - defun, 90
- line-clear, 90
 - uses \$line, 90
 - defmacro, 90
- line-current-char
 - calledby match-advance-string, 347
- line-current-index
 - calledby match-advance-string, 347
- line-current-segment, 92
 - calledby unget-tokens, 350
 - defun, 92
- Line-New-Line
 - calledby read-a-line, 88
- line-new-line, 92
 - calledby unget-tokens, 350
 - uses \$line, 92
 - defun, 92
- line-next-char, 91
 - calledby next-char, 356
 - uses \$line, 91
 - defun, 91
- line-number
 - calledby PARSE-Category, 316
 - calledby unget-tokens, 350, 351
- line-past-end-p, 91
 - calledby match-advance-string, 347
 - calledby match-string, 346
 - uses \$line, 91
 - defun, 91
- line-print, 90
 - uses \$line, 90, 92
 - defun, 90
- lispize
 - calledby compSubDomain1, 251
- lisplibDoRename, 158
 - calledby compDefineLisplib, 155

- calls replaceFile, 158
- local ref \$spadLibFT, 158
- defun, 158
- lisplibWrite, 166
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
 - calledby compileDocumentation, 158
 - calledby finalizeLisplib, 160
 - calls rwrite128, 166
 - local ref \$lisplib, 166
 - defun, 166
- ListCategory, 221
 - defplist, 221
- listOfIdentifiersIn
 - calledby compDefWhereClause, 184
- listOfPatternIds
 - calledby augmentLisplibModemapsFrom-Functor, 174
- listOrVectorElementNode
 - calledby augModemapsFromDomain, 190
- loadIfNecessary, 119
 - calledby parseHasRhs, 118
 - calls loadLibIfNecessary, 119
 - defun, 119
- loadLib
 - calledby loadLibIfNecessary, 119
- loadLibIfNecessary, 119
 - calledby loadIfNecessary, 119
 - calledby loadLibIfNecessary, 119
 - calls functionp, 119
 - calls getProplist, 119
 - calls getdatabase, 119
 - calls getl, 119
 - calls lassoc, 119
 - calls loadLibIfNecessary, 119
 - calls loadLib, 119
 - calls macrop, 119
 - calls throwKeyedMsg, 119
 - calls updateCategoryFrameForCategory, 119
 - calls updateCategoryFrameForConstructor, 119
 - local ref \$CategoryFrame, 119
 - local ref \$InteractiveMode, 119
 - defun, 119
- localdatabase
 - calledby compDefineLisplib, 155
- localdatabase[5]
 - called by compileSpadLispCmd, 432
- lt
 - calledby PARSE-Operation, 319
- macroExpand, 144
 - calledby compDefine1, 223
 - calledby compMacro, 235
 - calledby compSeqItem, 245
 - calledby compWhere, 255
 - calledby macroExpandInPlace, 143
 - calledby macroExpandList, 144
 - calledby macroExpand, 144
 - calls macroExpandList, 144
 - calls macroExpand, 144
 - defun, 144
- macroExpandInPlace, 143
 - calls macroExpand, 143
 - defun, 143
- macroExpandList, 144
 - calledby macroExpand, 144
 - calls getdatabase, 144
 - calls macroExpand, 144
 - defun, 144
- macrop
 - calledby loadLibIfNecessary, 119
- make-float
 - calledby PARSE-Float, 327
- make-full-cvec
 - calledby preparsel, 81
- make-input-filename
 - calledby compileDocumentation, 158
- make-reduction
 - calledby push-reduction, 360
- make-string-adjustable, 94
 - calledby get-a-line, 94
 - defun, 94
- make-symbol-of, 352
 - calledby PARSE-Expression, 317
 - calledby current-symbol, 352
 - uses \$token, 352
 - defun, 352
- makeCategoryForm
 - calledby compColon, 216
- makeCategoryPredicates, 146

- calledby compDefineCategory1, 145
 - uses \$FormalMapVariableList, 146
 - uses \$TriangleVariableList, 146
 - uses \$mvl, 146
 - uses \$tvl, 146
 - defun, 146
- makeFunctorArgumentParameters, 178
 - calledby compDefineFunctor1, 167
 - calls assq, 178
 - calls genDomainViewList0, 178
 - calls isCategoryForm, 178
 - calls msubst, 178
 - calls pairp, 178
 - calls qcar, 178
 - calls qcdr, 178
 - calls union, 179
 - uses \$ConditionalOperators, 179
 - uses \$alternateViewList, 179
 - uses \$forceAdd, 179
 - defun, 178
- makeInitialModemapFrame[5]
 - called by spad, 395
- makeInputFilename[5]
 - called by /rf-1, 386
- makeLiteral
 - calledby addEltModemap, 197
- makeNonAtomic
 - calledby parseHas, 117
- makeprop
 - calledby finalizeLisplib, 160
- makeSimplePredicateOrNil, 364
 - calledby parseIf,ifTran, 122
 - calls isAlmostSimple, 364
 - calls isSimple, 364
 - calls wrapSEQExit, 364
 - defun, 364
- mapInto
 - calledby parseSeq, 136
 - calledby parseWhere, 137
- Mapping, 211
 - defplist, 211
- match-advance-string, 347
 - calledby PARSE-Category, 315
 - calledby PARSE-Command, 310
 - calledby PARSE-Conditional, 343
 - calledby PARSE-Enclosure, 330
 - calledby PARSE-Exit, 341
 - calledby PARSE-FloatBasePart, 328
 - calledby PARSE-FloatExponent, 329
 - calledby PARSE-Form, 323
 - calledby PARSE-Import, 317
 - calledby PARSE-IteratorTail, 339
 - calledby PARSE-Iterator, 339
 - calledby PARSE-Label, 325
 - calledby PARSE-Leave, 342
 - calledby PARSE-Loop, 344
 - calledby PARSE-Option, 314
 - calledby PARSE-Primary1, 327
 - calledby PARSE-PrimaryOrQM, 313
 - calledby PARSE-Quad, 331
 - calledby PARSE-Qualification, 322
 - calledby PARSE-Return, 341
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-Scripts, 332
 - calledby PARSE-Selector, 325
 - calledby PARSE-SemiColon, 341
 - calledby PARSE-Sequence, 337
 - calledby PARSE-Sexpr1, 334
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-Statement, 314
 - calledby PARSE-TokenOption, 312
 - calledby PARSE-With, 315
 - calls current-token, 347
 - calls line-current-char, 347
 - calls line-current-index, 347
 - calls line-past-end-p, 347
 - calls match-string, 347
 - calls quote-if-string, 347
 - uses \$line, 347
 - uses \$token, 347
 - defun, 347
- match-current-token, 351
 - calledby PARSE-GlyphTok, 336
 - calledby PARSE-NBGlyphTok, 335
 - calledby PARSE-Operation, 319
 - calledby PARSE-SpecialKeyWord, 310
 - calledby parse-argument-designator, 367
 - calledby parse-identifier, 365
 - calledby parse-keyword, 366
 - calledby parse-number, 366
 - calledby parse-spadstring, 364
 - calledby parse-string, 365

- calls current-token, 351
- calls match-token, 351
- defun, 351
- match-next-token, 352
 - calledby PARSE-ReductionOp, 323
 - calls match-token, 352
 - calls next-token, 352
 - defun, 352
- match-string, 346
 - calledby PARSE-AnyId, 336
 - calledby PARSE-NewExpr, 309
 - calledby PARSE-Primary1, 327
 - calledby match-advance-string, 347
 - calls current-char, 346
 - calls initial-substring-p, 346
 - calls line-past-end-p, 346
 - calls skip-blanks, 346
 - calls subseq, 346
 - calls unget-tokens, 346
 - uses \$line, 346
 - uses line, 346
 - defun, 346
- match-token, 351
 - calledby match-current-token, 351
 - calledby match-next-token, 352
 - calls token-symbol, 352
 - calls token-type, 351
 - defun, 351
- maxindex
 - calledby compDefineFunctor1, 167
 - calledby floatexpid, 357
 - calledby preparsel, 81
 - calledby preparsereadline1, 87
 - calledby translable1, 361
- maxSuperType
 - calledby setqSingle, 247
- mdef, 131, 235
 - defplist, 131, 235
- member
 - calledby addDomain, 187
 - calledby augLisplibModemapsFromCategory, 153
 - calledby augModemapsFromDomain, 190
 - calledby augmentLisplibModemapsFromFunctor, 174
 - calledby displayMissingFunctions, 177
 - calledby genDomainView, 181
 - calledby isListConstructor, 111
 - calledby mkNewModemapList, 198
 - calledby parseHasRhs, 118
 - calledby parseHas, 117
 - calledby putDomainsInScope, 189
 - calledby transformOperationAlist, 164
- member[5]
 - called by comp3, 406
 - called by compColon, 216
 - called by compSymbol, 412
 - called by compilerDoit, 384
- mergeModemap, 199
 - calledby mkNewModemapList, 198
 - calls TruthP, 199
 - calls isSuperDomain, 199
 - local ref \$forceAdd, 199
 - defun, 199
- mergePathnames[5]
 - called by compiler, 379
- mergeSignatureAndLocalVarAlists, 165
 - calledby finalizeLisplib, 160
 - calls lassoc, 165
 - defun, 165
- meta-error-handler, 356
 - calledby meta-syntax-error, 357
 - usedby meta-syntax-error, 357
 - defvar, 356
- meta-syntax-error, 357
 - calledby must, 358
 - calls meta-error-handler, 357
 - uses meta-error-handler, 357
 - defun, 357
- mismatch
 - calledby initial-substring, 93
- mkAlistOfExplicitCategoryOps
 - calledby augLisplibModemapsFromCategory, 152
 - calledby augmentLisplibModemapsFromFunctor, 174
- mkCategoryPackage, 146
 - calledby compDefineCategory1, 145
 - calls JoinInner, 146
 - calls abbreviationsSpad2Cmd, 146
 - calls assoc, 146
 - calls getdatabase, 146

- calls msubst, 147
- calls pname, 146
- calls strconc, 146
- calls sublislis, 147
- uses \$FormalMapVariableList, 147
- uses \$categoryPredicateList, 147
- uses \$e, 147
- uses \$options, 147
- defun, 146
- mkConstructor, 154
 - calledby compDefineCategory2, 149
 - calledby mkConstructor, 154
 - calls mkConstructor, 154
 - defun, 154
- mkDatabasePred
 - calledby augmentLisplibModemapsFrom-Functor, 174
- mkDomainConstructor
 - calledby genDomainOps, 182
 - calledby genDomainView, 181
- mkErrorExpr
 - calledby compOrCroak1, 402
- mkEvalableCategoryForm, 148
 - calledby mkEvalableCategoryForm, 148
 - calls compOrCroak, 148
 - calls getdatabase, 148
 - calls get, 148
 - calls mkEvalableCategoryForm, 148
 - calls mkq, 148
 - calls pairp, 148
 - calls qcar, 148
 - calls qcdr, 148
 - calls quotifyCategoryArgument, 148
 - local def \$e, 148
 - local ref \$CategoryFrame, 148
 - local ref \$CategoryNames, 148
 - local ref \$Category, 148
 - local ref \$EmptyMode, 148
 - local ref \$e, 148
 - defun, 148
- mkExplicitCategoryFunction
 - calledby compCategory, 212
- mkNewModemapList, 198
 - calledby addModemap1, 198
 - calls assoc, 198
 - calls insertModemap, 199
 - calls member, 198
 - calls mergeModemap, 198
 - calls nequal, 198
 - calls nreverse0, 198
 - calls pairp, 198
 - calls qcar, 198
 - calls qcdr, 198
 - local ref \$InteractiveMode, 199
 - local ref \$forceAdd, 199
 - defun, 198
- mkOpVec, 183
 - calls AssocBarGensym, 183
 - calls assoc, 183
 - calls assq, 183
 - calls getOperationAlistFromLisplib, 183
 - calls getPrincipalView, 183
 - calls length, 183
 - calls msubst, 183
 - calls opOf, 183
 - calls pairp, 183
 - calls qcar, 183
 - calls qcdr, 183
 - calls sublis, 183
 - uses Undef, 183
 - uses \$FormalMapVariableList, 183
 - defun, 183
- mkpf
 - calledby augLisplibModemapsFromCat-egory, 153
 - calledby augmentLisplibModemapsFrom-Functor, 174
 - calledby compCapsuleInner, 208
- mkprompt[5]
 - called by get-a-line, 94
- mkq
 - calledby compCoerce1, 214
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
 - calledby compSeq1, 244
 - calledby genDomainOps, 182
 - calledby mkEvalableCategoryForm, 148
- moan
 - calledby parseExit, 115
 - calledby parseReturn, 135
- modeEqual
 - calledby compCase1, 210

- calledby domainMember, 195
- modeIsAggregateOf
 - calledby compAtom, 409
 - calledby compConstruct, 220
 - calledby compRepeatOrCollect, 240
- modemap2Signature
 - calledby compDefineFunctor1, 167
- modifyModeStack, 429
 - calledby compExit, 227
 - calledby compLeave, 234
 - calledby compReturn, 243
 - calls copy, 429
 - calls resolve, 429
 - calls say, 429
 - calls setelt, 429
 - calls setelt, 429
 - uses \$exitModeStack, 429
 - uses \$reportExitModeStack, 429
 - defun, 429
- msubst
 - calledby addConstructorModemaps, 192
 - calledby addModemap1, 198
 - calledby augModemapsFromCategoryRep, 203
 - calledby augmentLisplibModemapsFromFunctor, 174
 - calledby compCoerce1, 214
 - calledby compReduce1, 238
 - calledby compRepeatOrCollect, 240
 - calledby compSubsetCategory, 252
 - calledby hasAplExtension, 295
 - calledby makeFunctorArgumentParameters, 178
 - calledby mkCategoryPackage, 147
 - calledby mkOpVec, 183
 - calledby parseDollarGreaterEqual, 112
 - calledby parseDollarGreaterThan, 112
 - calledby parseDollarLessEqual, 113
 - calledby parseDollarNotEqual, 114
 - calledby parseNotEqual, 133
 - calledby parseTransform, 101
 - calledby parseType, 106
 - calledby substituteCategoryArguments, 192
- must, 358
 - calledby PARSE-Category, 315
 - calledby PARSE-Command, 310
 - calledby PARSE-Conditional, 343
 - calledby PARSE-Enclosure, 330
 - calledby PARSE-Exit, 341
 - calledby PARSE-FloatBasePart, 329
 - calledby PARSE-FloatBase, 328
 - calledby PARSE-FloatExponent, 329
 - calledby PARSE-Float, 327
 - calledby PARSE-Form, 323
 - calledby PARSE-Import, 317
 - calledby PARSE-Infix, 321
 - calledby PARSE-Iterator, 339
 - calledby PARSE-LabelExpr, 344
 - calledby PARSE-Label, 325
 - calledby PARSE-Leave, 342
 - calledby PARSE-Loop, 344
 - calledby PARSE-NewExpr, 309
 - calledby PARSE-Option, 314
 - calledby PARSE-Prefix, 321
 - calledby PARSE-Primary1, 326
 - calledby PARSE-Qualification, 322
 - calledby PARSE-Reduction, 323
 - calledby PARSE-Return, 341
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-Scripts, 332
 - calledby PARSE-Selector, 325
 - calledby PARSE-SemiColon, 341
 - calledby PARSE-Sequence, 337
 - calledby PARSE-Sexpr1, 334
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-Statement, 314
 - calledby PARSE-TokenOption, 312
 - calledby PARSE-With, 315
 - calls meta-syntax-error, 358
 - defmacro, 358
- namestring
 - calledby finalizeLisplib, 160
- namestring[5]
 - called by compileSpad2Cmd, 382
 - called by compileSpadLispCmd, 431
 - called by compiler, 379
- ncINTERPFILE, 431
 - calledby /rf-1, 386
 - calls SpadInterpretStream[5], 431
 - uses \$EchoLines, 431
 - uses \$ReadingFile, 431
 - defun, 431

- nequal
 - calledby comp2, 405
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
 - calledby compElt, 226
 - calledby compPretend, 236
 - calledby compReturn, 242
 - calledby compileSpad2Cmd, 382
 - calledby displayPreCompilationErrors, 362
 - calledby mkNewModemapList, 198
 - calledby postDef, 277
 - calledby postError, 262
 - calledby setqSingle, 247
- new2OldLisp, 364
 - calledby s-process, 396
 - calls new2OldTran, 364
 - calls postTransform, 364
 - defun, 364
- new2OldTran
 - calledby new2OldLisp, 364
- newComp
 - calledby compTopLevel, 400
- next-char, 355
 - calledby PARSE-FloatBase, 328
 - calls line-at-end-p, 356
 - calls line-next-char, 356
 - uses current-line, 356
 - defun, 355
- next-line, 92
 - calledby Advance-Char, 93
 - defun, 92
- next-tab-loc, 373
 - defun, 373
- next-token, 97, 354
 - calledby match-next-token, 352
 - calls current-token, 354
 - calls try-get-token, 354
 - usedby next-token, 354
 - uses \$token, 97
 - uses next-token, 354
 - uses valid-tokens, 354
 - defun, 354
 - defvar, 97
- nonblank, 97
 - defvar, 97
- nonblankloc, 374
 - calledby add-parens-and-semis-to-line, 84
 - calls blankp, 374
 - defun, 374
- normalizeStatAndStringify
 - calledby reportOnFunctorCompilation, 177
- not, 132
 - defplist, 132
- notequal, 133
 - defplist, 133
- nreverse0
 - calledby aplTran1, 293
 - calledby compAdd, 205
 - calledby compCase1, 210
 - calledby compColon, 216
 - calledby compForm1, 414
 - calledby compForm2, 416
 - calledby compJoin, 231
 - calledby compReduce1, 238
 - calledby compSeq1, 244
 - calledby getModemapList, 194
 - calledby hasAplExtension, 295
 - calledby mkNewModemapList, 198
 - calledby parseHas, 117
 - calledby postCategory, 269
 - calledby postDef, 277
 - calledby postIf, 279
 - calledby postMDef, 283
 - calledby substNames, 202
 - calledby transIs1, 110
- NRTassocIndex
 - calledby setqSingle, 247
- NRTgenInitialAttributeAlist
 - calledby compDefineFunctor1, 167
 - calledby finalizeLisplib, 160
- NRTgetLocalIndex
 - calledby compAdd, 205
 - calledby compDefineFunctor1, 167
 - calledby compSymbol, 412
- NRTgetLookupFunction
 - calledby compDefineFunctor1, 167
- NRTmakeSlot1Info
 - calledby compDefineFunctor1, 167
- nth-stack, 370
 - calledby PARSE-Category, 316
 - calledby PARSE-Sexpr1, 334
 - calls reduction-value, 370

- calls stack-store, 370
- defmacro, 370
- object2String
 - calledby compileSpad2Cmd, 382
 - calledby compileSpadLispCmd, 432
- opFf
 - calledby parseDEF, 109
- opOf
 - calledby augModemapsFromDomain, 190
 - calledby comp2, 405
 - calledby compColonInside, 408
 - calledby compDefineCategory2, 149
 - calledby compElt, 225
 - calledby compPretend, 236
 - calledby compilerDoit, 384
 - calledby isFunctor, 188
 - calledby isSuperDomain, 190
 - calledby mkOpVec, 183
 - calledby parseHas, 116
 - calledby parseLET, 130
 - calledby parseMDEF, 131
- optFunctorBody
 - calledby compDefineCategory2, 149
- optimizeFunctionDef
 - calledby compWithMappingModel1, 422
- optional, 359
 - calledby PARSE-Application, 324
 - calledby PARSE-Category, 315
 - calledby PARSE-CommandTail, 313
 - calledby PARSE-Conditional, 343
 - calledby PARSE-Expr, 318
 - calledby PARSE-Form, 323
 - calledby PARSE-Import, 317
 - calledby PARSE-Infix, 321
 - calledby PARSE-IteratorTail, 339
 - calledby PARSE-Iterator, 339
 - calledby PARSE-Prefix, 321
 - calledby PARSE-Primary1, 326
 - calledby PARSE-PrimaryNoFloat, 326
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-Seg, 342
 - calledby PARSE-Sequence1, 337
 - calledby PARSE-Sexpr1, 334
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-Statement, 314
 - calledby PARSE-Suffix, 340
 - calledby PARSE-TokenCommandTail, 312
 - calledby PARSE-VarForm, 332
 - defun, 359
- or, 133
 - defplist, 133
- orderByDependency
 - calledby compDefWhereClause, 184
- outputComp
 - calledby compForm1, 414
 - calledby setqSingle, 247
- pack
 - calledby quote-if-string, 348
- pairList
 - calledby compDefWhereClause, 184
- pairp
 - calledby addConstructorModemaps, 192
 - calledby addDomain, 187
 - calledby addEltModemap, 197
 - calledby addModemap0, 196
 - calledby compAdd, 205
 - calledby compCons1, 219
 - calledby compDefWhereClause, 184
 - calledby compDefineFunctor1, 167
 - calledby compJoin, 231
 - calledby genDomainViewList, 181
 - calledby genDomainView, 181
 - calledby getModemapList, 194
 - calledby isDomainConstructorForm, 249
 - calledby isDomainForm, 248
 - calledby makeFunctorArgumentParameters, 178
 - calledby mkEvalableCategoryForm, 148
 - calledby mkNewModemapList, 198
 - calledby mkOpVec, 183
 - calledby postSignature, 288
 - calledby postTran, 258
 - calledby transIs1, 110
- PARSE-AnyId, 336
 - calledby PARSE-Sexpr1, 334
 - calls action, 336
 - calls advance-token, 336
 - calls current-symbol, 336
 - calls match-string, 336
 - calls parse-identifier, 336

- calls parse-keyword, 336
- calls push-reduction, 336
- defun, 336
- PARSE-Application, 324
 - calledby PARSE-Application, 324
 - calledby PARSE-Category, 316
 - calledby PARSE-Form, 324
 - calls PARSE-Application, 324
 - calls PARSE-Primary, 324
 - calls PARSE-Selector, 324
 - calls optional, 324
 - calls pop-stack-1, 324
 - calls pop-stack-2, 324
 - calls push-reduction, 324
 - calls star, 324
 - defun, 324
- parse-argument-designator, 367
 - calledby PARSE-FormalParameterTok, 331
 - calls advance-token, 367
 - calls match-current-token, 367
 - calls push-reduction, 367
 - calls token-symbol, 367
 - defun, 367
- PARSE-Category, 315
 - calledby PARSE-Category, 315
 - calls PARSE-Application, 316
 - calls PARSE-Category, 315
 - calls PARSE-Expression, 315
 - calls action, 316
 - calls bang, 315
 - calls line-number, 316
 - calls match-advance-string, 315
 - calls must, 315
 - calls nth-stack, 316
 - calls optional, 315
 - calls pop-stack-1, 316
 - calls pop-stack-2, 315
 - calls pop-stack-3, 315
 - calls push-reduction, 315
 - calls recordAttributeDocumentation, 316
 - calls recordSignatureDocumentation, 316
 - calls star, 316
 - uses current-line, 316
 - defun, 315
- PARSE-Command, 310
 - calls PARSE-SpecialCommand, 310
 - calls PARSE-SpecialKeyWord, 310
 - calls match-advance-string, 310
 - calls must, 310
 - calls push-reduction, 310
 - defun, 310
- PARSE-CommandTail, 313
 - calledby PARSE-CommandTail, 313
 - calledby PARSE-SpecialCommand, 311
 - calls PARSE-CommandTail, 313
 - calls PARSE-Option, 313
 - calls action, 313
 - calls bang, 313
 - calls optional, 313
 - calls pop-stack-1, 313
 - calls pop-stack-2, 313
 - calls push-reduction, 313
 - calls star, 313
 - calls systemCommand[5], 313
 - defun, 313
- PARSE-Conditional, 343
 - calledby PARSE-ElseClause, 343
 - calls PARSE-ElseClause, 343
 - calls PARSE-Expression, 343
 - calls bang, 343
 - calls match-advance-string, 343
 - calls must, 343
 - calls optional, 343
 - calls pop-stack-1, 343
 - calls pop-stack-2, 343
 - calls pop-stack-3, 343
 - calls push-reduction, 343
 - defun, 343
- PARSE-Data, 334
 - calledby PARSE-Primary1, 327
 - calls PARSE-Sexpr, 334
 - calls action, 334
 - calls pop-stack-1, 334
 - calls push-reduction, 334
 - calls translabel, 334
 - uses labasoc, 334
 - defun, 334
- PARSE-ElseClause, 343
 - calledby PARSE-Conditional, 343
 - calls PARSE-Conditional, 343
 - calls PARSE-Expression, 344
 - calls current-symbol, 343

- defun, 343
- PARSE-Enclosure, 330
 - calledby PARSE-Primary1, 327
 - calls PARSE-Expr, 330
 - calls match-advance-string, 330
 - calls must, 330
 - calls pop-stack-1, 330
 - calls push-reduction, 330
 - defun, 330
- PARSE-Exit, 341
 - calls PARSE-Expression, 341
 - calls match-advance-string, 341
 - calls must, 341
 - calls pop-stack-1, 342
 - calls push-reduction, 341
 - defun, 341
- PARSE-Expr, 318
 - calledby PARSE-Enclosure, 330
 - calledby PARSE-Expression, 317
 - calledby PARSE-Import, 317
 - calledby PARSE-Iterator, 339
 - calledby PARSE-LabelExpr, 344
 - calledby PARSE-Loop, 344
 - calledby PARSE-Primary1, 327
 - calledby PARSE-Reduction, 323
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-SemiColon, 341
 - calledby PARSE-Statement, 314
 - calls PARSE-LedPart, 318
 - calls PARSE-NudPart, 318
 - calls optional, 318
 - calls pop-stack-1, 318
 - calls push-reduction, 318
 - calls star, 318
 - defun, 318
- PARSE-Expression, 317
 - calledby PARSE-Category, 315
 - calledby PARSE-Conditional, 343
 - calledby PARSE-ElseClause, 344
 - calledby PARSE-Exit, 341
 - calledby PARSE-Infix, 321
 - calledby PARSE-Iterator, 339
 - calledby PARSE-Leave, 342
 - calledby PARSE-Prefix, 321
 - calledby PARSE-Return, 341
 - calledby PARSE-Seg, 342
 - calledby PARSE-Sequence1, 337
 - calledby PARSE-SpecialCommand, 311
 - calls PARSE-Expr, 317
 - calls PARSE-rightBindingPowerOf, 317
 - calls make-symbol-of, 317
 - calls pop-stack-1, 317
 - calls push-reduction, 317
 - uses ParseMode, 317
 - uses prior-token, 317
 - defun, 317
- PARSE-Float, 327
 - calledby PARSE-Primary, 326
 - calledby PARSE-Selector, 325
 - calls PARSE-FloatBase, 327
 - calls PARSE-FloatExponent, 327
 - calls make-float, 327
 - calls must, 327
 - calls pop-stack-1, 327
 - calls pop-stack-2, 327
 - calls pop-stack-3, 327
 - calls pop-stack-4, 327
 - calls push-reduction, 327
 - defun, 327
- PARSE-FloatBase, 328
 - calledby PARSE-Float, 327
 - calls PARSE-FloatBasePart, 328
 - calls PARSE-IntegerTok, 328
 - calls char-eq, 328
 - calls char-ne, 328
 - calls current-char, 328
 - calls current-symbol, 328
 - calls digitp[5], 328
 - calls must, 328
 - calls next-char, 328
 - calls push-reduction, 328
 - defun, 328
- PARSE-FloatBasePart, 328
 - calledby PARSE-FloatBase, 328
 - calls PARSE-IntegerTok, 329
 - calls current-char, 329
 - calls current-token, 329
 - calls digitp[5], 329
 - calls match-advance-string, 328
 - calls must, 329
 - calls push-reduction, 329
 - calls token-nonblank, 329

- defun, 328
- PARSE-FloatExponent, 329
 - calledby PARSE-Float, 327
 - calls PARSE-IntegerTok, 329
 - calls action, 329
 - calls advance-token, 329
 - calls current-char, 329
 - calls current-symbol, 329
 - calls floatexpid, 329
 - calls identp[5], 329
 - calls match-advance-string, 329
 - calls must, 329
 - calls push-reduction, 329
 - defun, 329
- PARSE-FloatTok, 345
 - calls bfp-, 345
 - calls parse-number, 345
 - calls pop-stack-1, 345
 - calls push-reduction, 345
 - uses \$boot, 345
 - defun, 345
- PARSE-Form, 323
 - calledby PARSE-NudPart, 318
 - calls PARSE-Application, 324
 - calls bang, 323
 - calls match-advance-string, 323
 - calls must, 323
 - calls optional, 323
 - calls pop-stack-1, 324
 - calls push-reduction, 323
 - defun, 323
- PARSE-FormalParameter, 331
 - calledby PARSE-Primary1, 327
 - calls PARSE-FormalParameterTok, 331
 - defun, 331
- PARSE-FormalParameterTok, 331
 - calledby PARSE-FormalParameter, 331
 - calls parse-argument-designator, 331
 - defun, 331
- PARSE-getSemanticForm, 320
 - calledby PARSE-Operation, 319
 - calls PARSE-Infix, 320
 - calls PARSE-Prefix, 320
 - defun, 320
- PARSE-GlyphTok, 336
 - calledby PARSE-Quad, 331
- calledby PARSE-Seg, 342
- calledby PARSE-Sexpr1, 335
- calls action, 336
- calls advance-token, 336
- calls match-current-token, 336
- uses tok, 336
- defun, 336
- parse-identifier, 365
 - calledby PARSE-AnyId, 336
 - calledby PARSE-Name, 333
 - calls advance-token, 365
 - calls match-current-token, 365
 - calls push-reduction, 365
 - calls token-symbol, 365
 - defun, 365
- PARSE-Import, 317
 - calls PARSE-Expr, 317
 - calls bang, 317
 - calls match-advance-string, 317
 - calls must, 317
 - calls optional, 317
 - calls pop-stack-1, 317
 - calls pop-stack-2, 317
 - calls push-reduction, 317
 - calls star, 317
 - defun, 317
- PARSE-Infix, 321
 - calledby PARSE-getSemanticForm, 320
 - calls PARSE-Expression, 321
 - calls PARSE-TokTail, 321
 - calls action, 321
 - calls advance-token, 321
 - calls current-symbol, 321
 - calls must, 321
 - calls optional, 321
 - calls pop-stack-1, 321
 - calls pop-stack-2, 321
 - calls push-reduction, 321
 - defun, 321
- PARSE-InfixWith, 315
 - calls PARSE-With, 315
 - calls pop-stack-1, 315
 - calls pop-stack-2, 315
 - calls push-reduction, 315
 - defun, 315
- PARSE-IntegerTok, 330

- calledby PARSE-FloatBasePart, 329
- calledby PARSE-FloatBase, 328
- calledby PARSE-FloatExponent, 329
- calledby PARSE-Primary1, 327
- calledby PARSE-Sexpr1, 334
- calls parse-number, 330
- defun, 330
- PARSE-Iterator, 339
 - calledby PARSE-IteratorTail, 339
 - calledby PARSE-Loop, 344
 - calls PARSE-Expression, 339
 - calls PARSE-Expr, 339
 - calls PARSE-Primary, 339
 - calls match-advance-string, 339
 - calls must, 339
 - calls optional, 339
 - calls pop-stack-1, 339
 - calls pop-stack-2, 339
 - calls pop-stack-3, 339
 - defun, 339
- PARSE-IteratorTail, 339
 - calledby PARSE-Sequence1, 337
 - calls PARSE-Iterator, 339
 - calls bang, 339
 - calls match-advance-string, 339
 - calls optional, 339
 - calls star, 339
 - defun, 339
- parse-keyword, 366
 - calledby PARSE-AnyId, 336
 - calls advance-token, 366
 - calls match-current-token, 366
 - calls push-reduction, 366
 - calls token-symbol, 366
 - defun, 366
- PARSE-Label, 325
 - calledby PARSE-LabelExpr, 344
 - calledby PARSE-Leave, 342
 - calls PARSE-Name, 325
 - calls match-advance-string, 325
 - calls must, 325
 - defun, 325
- PARSE-LabelExpr, 344
 - calls PARSE-Expr, 344
 - calls PARSE-Label, 344
 - calls must, 344
- calls pop-stack-1, 344
- calls pop-stack-2, 344
- calls push-reduction, 344
- defun, 344
- PARSE-Leave, 342
 - calls PARSE-Expression, 342
 - calls PARSE-Label, 342
 - calls match-advance-string, 342
 - calls must, 342
 - calls pop-stack-1, 342
 - calls push-reduction, 342
 - defun, 342
- PARSE-LedPart, 318
 - calledby PARSE-Expr, 318
 - calls PARSE-Operation, 318
 - calls pop-stack-1, 318
 - calls push-reduction, 318
 - defun, 318
- PARSE-leftBindingPowerOf, 319
 - calledby PARSE-Operation, 319
 - calls elemn, 320
 - calls getl, 319
 - defun, 319
- PARSE-Loop, 344
 - calls PARSE-Expr, 344
 - calls PARSE-Iterator, 344
 - calls match-advance-string, 344
 - calls must, 344
 - calls pop-stack-1, 344
 - calls pop-stack-2, 344
 - calls push-reduction, 344
 - calls star, 344
 - defun, 344
- PARSE-Name, 333
 - calledby PARSE-Label, 325
 - calledby PARSE-VarForm, 332
 - calls parse-identifier, 333
 - calls pop-stack-1, 333
 - calls push-reduction, 333
 - defun, 333
- PARSE-NBGliphTok, 335
 - calledby PARSE-Sexpr1, 334
 - calls action, 335
 - calls advance-token, 335
 - calls match-current-token, 335
 - uses tok, 335

- defun, 335
- PARSE-NewExpr, 309
 - calledby spad, 395
 - calls PARSE-Statement, 309
 - calls action, 309
 - calls current-symbol, 309
 - calls match-string, 309
 - calls must, 309
 - calls processSynonyms[5], 309
 - uses definition-name, 309
 - defun, 309
- PARSE-NudPart, 318
 - calledby PARSE-Expr, 318
 - calls PARSE-Form, 318
 - calls PARSE-Operation, 318
 - calls PARSE-Reduction, 318
 - calls pop-stack-1, 319
 - calls push-reduction, 318
 - uses rbp, 319
 - defun, 318
- parse-number, 366
 - calledby PARSE-FloatTok, 345
 - calledby PARSE-IntegerTok, 330
 - calls advance-token, 366
 - calls match-current-token, 366
 - calls push-reduction, 366
 - calls token-symbol, 366
 - defun, 366
- PARSE-OpenBrace, 338
 - calledby PARSE-Sequence, 337
 - calls action, 338
 - calls advance-token, 338
 - calls current-symbol, 338
 - calls eqcar, 338
 - calls getToken, 338
 - calls push-reduction, 338
 - defun, 338
- PARSE-OpenBracket, 338
 - calledby PARSE-Sequence, 337
 - calls action, 338
 - calls advance-token, 338
 - calls current-symbol, 338
 - calls eqcar, 338
 - calls getToken, 338
 - calls push-reduction, 338
 - defun, 338
- PARSE-Operation, 319
 - calledby PARSE-LedPart, 318
 - calledby PARSE-NudPart, 318
 - calls PARSE-getSemanticForm, 319
 - calls PARSE-leftBindingPowerOf, 319
 - calls PARSE-rightBindingPowerOf, 319
 - calls action, 319
 - calls current-symbol, 319
 - calls elemn, 319
 - calls getl, 319
 - calls lt, 319
 - calls match-current-token, 319
 - uses ParseMode, 319
 - uses rbp, 319
 - uses tmptok, 319
 - defun, 319
- PARSE-Option, 314
 - calledby PARSE-CommandTail, 313
 - calls PARSE-PrimaryOrQM, 314
 - calls match-advance-string, 314
 - calls must, 314
 - calls star, 314
 - defun, 314
- PARSE-Prefix, 320
 - calledby PARSE-getSemanticForm, 320
 - calls PARSE-Expression, 321
 - calls PARSE-TokTail, 321
 - calls action, 320
 - calls advance-token, 321
 - calls current-symbol, 320
 - calls must, 321
 - calls optional, 321
 - calls pop-stack-1, 321
 - calls pop-stack-2, 321
 - calls push-reduction, 320, 321
 - defun, 320
- PARSE-Primary, 326
 - calledby PARSE-Application, 324
 - calledby PARSE-Iterator, 339
 - calledby PARSE-PrimaryOrQM, 313
 - calledby PARSE-Selector, 325
 - calls PARSE-Float, 326
 - calls PARSE-PrimaryNoFloat, 326
 - defun, 326
- PARSE-Primary1, 326
 - calledby PARSE-Primary1, 326

- calledby PARSE-PrimaryNoFloat, 326
- calledby PARSE-Qualification, 322
- calls PARSE-Data, 327
- calls PARSE-Enclosure, 327
- calls PARSE-Expr, 327
- calls PARSE-FormalParameter, 327
- calls PARSE-IntegerTok, 327
- calls PARSE-Primary1, 326
- calls PARSE-Quad, 327
- calls PARSE-Sequence, 327
- calls PARSE-String, 327
- calls PARSE-VarForm, 326
- calls current-symbol, 326
- calls match-advance-string, 327
- calls match-string, 327
- calls must, 326
- calls optional, 326
- calls pop-stack-1, 326
- calls pop-stack-2, 326
- calls push-reduction, 326
- uses \$boot, 327
- defun, 326
- PARSE-PrimaryNoFloat, 326
 - calledby PARSE-Primary, 326
 - calledby PARSE-Selector, 325
 - calls PARSE-Primary1, 326
 - calls PARSE-TokTail, 326
 - calls optional, 326
 - defun, 326
- PARSE-PrimaryOrQM, 313
 - calledby PARSE-Option, 314
 - calledby PARSE-PrimaryOrQM, 313
 - calledby PARSE-SpecialCommand, 311
 - calls PARSE-PrimaryOrQM, 313
 - calls PARSE-Primary, 313
 - calls match-advance-string, 313
 - calls push-reduction, 313
 - defun, 313
- PARSE-Quad, 331
 - calledby PARSE-Primary1, 327
 - calls PARSE-GlyphTok, 331
 - calls match-advance-string, 331
 - calls push-reduction, 331
 - uses \$boot, 331
 - defun, 331
- PARSE-Qualification, 322
 - calledby PARSE-TokTail, 322
 - calls PARSE-Primary1, 322
 - calls dollarTran, 322
 - calls match-advance-string, 322
 - calls must, 322
 - calls pop-stack-1, 322
 - calls push-reduction, 322
 - defun, 322
- PARSE-Reduction, 323
 - calledby PARSE-NudPart, 318
 - calls PARSE-Expr, 323
 - calls PARSE-ReductionOp, 323
 - calls must, 323
 - calls pop-stack-1, 323
 - calls pop-stack-2, 323
 - calls push-reduction, 323
 - defun, 323
- PARSE-ReductionOp, 323
 - calledby PARSE-Reduction, 323
 - calls action, 323
 - calls advance-token, 323
 - calls current-symbol, 323
 - calls getl, 323
 - calls match-next-token, 323
 - defun, 323
- PARSE-Return, 341
 - calls PARSE-Expression, 341
 - calls match-advance-string, 341
 - calls must, 341
 - calls pop-stack-1, 341
 - calls push-reduction, 341
 - defun, 341
- PARSE-rightBindingPowerOf, 320
 - calledby PARSE-Expression, 317
 - calledby PARSE-Operation, 319
 - calls elemn, 320
 - calls getl, 320
 - defun, 320
- PARSE-ScriptItem, 333
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-Scripts, 332
 - calls PARSE-Expr, 333
 - calls PARSE-ScriptItem, 333
 - calls match-advance-string, 333
 - calls must, 333
 - calls optional, 333

- calls pop-stack-1, 333
- calls pop-stack-2, 333
- calls push-reduction, 333
- calls star, 333
- defun, 333
- PARSE-Scripts, 332
 - calledby PARSE-VarForm, 332
 - calls PARSE-ScriptItem, 332
 - calls match-advance-string, 332
 - calls must, 332
 - defun, 332
- PARSE-Seg, 342
 - calls PARSE-Expression, 342
 - calls PARSE-GlyphTok, 342
 - calls bang, 342
 - calls optional, 342
 - calls pop-stack-1, 343
 - calls pop-stack-2, 343
 - calls push-reduction, 342
 - defun, 342
- PARSE-Selector, 325
 - calledby PARSE-Application, 324
 - calls PARSE-Float, 325
 - calls PARSE-PrimaryNoFloat, 325
 - calls PARSE-Primary, 325
 - calls char-ne, 325
 - calls current-char, 325
 - calls current-symbol, 325
 - calls match-advance-string, 325
 - calls must, 325
 - calls pop-stack-1, 325
 - calls pop-stack-2, 325
 - calls push-reduction, 325
 - uses \$boot, 325
 - defun, 325
- PARSE-SemiColon, 341
 - calls PARSE-Expr, 341
 - calls match-advance-string, 341
 - calls must, 341
 - calls pop-stack-1, 341
 - calls pop-stack-2, 341
 - calls push-reduction, 341
 - defun, 341
- PARSE-Sequence, 337
 - calledby PARSE-Primary1, 327
 - calls PARSE-OpenBrace, 337
 - calls PARSE-OpenBracket, 337
 - calls PARSE-Sequence1, 337
 - calls match-advance-string, 337
 - calls must, 337
 - calls pop-stack-1, 337
 - calls push-reduction, 337
 - defun, 337
- PARSE-Sequence1, 337
 - calledby PARSE-Sequence, 337
 - calls PARSE-Expression, 337
 - calls PARSE-IteratorTail, 337
 - calls optional, 337
 - calls pop-stack-1, 337
 - calls pop-stack-2, 337
 - calls push-reduction, 337
 - defun, 337
- PARSE-Sexpr, 334
 - calledby PARSE-Data, 334
 - calls PARSE-Sexpr1, 334
 - defun, 334
- PARSE-Sexpr1, 334
 - calledby PARSE-Sexpr1, 334
 - calledby PARSE-Sexpr, 334
 - calls PARSE-AnyId, 334
 - calls PARSE-GlyphTok, 335
 - calls PARSE-IntegerTok, 334
 - calls PARSE-NBGlyphTok, 334
 - calls PARSE-Sexpr1, 334
 - calls PARSE-String, 335
 - calls action, 334
 - calls bang, 335
 - calls match-advance-string, 334
 - calls must, 334
 - calls nth-stack, 334
 - calls optional, 334
 - calls pop-stack-1, 335
 - calls pop-stack-2, 334
 - calls push-reduction, 334
 - calls star, 335
 - defun, 334
- parse-spadstring, 364
 - calledby PARSE-String, 331
 - calls advance-token, 365
 - calls match-current-token, 364
 - calls push-reduction, 364
 - calls token-symbol, 364

- defun, 364
- PARSE-SpecialCommand, 311
 - calledby PARSE-Command, 310
 - calledby PARSE-SpecialCommand, 311
 - calls PARSE-CommandTail, 311
 - calls PARSE-Expression, 311
 - calls PARSE-PrimaryOrQM, 311
 - calls PARSE-SpecialCommand, 311
 - calls PARSE-TokenCommandTail, 311
 - calls PARSE-TokenList, 311
 - calls action, 311
 - calls bang, 311
 - calls current-symbol, 311
 - calls match-advance-string, 311
 - calls must, 311
 - calls optional, 311
 - calls pop-stack-1, 311
 - calls push-reduction, 311
 - calls star, 311
 - uses \$noParseCommands, 311
 - uses \$tokenCommands, 311
 - defun, 311
- PARSE-SpecialKeyWord, 310
 - calledby PARSE-Command, 310
 - calls action, 310
 - calls current-symbol, 310
 - calls current-token, 310
 - calls match-current-token, 310
 - calls token-symbol, 310
 - calls unAbbreviateKeyword[5], 310
 - defun, 310
- PARSE-Statement, 314
 - calledby PARSE-NewExpr, 309
 - calls PARSE-Expr, 314
 - calls match-advance-string, 314
 - calls must, 314
 - calls optional, 314
 - calls pop-stack-1, 314
 - calls pop-stack-2, 314
 - calls push-reduction, 314
 - calls star, 314
 - defun, 314
- PARSE-String, 331
 - calledby PARSE-Primary1, 327
 - calledby PARSE-Sexpr1, 335
 - calls parse-spadstring, 331
- defun, 331
- parse-string, 365
 - calls advance-token, 365
 - calls match-current-token, 365
 - calls push-reduction, 365
 - calls token-symbol, 365
 - defun, 365
- PARSE-Suffix, 340
 - calls PARSE-TokTail, 340
 - calls action, 340
 - calls advance-token, 340
 - calls current-symbol, 340
 - calls optional, 340
 - calls pop-stack-1, 340
 - calls push-reduction, 340
 - defun, 340
- PARSE-TokenCommandTail, 311
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-TokenCommandTail, 312
 - calls PARSE-TokenCommandTail, 312
 - calls PARSE-TokenOption, 312
 - calls action, 312
 - calls atEndOfLine, 312
 - calls bang, 311
 - calls optional, 312
 - calls pop-stack-1, 312
 - calls pop-stack-2, 312
 - calls push-reduction, 312
 - calls star, 312
 - calls systemCommand[5], 312
 - defun, 311
- PARSE-TokenList, 312
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-TokenOption, 312
 - calls action, 312
 - calls advance-token, 312
 - calls current-symbol, 312
 - calls isTokenDelimiter, 312
 - calls push-reduction, 312
 - calls star, 312
 - defun, 312
- PARSE-TokenOption, 312
 - calledby PARSE-TokenCommandTail, 312
 - calls PARSE-TokenList, 312
 - calls match-advance-string, 312
 - calls must, 312

- defun, 312
- PARSE-TokTail, 322
 - calledby PARSE-Infix, 321
 - calledby PARSE-Prefix, 321
 - calledby PARSE-PrimaryNoFloat, 326
 - calledby PARSE-Suffix, 340
 - calls PARSE-Qualification, 322
 - calls action, 322
 - calls char-eq, 322
 - calls copy-token, 322
 - calls current-char, 322
 - calls current-symbol, 322
 - uses \$boot, 322
 - defun, 322
- PARSE-VarForm, 332
 - calledby PARSE-Primary1, 326
 - calls PARSE-Name, 332
 - calls PARSE-Scripts, 332
 - calls optional, 332
 - calls pop-stack-1, 332
 - calls pop-stack-2, 332
 - calls push-reduction, 332
 - defun, 332
- PARSE-With, 315
 - calledby PARSE-InfixWith, 315
 - calls match-advance-string, 315
 - calls must, 315
 - calls pop-stack-1, 315
 - calls push-reduction, 315
 - defun, 315
- parseAnd, 105
 - calledby parseAnd, 105
 - calls parseAnd, 105
 - calls parseIf, 105
 - calls parseTranList, 105
 - calls parseTran, 105
 - uses \$InteractiveMode, 105
 - defun, 105
- parseAtom, 102
 - calledby parseTran, 101
 - calls parseLeave, 102
 - uses \$NoValue, 102
 - defun, 102
- parseAtSign, 106
 - calls parseTran, 106
 - calls parseType, 106
 - uses \$InteractiveMode, 106
 - defun, 106
- parseCategory, 107
 - calls contained, 107
 - calls parseDropAssertions, 107
 - calls parseTranList, 107
 - defun, 107
- parseCoerce, 108
 - calls parseTran, 108
 - calls parseType, 108
 - uses \$InteractiveMode, 108
 - defun, 108
- parseColon, 108
 - calls parseTran, 108
 - calls parseType, 108
 - uses \$InteractiveMode, 108
 - uses \$insideConstructIfTrue, 108
 - defun, 108
- parseConstruct, 103
 - calledby parseTran, 101
 - calls parseTranList, 103
 - uses \$insideConstructIfTrue, 103
 - defun, 103
- parseDEF, 109
 - calls opFf, 109
 - calls parseLhs, 109
 - calls parseTranCheckForRecord, 109
 - calls parseTranList, 109
 - calls setDefOp, 109
 - uses \$lhs, 109
 - defun, 109
- parseDollarGreaterEqual, 112
 - calls msubst, 112
 - calls parseTran, 112
 - uses \$op, 112
 - defun, 112
- parseDollarGreaterThan, 112
 - calls msubst, 112
 - calls parseTran, 112
 - uses \$op, 112
 - defun, 112
- parseDollarLessEqual, 113
 - calls msubst, 113
 - calls parseTran, 113
 - uses \$op, 113
 - defun, 113

- parseDollarNotEqual, 113
 - calls msubst, 114
 - calls parseTran, 113
 - uses \$op, 114
 - defun, 113
- parseDropAssertions, 107
 - calledby parseCategory, 107
 - calledby parseDropAssertions, 107
 - calls parseDropAssertions, 107
 - defun, 107
- parseEquivalence, 114
 - calls parseIf, 114
 - defun, 114
- parseExit, 115
 - calls moan, 115
 - calls parseTran, 115
 - defun, 115
- parseGreaterEqual, 115
 - calls parseTran, 115
 - uses \$op, 115
 - defun, 115
- parseGreaterThan, 116
 - calls parseTran, 116
 - uses \$op, 116
 - defun, 116
- parseHas, 116
 - calls getdatabase, 116
 - calls makeNonAtomic, 117
 - calls member, 117
 - calls nreverse0, 117
 - calls opOf, 116
 - calls parseHasRhs, 117
 - calls parseType, 117
 - calls qcar, 116
 - calls qcdr, 116
 - calls unabbrevAndLoad, 116
 - uses \$CategoryFrame, 117
 - uses \$InteractiveMode, 117
 - defun, 116
- parseHasRhs, 118
 - calledby parseHas, 117
 - calls abbreviation?, 118
 - calls get, 118
 - calls loadIfNecessary, 118
 - calls member, 118
 - calls qcar, 118
 - calls qcdr, 118
 - calls unabbrevAndLoad, 118
 - uses \$CategoryFrame, 118
 - defun, 118
- parseIf, 122
 - calledby parseAnd, 105
 - calledby parseEquivalence, 114
 - calledby parseImplies, 124
 - calledby parseOr, 134
 - calls parseIf,ifTran, 122
 - calls parseTran, 122
 - defun, 122
- parseIf,ifTran, 122
 - calledby parseIf,ifTran, 122
 - calledby parseIf, 122
 - calls incExitLevel, 122
 - calls makeSimplePredicateOrNil, 122
 - calls parseIf,ifTran, 122
 - calls parseTran, 122
 - uses \$InteractiveMode, 122
 - defun, 122
- parseImplies, 124
 - calls parseIf, 124
 - defun, 124
- parseIn, 125
 - calledby parseInBy, 126
 - calls parseTran, 125
 - calls postError, 125
 - defun, 125
- parseInBy, 126
 - calls bright, 126
 - calls parseIn, 126
 - calls parseTran, 126
 - calls postError, 126
 - defun, 126
- parseIs, 127
 - calls parseTran, 127
 - calls transIs, 127
 - defun, 127
- parseIsnt, 128
 - calls parseTran, 128
 - calls transIs, 128
 - defun, 128
- parseJoin, 128
 - calls parseTranList, 128
 - defun, 128

- parseLeave, 129
 - calledby parseAtom, 102
 - calls parseTran, 129
 - defun, 129
- parseLessEqual, 130
 - calls parseTran, 130
 - uses \$op, 130
 - defun, 130
- parseLET, 130
 - calls opOf, 130
 - calls parseTranCheckForRecord, 130
 - calls parseTran, 130
 - calls transIs, 130
 - defun, 130
- parseLETD, 131
 - calls parseTran, 131
 - calls parseType, 131
 - defun, 131
- parseLhs, 110
 - calledby parseDEF, 109
 - calls parseTran, 110
 - calls transIs, 110
 - defun, 110
- parseMDEF, 131
 - calls opOf, 131
 - calls parseTranCheckForRecord, 131
 - calls parseTranList, 131
 - calls parseTran, 131
 - uses \$lhs, 131
 - defun, 131
- ParseMode, 309
 - usedby PARSE-Expression, 317
 - usedby PARSE-Operation, 319
 - defvar, 309
- parseNot, 132
 - calls parseTran, 132
 - uses \$InteractiveMode, 132
 - defun, 132
- parseNotEqual, 133
 - calls msubst, 133
 - calls parseTran, 133
 - uses \$op, 133
 - defun, 133
- parseOr, 133
 - calledby parseOr, 134
 - calls parseIf, 134
 - calls parseOr, 134
 - calls parseTranList, 134
 - calls parseTran, 133
 - defun, 133
- parsepiles, 84
 - calledby preparse1, 81
 - calls add-parens-and-semis-to-line, 84
 - defun, 84
- parsePretend, 134
 - calls parseTran, 134
 - calls parseType, 134
 - defun, 134
- parseprint, 374
 - calledby preparse, 76
 - defun, 374
- parseReturn, 135
 - calls moan, 135
 - calls parseTran, 135
 - defun, 135
- parseSegment, 136
 - calls parseTran, 136
 - defun, 136
- parseSeq, 136
 - calls last, 136
 - calls mapInto, 136
 - calls postError, 136
 - calls transSeq, 136
 - defun, 136
- parseTran, 101
 - calledby compReduce1, 238
 - calledby parseAnd, 105
 - calledby parseAtSign, 106
 - calledby parseCoerce, 108
 - calledby parseColon, 108
 - calledby parseDollarGreaterEqual, 112
 - calledby parseDollarGreaterThan, 112
 - calledby parseDollarLessEqual, 113
 - calledby parseDollarNotEqual, 113
 - calledby parseExit, 115
 - calledby parseGreaterEqual, 115
 - calledby parseGreaterThan, 116
 - calledby parseIf,ifTran, 122
 - calledby parseIf, 122
 - calledby parseInBy, 126
 - calledby parseIn, 125
 - calledby parseIsnt, 128

- calledby parseIs, 127
- calledby parseLETD, 131
- calledby parseLET, 130
- calledby parseLeave, 129
- calledby parseLessEqual, 130
- calledby parseLhs, 110
- calledby parseMDEF, 131
- calledby parseNotEqual, 133
- calledby parseNot, 132
- calledby parseOr, 133
- calledby parsePretend, 134
- calledby parseReturn, 135
- calledby parseSegment, 136
- calledby parseTranCheckForRecord, 363
- calledby parseTranList, 103
- calledby parseTransform, 101
- calledby parseTran, 101
- calledby parseType, 106
- calls getl, 101
- calls parseAtom, 101
- calls parseConstruct, 101
- calls parseTranList, 101
- calls parseTran, 101
- uses \$op, 101
- defun, 101
- parseTranCheckForRecord, 363
 - calledby parseDEF, 109
 - calledby parseLET, 130
 - calledby parseMDEF, 131
 - calls parseTran, 363
 - calls postError, 363
 - calls qcar, 363
 - calls qcdr, 363
 - defun, 363
- parseTranList, 103
 - calledby parseAnd, 105
 - calledby parseCategory, 107
 - calledby parseConstruct, 103
 - calledby parseDEF, 109
 - calledby parseJoin, 128
 - calledby parseMDEF, 131
 - calledby parseOr, 134
 - calledby parseTranList, 103
 - calledby parseTran, 101
 - calledby parseVCONS, 137
 - calls parseTranList, 103
- calls parseTran, 103
 - defun, 103
- parseTransform, 101
 - calledby s-process, 396
 - calls msubst, 101
 - calls parseTran, 101
 - uses \$defOp, 101
 - defun, 101
- parseType, 106
 - calledby parseAtSign, 106
 - calledby parseCoerce, 108
 - calledby parseColon, 108
 - calledby parseHas, 117
 - calledby parseLETD, 131
 - calledby parsePretend, 134
 - calls msubst, 106
 - calls parseTran, 106
 - defun, 106
- parseVCONS, 137
 - calls parseTranList, 137
 - defun, 137
- parseWhere, 137
 - calls mapInto, 137
 - defun, 137
- pathname[5]
 - called by compileSpad2Cmd, 382
 - called by compileSpadLispCmd, 431
 - called by compiler, 379
- pathnameDirectory[5]
 - called by compileSpadLispCmd, 432
- pathnameName[5]
 - called by compileSpadLispCmd, 432
- pathnameType[5]
 - called by compileSpad2Cmd, 382
 - called by compileSpadLispCmd, 431
 - called by compiler, 379
- pathnameTypeId
 - calledby initializeLisplib, 159
- pname
 - calledby compDefineFunctor1, 167
 - calledby mkCategoryPackage, 146
- pname[5]
 - called by comp3, 406
 - called by floatexpid, 357
 - called by getScriptName, 297
- Pop-Reduction, 370

- calledby pop-stack-1, 368
- calledby pop-stack-2, 369
- calledby pop-stack-3, 369
- calledby pop-stack-4, 369
- calls stack-pop, 370
- defun, 370
- pop-stack-1, 368
 - calledby PARSE-Application, 324
 - calledby PARSE-Category, 316
 - calledby PARSE-CommandTail, 313
 - calledby PARSE-Conditional, 343
 - calledby PARSE-Data, 334
 - calledby PARSE-Enclosure, 330
 - calledby PARSE-Exit, 342
 - calledby PARSE-Expression, 317
 - calledby PARSE-Expr, 318
 - calledby PARSE-FloatTok, 345
 - calledby PARSE-Float, 327
 - calledby PARSE-Form, 324
 - calledby PARSE-Import, 317
 - calledby PARSE-InfixWith, 315
 - calledby PARSE-Infix, 321
 - calledby PARSE-Iterator, 339
 - calledby PARSE-LabelExpr, 344
 - calledby PARSE-Leave, 342
 - calledby PARSE-LedPart, 318
 - calledby PARSE-Loop, 344
 - calledby PARSE-Name, 333
 - calledby PARSE-NudPart, 319
 - calledby PARSE-Prefix, 321
 - calledby PARSE-Primary1, 326
 - calledby PARSE-Qualification, 322
 - calledby PARSE-Reduction, 323
 - calledby PARSE-Return, 341
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-Seg, 343
 - calledby PARSE-Selector, 325
 - calledby PARSE-SemiColon, 341
 - calledby PARSE-Sequence1, 337
 - calledby PARSE-Sequence, 337
 - calledby PARSE-Sexpr1, 335
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-Statement, 314
 - calledby PARSE-Suffix, 340
 - calledby PARSE-TokenCommandTail, 312
 - calledby PARSE-VarForm, 332
- calledby PARSE-With, 315
- calledby spad, 395
- calledby star, 359
- calls Pop-Reduction, 368
- calls reduction-value, 368
- defmacro, 368
- pop-stack-2, 369
 - calledby PARSE-Application, 324
 - calledby PARSE-Category, 315
 - calledby PARSE-CommandTail, 313
 - calledby PARSE-Conditional, 343
 - calledby PARSE-Float, 327
 - calledby PARSE-Import, 317
 - calledby PARSE-InfixWith, 315
 - calledby PARSE-Infix, 321
 - calledby PARSE-Iterator, 339
 - calledby PARSE-LabelExpr, 344
 - calledby PARSE-Loop, 344
 - calledby PARSE-Prefix, 321
 - calledby PARSE-Primary1, 326
 - calledby PARSE-Reduction, 323
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-Seg, 343
 - calledby PARSE-Selector, 325
 - calledby PARSE-SemiColon, 341
 - calledby PARSE-Sequence1, 337
 - calledby PARSE-Sexpr1, 334
 - calledby PARSE-Statement, 314
 - calledby PARSE-TokenCommandTail, 312
 - calledby PARSE-VarForm, 332
 - calls Pop-Reduction, 369
 - calls reduction-value, 369
 - calls stack-push, 369
 - defmacro, 369
- pop-stack-3, 369
 - calledby PARSE-Category, 315
 - calledby PARSE-Conditional, 343
 - calledby PARSE-Float, 327
 - calledby PARSE-Iterator, 339
 - calls Pop-Reduction, 369
 - calls reduction-value, 369
 - calls stack-push, 369
 - defmacro, 369
- pop-stack-4, 369
 - calledby PARSE-Float, 327
 - calls Pop-Reduction, 369

- calls reduction-value, 369
- calls stack-push, 369
- defmacro, 369
- postAdd, 264
 - calls postCapsule, 264
 - calls postTran, 264
 - defun, 264
- postAtom, 259
 - calledby postTran, 258
 - uses \$boot, 259
 - defun, 259
- postAtSign, 267
 - calls postTran, 267
 - calls postType, 267
 - defun, 267
- postBigFloat, 268
 - calls postTran, 268
 - uses \$InteractiveMode, 268
 - uses \$boot, 268
 - defun, 268
- postBlock, 268
 - calledby postSemiColon, 287
 - calls postBlockItemList, 268
 - calls postTran, 268
 - defun, 268
- postBlockItem, 266
 - calledby postBlockItemList, 265
 - calledby postCapsule, 265
 - calls postTran, 266
 - defun, 266
- postBlockItemList, 265
 - calledby postBlock, 268
 - calledby postCapsule, 265
 - calls postBlockItem, 265
 - defun, 265
- postCapsule, 265
 - calledby postAdd, 264
 - calls checkWarning, 265
 - calls postBlockItemList, 265
 - calls postBlockItem, 265
 - calls postFlatten, 265
 - defun, 265
- postCategory, 269
 - calls nreverse0, 269
 - calls postTran, 269
 - uses \$insidePostCategoryIfTrue, 269
- defun, 269
- postcheck, 261
 - calledby postTransformCheck, 261
 - calledby postcheck, 261
 - calls postcheck, 261
 - calls setDefOp, 261
 - defun, 261
- postCollect, 271
 - calledby postCollect, 271
 - calledby postTupleCollect, 291
 - calls postCollect,finish, 271
 - calls postCollect, 271
 - calls postIteratorList, 271
 - calls postTran, 271
 - defun, 271
- postCollect,finish, 270
 - calledby postCollect, 271
 - calls postMakeCons, 270
 - calls postTranList, 270
 - calls qcar, 270
 - calls qcdr, 270
 - calls tuple2List, 270
 - defun, 270
- postColon, 273
 - calls postTran, 273
 - calls postType, 273
 - defun, 273
- postColonColon, 273
 - calls postForm, 273
 - uses \$boot, 273
 - defun, 273
- postComma, 274
 - calls comma2Tuple, 274
 - calls postTuple, 274
 - defun, 274
- postConstruct, 275
 - calls comma2Tuple, 275
 - calls postMakeCons, 275
 - calls postTranList, 275
 - calls postTranSegment, 275
 - calls postTran, 275
 - calls tuple2List, 275
 - defun, 275
- postDef, 276
 - calls nequal, 277
 - calls nreverse0, 277

- calls postDefArgs, 277
- calls postMDef, 276
- calls postTran, 277
- calls recordHeaderDocumentation, 277
- uses \$InteractiveMode, 277
- uses \$boot, 277
- uses \$docList, 277
- uses \$headerDocumentation, 277
- uses \$maxSignatureLineNumber, 277
- defun, 276
- postDefArgs, 278
 - calledby postDefArgs, 278
 - calledby postDef, 277
 - calls postDefArgs, 278
 - calls postError, 278
 - defun, 278
- postError, 262
 - calledby checkWarning, 367
 - calledby getScriptName, 297
 - calledby parseInBy, 126
 - calledby parseIn, 125
 - calledby parseSeq, 136
 - calledby parseTranCheckForRecord, 363
 - calledby postDefArgs, 278
 - calledby postForm, 262
 - calls bumperrorcount, 262
 - calls nequal, 262
 - uses \$InteractiveMode, 262
 - uses \$defOp, 262
 - uses \$postStack, 262
 - defun, 262
- postExit, 279
 - calls postTran, 279
 - defun, 279
- postFlatten, 274
 - calledby comma2Tuple, 274
 - calledby postCapsule, 265
 - calledby postFlatten, 274
 - calls postFlatten, 274
 - defun, 274
- postFlattenLeft, 287
 - calledby postFlattenLeft, 287
 - calledby postSemiColon, 287
 - calls postFlattenLeft, 287
 - defun, 287
- postForm, 262
 - calledby postColonColon, 273
 - calledby postTran, 258
 - calls bright, 262
 - calls internl, 262
 - calls postError, 262
 - calls postTranList, 262
 - calls postTran, 262
 - uses \$boot, 262
 - defun, 262
- postIf, 279
 - calls nreverse0, 279
 - calls postTran, 279
 - uses \$boot, 279
 - defun, 279
- postIn, 281
 - calls postInSeq, 281
 - calls postTran, 281
 - calls systemErrorHere, 281
 - defun, 281
- postin, 280
 - calls postInSeq, 280
 - calls postTran, 280
 - calls systemErrorHere, 280
 - defun, 280
- postInSeq, 280
 - calledby postIn, 281
 - calledby postIteratorList, 272
 - calledby postin, 280
 - calls postTranSegment, 280
 - calls postTran, 280
 - calls tuple2List, 280
 - defun, 280
- postIteratorList, 272
 - calledby postCollect, 271
 - calledby postIteratorList, 272
 - calledby postRepeat, 286
 - calls postInSeq, 272
 - calls postIteratorList, 272
 - calls postTran, 272
 - defun, 272
- postJoin, 282
 - calls postTranList, 282
 - calls postTran, 282
 - defun, 282
- postMakeCons, 270
 - calledby postCollect, finish, 270

- calledby postConstruct, 275
 - calledby postMakeCons, 270
 - calls postMakeCons, 270
 - calls postTran, 270
 - defun, 270
- postMapping, 282
 - calls postTran, 282
 - calls unTuple, 282
 - defun, 282
- postMDef, 283
 - calledby postDef, 276
 - calls nreverse0, 283
 - calls postTran, 283
 - calls throwkeyedmsg, 283
 - uses \$InteractiveMode, 283
 - uses \$boot, 283
 - defun, 283
- postOp, 259
 - calledby postTran, 258
 - defun, 259
- postPretend, 284
 - calls postTran, 284
 - calls postType, 284
 - defun, 284
- postQUOTE, 285
 - defun, 285
- postReduce, 285
 - calledby postReduce, 285
 - calls postReduce, 285
 - calls postTran, 285
 - uses \$InteractiveMode, 285
 - defun, 285
- postRepeat, 286
 - calls postIteratorList, 286
 - calls postTran, 286
 - defun, 286
- postScripts, 287
 - calls getScriptName, 287
 - calls postTranScripts, 287
 - defun, 287
- postScriptsForm, 260
 - calledby postTran, 258
 - calls getScriptName, 260
 - calls length, 260
 - calls postTranScripts, 260
 - defun, 260
- postSemiColon, 287
 - calls postBlock, 287
 - calls postFlattenLeft, 287
 - defun, 287
- postSignature, 288
 - calls killColons, 288
 - calls pairp, 288
 - calls postType, 288
 - calls removeSuperfluousMapping, 288
 - defun, 288
- postSlash, 289
 - calls postTran, 289
 - defun, 289
- postTran, 258
 - calledby postAdd, 264
 - calledby postAtSign, 267
 - calledby postBigFloat, 268
 - calledby postBlockItem, 266
 - calledby postBlock, 268
 - calledby postCategory, 269
 - calledby postCollect, 271
 - calledby postColon, 273
 - calledby postConstruct, 275
 - calledby postDef, 277
 - calledby postExit, 279
 - calledby postForm, 262
 - calledby postIf, 279
 - calledby postInSeq, 280
 - calledby postIn, 281
 - calledby postIteratorList, 272
 - calledby postJoin, 282
 - calledby postMDef, 283
 - calledby postMakeCons, 270
 - calledby postMapping, 282
 - calledby postPretend, 284
 - calledby postReduce, 285
 - calledby postRepeat, 286
 - calledby postSlash, 289
 - calledby postTranList, 260
 - calledby postTranScripts, 260
 - calledby postTranSegment, 276
 - calledby postTransform, 257
 - calledby postTran, 258
 - calledby postType, 267
 - calledby postWhere, 291
 - calledby postWith, 292

- calledby postin, 280
- calledby tuple2List, 368
- calls pairp, 258
- calls postAtom, 258
- calls postForm, 258
- calls postOp, 258
- calls postScriptsForm, 258
- calls postTranList, 258
- calls postTran, 258
- calls qcar, 258
- calls qcdr, 258
- calls unTuple, 258
- defun, 258
- postTranList, 260
 - calledby postCollect,finish, 270
 - calledby postConstruct, 275
 - calledby postForm, 262
 - calledby postJoin, 282
 - calledby postTran, 258
 - calledby postTuple, 290
 - calledby postWhere, 291
 - calls postTran, 260
 - defun, 260
- postTranScripts, 260
 - calledby postScriptsForm, 260
 - calledby postScripts, 287
 - calledby postTranScripts, 260
 - calls postTranScripts, 260
 - calls postTran, 260
 - defun, 260
- postTranSegment, 276
 - calledby postConstruct, 275
 - calledby postInSeq, 280
 - calledby tuple2List, 368
 - calls postTran, 276
 - defun, 276
- postTransform, 257
 - calledby new2OldLisp, 364
 - calledby s-process, 396
 - calls aplTran, 257
 - calls identp[5], 257
 - calls postTransformCheck, 257
 - calls postTran, 257
 - defun, 257
- postTransformCheck, 261
 - calledby postTransform, 257
 - calls postcheck, 261
 - uses \$defOp, 261
 - defun, 261
- postTuple, 290
 - calledby postComma, 274
 - calls postTranList, 290
 - defun, 290
- postTupleCollect, 291
 - calls postCollect, 291
 - defun, 291
- postType, 267
 - calledby postAtSign, 267
 - calledby postColon, 273
 - calledby postPretend, 284
 - calledby postSignature, 288
 - calls postTran, 267
 - calls unTuple, 267
 - defun, 267
- postWhere, 291
 - calls postTranList, 291
 - calls postTran, 291
 - defun, 291
- postWith, 292
 - calls postTran, 292
 - uses \$insidePostCategoryIfTrue, 292
 - defun, 292
- pp
 - calledby compDefineFunctor1, 167
- PredImplies
 - calledby compForm2, 416
- preparse, 71, 76
 - calledby preparse, 76
 - calledby spad, 395
 - calls ifcar, 76
 - calls parseprint, 76
 - calls preparse1, 76
 - calls preparse, 76
 - uses \$comblocklist, 77
 - uses \$constructorLineNumber, 77
 - uses \$docList, 77
 - uses \$headerDocumentation, 77
 - uses \$index, 77
 - uses \$maxSignatureLineNumber, 77
 - uses \$preparse-last-line, 77
 - uses \$preparseReportIfTrue, 77
 - uses \$skipme, 77

- defun, 76
- preparse-echo, 88
 - calledby fincomblock, 372
 - calledby preparse1, 81
 - uses Echo-Meta, 88
 - uses \$EchoLineStack, 88
 - defun, 88
- preparse1, 81
 - calledby preparse, 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 preparse-echo, 81
 - calls preparseReadLine, 81
 - calls strposl[5], 81
 - uses \$byConstructors, 81
 - uses \$constructorsSeen, 81
 - uses \$echolinestack, 81
 - uses \$linelist, 81
 - uses \$preparse-last-line, 81
 - uses \$skipme, 81
 - catches, 81
 - defun, 81
- preparseReadLine, 85
 - calledby preparse1, 81
 - calledby preparseReadLine, 85
 - calledby skip-to-endif, 374
 - calls dcq, 85
 - calls initial-substring, 85
 - calls preparseReadLine1, 85
 - calls preparseReadLine, 85
 - calls skip-to-endif, 85
 - calls storeblanks, 85
 - calls string2BootTree, 85
 - defun, 85
- preparseReadLine1, 87
 - calledby preparseReadLine1, 87
 - calledby preparseReadLine, 85
 - calledby skip-ifblock, 86
 - calledby skip-to-endif, 374
 - calls expand-tabs, 87
 - calls get-a-line, 87
 - calls maxindex, 87
 - calls preparseReadLine1, 87
 - calls strconc, 87
 - uses \$EchoLineStack, 87
 - uses \$index, 87
 - uses \$linelist, 87
 - uses \$preparse-last-line, 87
 - defun, 87
- pretend, 134, 236, 284
 - defplist, 134, 236, 284
- prettyprint
 - calledby s-process, 396
- primitiveType, 411
 - calledby compAtom, 409
 - uses \$DoubleFloat, 411
 - uses \$EmptyMode, 411
 - uses \$NegativeInteger, 411
 - uses \$NonNegativeInteger, 411
 - uses \$PositiveInteger, 411
 - uses \$String, 411
 - defun, 411
- print-defun, 399
 - calls is-console, 399
 - calls print-full, 399
 - uses \$PrettyPrint, 399
 - uses vmlisp::optionlist, 399
 - defun, 399
- print-full
 - calledby print-defun, 399
- print-package, 367
 - defun, 367
- prior-token, 97
 - usedby PARSE-Expression, 317
 - uses \$token, 97
 - defvar, 97
- processFunctorOrPackage
 - calledby compCapsuleInner, 208
- processInteractive[5]
 - called by s-process, 396
- processSynonyms[5]
 - called by PARSE-NewExpr, 309
- profileRecord
 - calledby setqSingle, 247
- profileWrite
 - calledby finalizeLisplib, 160

- push-reduction, 360
 - calledby PARSE-AnyId, 336
 - calledby PARSE-Application, 324
 - calledby PARSE-Category, 315
 - calledby PARSE-CommandTail, 313
 - calledby PARSE-Command, 310
 - calledby PARSE-Conditional, 343
 - calledby PARSE-Data, 334
 - calledby PARSE-Enclosure, 330
 - calledby PARSE-Exit, 341
 - calledby PARSE-Expression, 317
 - calledby PARSE-Expr, 318
 - calledby PARSE-FloatBasePart, 329
 - calledby PARSE-FloatBase, 328
 - calledby PARSE-FloatExponent, 329
 - calledby PARSE-FloatTok, 345
 - calledby PARSE-Float, 327
 - calledby PARSE-Form, 323
 - calledby PARSE-Import, 317
 - calledby PARSE-InfixWith, 315
 - calledby PARSE-Infix, 321
 - calledby PARSE-LabelExpr, 344
 - calledby PARSE-Leave, 342
 - calledby PARSE-LedPart, 318
 - calledby PARSE-Loop, 344
 - calledby PARSE-Name, 333
 - calledby PARSE-NudPart, 318
 - calledby PARSE-OpenBrace, 338
 - calledby PARSE-OpenBracket, 338
 - calledby PARSE-Prefix, 320, 321
 - calledby PARSE-Primary1, 326
 - calledby PARSE-PrimaryOrQM, 313
 - calledby PARSE-Quad, 331
 - calledby PARSE-Qualification, 322
 - calledby PARSE-Reduction, 323
 - calledby PARSE-Return, 341
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-Seg, 342
 - calledby PARSE-Selector, 325
 - calledby PARSE-SemiColon, 341
 - calledby PARSE-Sequence1, 337
 - calledby PARSE-Sequence, 337
 - calledby PARSE-Sexpr1, 334
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-Statement, 314
 - calledby PARSE-Suffix, 340
 - calledby PARSE-TokenCommandTail, 312
 - calledby PARSE-TokenList, 312
 - calledby PARSE-VarForm, 332
 - calledby PARSE-With, 315
 - calledby parse-argument-designator, 367
 - calledby parse-identifier, 365
 - calledby parse-keyword, 366
 - calledby parse-number, 366
 - calledby parse-spadstring, 364
 - calledby parse-string, 365
 - calledby star, 359
 - calls make-reduction, 360
 - calls stack-push, 360
 - uses reduce-stack, 360
 - defun, 360
- put
 - calledby compColon, 216
 - calledby compMacro, 235
 - calledby compSubsetCategory, 252
 - calledby compSuchthat, 253
 - calledby compTypeOf, 408
 - calledby evalAndSub, 201
 - calledby giveFormalParametersValues, 143
 - calledby putDomainsInScope, 189
 - calledby updateCategoryFrameForCategory, 121
 - calledby updateCategoryFrameForConstructor, 120
- putDomainsInScope, 189
 - calledby addConstructorModemaps, 192
 - calledby augModemapsFromCategoryRep, 203
 - calledby augModemapsFromCategory, 195
 - calls delete, 189
 - calls getDomainsInScope, 189
 - calls member, 189
 - calls put, 189
 - calls say, 189
 - local def \$CapsuleDomainsInScope, 189
 - local ref \$insideCapsuleFunctionIfTrue, 189
 - defun, 189
- qcar
 - calledby addConstructorModemaps, 192
 - calledby addEltModemap, 197

- calledby addModemap0, 196
- calledby compAdd, 205
- calledby compCategory, 212
- calledby compCons1, 219
- calledby compDefWhereClause, 184
- calledby compDefineFunctor1, 167
- calledby compJoin, 231
- calledby compLambda, 233
- calledby compMacro, 235
- calledby compSetq1, 246
- calledby compWithMappingModel, 422
- calledby decodeScripts, 297
- calledby genDomainView, 181
- calledby getModemapList, 194
- calledby isDomainConstructorForm, 249
- calledby isDomainForm, 248
- calledby makeFunctorArgumentParameters, 178
- calledby mkEvalableCategoryForm, 148
- calledby mkNewModemapList, 198
- calledby mkOpVec, 183
- calledby parseHasRhs, 118
- calledby parseHas, 116
- calledby parseTranCheckForRecord, 363
- calledby postCollect,finish, 270
- calledby postTran, 258
- calledby transIs1, 110
- qcdr
 - calledby addConstructorModemaps, 192
 - calledby addEltModemap, 197
 - calledby compAdd, 205
 - calledby compCategory, 212
 - calledby compCons1, 219
 - calledby compDefWhereClause, 184
 - calledby compDefineFunctor1, 167
 - calledby compJoin, 231
 - calledby compLambda, 233
 - calledby compSetq1, 246
 - calledby compWithMappingModel, 422
 - calledby decodeScripts, 297
 - calledby genDomainViewList, 181
 - calledby genDomainView, 181
 - calledby getModemapList, 194
 - calledby isDomainConstructorForm, 249
 - calledby isDomainForm, 249
- calledby makeFunctorArgumentParameters, 178
- calledby mkEvalableCategoryForm, 148
- calledby mkNewModemapList, 198
- calledby mkOpVec, 183
- calledby parseHasRhs, 118
- calledby parseHas, 116
- calledby parseTranCheckForRecord, 363
- calledby postCollect,finish, 270
- calledby postTran, 258
- calledby transIs1, 110
- qslessp
 - calledby addDomain, 186
 - calledby getUniqueModemap, 194
- quote, 237, 285
 - defplist, 237, 285
- quote-if-string, 348
 - calledby match-advance-string, 347
 - calledby unget-tokens, 350
- calls escape-keywords, 348
- calls pack, 348
- calls strconc, 348
- calls token-nonblank, 348
- calls token-symbol, 348
- calls token-type, 348
- calls underscore, 348
- uses \$boot, 348
- uses \$spad, 348
- defun, 348
- quotify
 - calledby compIf, 229
- quotifyCategoryArgument
 - calledby mkEvalableCategoryForm, 148
- rbp
 - usedby PARSE-NudPart, 319
 - usedby PARSE-Operation, 319
- rdefiostream
 - calledby compileDocumentation, 158
 - calledby writeLib1, 160
- read-a-line, 88
 - calledby get-a-line, 94
 - calledby read-a-line, 88
 - calls Line-New-Line, 88
 - calls read-a-line, 88
 - calls subseq, 88

- uses *eof*, 88
- defun, 88
- recompile-lib-file-if-necessary, 433
 - calledby compileSpadLispCmd, 432
 - calls compile-lib-file, 433
 - uses *lisp-bin-filetype*, 433
- defun, 433
- Record, 211
 - defplist, 211
- recordAttributeDocumentation
 - calledby PARSE-Category, 316
- RecordCategory, 221
 - defplist, 221
- recordHeaderDocumentation
 - calledby postDef, 277
- recordSignatureDocumentation
 - calledby PARSE-Category, 316
- reduce, 237, 285
 - defplist, 237, 285
- reduce-stack, 360
 - usedby push-reduction, 360
 - uses \$stack, 360
 - defvar, 360
- reduce-stack-clear, 360
 - defmacro, 360
- reduction, 98
 - defstruct, 98
- reduction-value
 - calledby nth-stack, 370
 - calledby pop-stack-1, 368
 - calledby pop-stack-2, 369
 - calledby pop-stack-3, 369
 - calledby pop-stack-4, 369
- refvecp
 - calledby translabl1, 361
- remdup
 - calledby compDefineFunctor1, 167
 - calledby displayPreCompilationErrors, 362
- removeEnv
 - calledby setqSingle, 247
- removeSuperfluousMapping, 289
 - calledby postSignature, 288
- defun, 289
- removeZeroOne
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
- calledby finalizeLisplib, 160
- repeat, 240, 286
 - defplist, 240, 286
- replaceExitEtc
 - calledby compSeq1, 244
- replaceFile
 - calledby compileDocumentation, 158
 - calledby lisplibDoRename, 158
- reportOnFunctorCompilation, 177
 - calledby compDefineFunctor1, 167
 - calls addStats, 177
 - calls displayMissingFunctions, 177
 - calls displaySemanticErrors, 177
 - calls displayWarnings, 177
 - calls normalizeStatAndStringify, 177
 - calls sayBrightly, 177
 - uses \$functionStats, 177
 - uses \$functorStats, 177
 - uses \$op, 177
 - uses \$semanticErrorStack, 177
 - uses \$warningStack, 177
 - defun, 177
- resolve
 - calledby compCategory, 212
 - calledby compCoerce1, 214
 - calledby compConstructorCategory, 222
 - calledby compIf, 229
 - calledby compReturn, 242
 - calledby compString, 250
 - calledby convert, 411
 - calledby modifyModeStack, 429
- return, 135, 242
 - defplist, 135, 242
- rpacfile
 - calledby compDefineLisplib, 155
 - calledby compileDocumentation, 158
- rshut
 - calledby compDefineLisplib, 155
 - calledby compileDocumentation, 158
- rwrite128
 - calledby lisplibWrite, 166
- rwriteLispForm, 154
 - calledby evalAndRwriteLispForm, 154
 - local ref \$libFile, 154
 - local ref \$lisplib, 154
 - defun, 154

- s-process, 396
 - calledby spad, 395
 - calls compTopLevel, 396
 - calls curstrm, 396
 - calls def-process, 396
 - calls def-rename, 396
 - calls displayPreCompilationErrors, 396
 - calls displaySemanticErrors, 396
 - calls get-internal-run-time, 396
 - calls new2OldLisp, 396
 - calls parseTransform, 396
 - calls postTransform, 396
 - calls prettyprint, 396
 - calls processInteractive[5], 396
 - calls terpri, 396
 - uses \$DomainFrame, 397
 - uses \$EmptyEnvironment, 397
 - uses \$EmptyMode, 397
 - uses \$Index, 396
 - uses \$LocalFrame, 397
 - uses \$PolyMode, 397
 - uses \$Translation, 397
 - uses \$VariableCount, 397
 - uses \$compUniquelyIfTrue, 397
 - uses \$currentFunction, 397
 - uses \$exitModeStack, 397
 - uses \$exitMode, 397
 - uses \$e, 397
 - uses \$form, 397
 - uses \$genFVar, 397
 - uses \$genSDVar, 397
 - uses \$insideCapsuleFunctionIfTrue, 397
 - uses \$insideCategoryIfTrue, 397
 - uses \$insideCoerceInteractiveHardIfTrue, 397
 - uses \$insideExpressionIfTrue, 397
 - uses \$insideFunctorIfTrue, 397
 - uses \$insideWhereIfTrue, 397
 - uses \$leaveLevelStack, 397
 - uses \$leaveMode, 397
 - uses \$macroassoc, 397
 - uses \$newspad, 397
 - uses \$postStack, 397
 - uses \$previousTime, 397
 - uses \$returnMode, 397
 - uses \$semanticErrorStack, 397
 - uses \$stop-level, 397
 - uses \$stopOp, 397
 - uses \$warningStack, 397
 - uses curoutstream, 397
 - defun, 396
- say
 - calledby compOrCroak1, 402
 - calledby modifyModeStack, 429
 - calledby putDomainsInScope, 189
- sayBrightly
 - calledby compDefineFunctor1, 167
 - calledby compMacro, 235
 - calledby compilerDoit, 384
 - calledby displayMissingFunctions, 177
 - calledby displayPreCompilationErrors, 362
 - calledby reportOnFunctorCompilation, 177
- sayKeyedMsg[5]
 - called by compileSpad2Cmd, 382
 - called by compileSpadLispCmd, 432
- sayMath
 - calledby displayPreCompilationErrors, 362
- sayMSG
 - calledby compDefineLisplib, 155
 - calledby finalizeLisplib, 160
- ScanOrPairVec[5]
 - called by hasFormalMapVariable, 427
- Scripts, 286
 - defplist, 286
- segment, 135, 136
 - defplist, 135, 136
- selectOptionLC[5]
 - called by compileSpad2Cmd, 382
 - called by compileSpadLispCmd, 431
 - called by compiler, 379
- seq, 243
 - defplist, 243
- setDefOp, 292
 - calledby parseDEF, 109
 - calledby postcheck, 261
 - uses \$defOp, 292
 - uses \$stopOp, 292
 - defun, 292
- setelt
 - calledby modifyModeStack, 429
- setq, 245
 - defplist, 245

- setqMultiple
 - calledby compSetq1, 246
- setqSetelt, 246
 - calledby compSetq1, 246
 - calls comp, 246
 - defun, 246
- setqSingle, 247
 - calledby compSetq1, 246
 - calls NRTassocIndex, 247
 - calls addBinding[5], 247
 - calls assignError, 247
 - calls augModemapsFromDomain1, 247
 - calls comp, 247
 - calls consProplistOf, 247
 - calls convert, 247
 - calls getProplist[5], 247
 - calls getmode, 247
 - calls get, 247
 - calls identp[5], 247
 - calls isDomainForm, 247
 - calls isDomainInScope, 247
 - calls maxSuperType, 247
 - calls nequal, 247
 - calls outputComp, 247
 - calls profileRecord, 247
 - calls removeEnv, 247
 - calls stackWarning, 247
 - uses \$EmptyMode, 247
 - uses \$NoValueMode, 247
 - uses \$QuickLet, 247
 - uses \$form, 247
 - uses \$insideSetqSingleIfTrue, 247
 - uses \$profileCompiler, 247
 - defun, 247
- shut[5]
 - called by spad, 395
- Signature, 288
 - defplist, 288
- simpBool
 - calledby compDefineFunctor1, 167
- skip-blanks, 346
 - calledby match-string, 346
 - calls advance-char, 346
 - calls current-char, 346
 - calls token-lookahead-type, 346
 - defun, 346
- skip-ifblock, 86
 - calledby skip-ifblock, 86
 - calls initial-substring, 86
 - calls preparsedReadLine1, 86
 - calls skip-ifblock, 86
 - calls storeblanks, 86
 - calls string2BootTree, 86
 - defun, 86
- skip-to-endif, 374
 - calledby preparsedReadLine, 85
 - calledby skip-to-endif, 374
 - calls initial-substring, 374
 - calls preparsedReadLine1, 374
 - calls preparsedReadLine, 374
 - calls skip-to-endif, 374
 - defun, 374
- spad, 395
 - calls PARSE-NewExpr, 395
 - calls addBinding[5], 395
 - calls init-boot/spad-reader[5], 395
 - calls initialize-preparse, 395
 - calls ioclear, 395
 - calls makeInitialModemapFrame[5], 395
 - calls pop-stack-1, 395
 - calls preparse, 395
 - calls s-process, 395
 - calls shut[5], 395
 - uses *comp370-apply*, 395
 - uses *eof*, 395
 - uses /editfile, 395
 - uses \$InitialDomainsInScope, 395
 - uses \$InteractiveFrame, 395
 - uses \$InteractiveMode, 395
 - uses \$noSubsumption, 395
 - uses echo-meta, 395
 - uses file-closed, 395
 - uses line, 395
 - catches, 395
 - defun, 395
- spad-fixed-arg, 433
 - defun, 433
- spad2AsTranslatorAutoloadOnceTrigger
 - calledby compileSpad2Cmd, 382
- spad[5]
 - called by /rf-1, 386
- SpadInterpretStream[5]

- called by ncINTERPFILE, 431
- spadPrompt
 - calledby compileSpad2Cmd, 382
 - calledby compileSpadLispCmd, 432
- spadreduce
 - calledby floatexpid, 357
- stack, 94
 - defstruct, 94
- stack-/empty, 95
 - uses \$stack, 95
 - defmacro, 95
- stack-clear, 95
 - uses \$stack, 95
 - defun, 95
- stack-load, 95
 - uses \$stack, 95
 - defun, 95
- stack-pop, 96
 - calledby Pop-Reduction, 370
 - uses \$stack, 96
 - defun, 96
- stack-push, 96
 - calledby pop-stack-2, 369
 - calledby pop-stack-3, 369
 - calledby pop-stack-4, 369
 - calledby push-reduction, 360
 - uses \$stack, 96
 - defun, 96
- stack-size
 - calledby star, 359
- stack-store
 - calledby nth-stack, 370
- stackAndThrow
 - calledby compDefine1, 223
 - calledby compLambda, 233
 - calledby compWithMappingModel1, 422
- stackMessage
 - calledby augModemapsFromDomain1, 191
 - calledby compElt, 225
 - calledby compRepeatOrCollect, 240
 - calledby compSymbol, 412
 - calledby getOperationAlist, 201
- stackMessageIfNone
 - calledby compExit, 227
 - calledby compForm, 414
- stackSemanticError
 - calledby compColonInside, 408
 - calledby compJoin, 231
 - calledby compOrCroak1, 402
 - calledby compPretend, 236
 - calledby compReturn, 242
 - calledby compSubDomain1, 251
 - calledby getTargetFromRhs, 142
- stackWarning
 - calledby compColonInside, 408
 - calledby compElt, 225
 - calledby compPretend, 236
 - calledby getUniqueModemap, 194
 - calledby setqSingle, 247
- star, 359
 - calledby PARSE-Application, 324
 - calledby PARSE-Category, 316
 - calledby PARSE-CommandTail, 313
 - calledby PARSE-Expr, 318
 - calledby PARSE-Import, 317
 - calledby PARSE-IteratorTail, 339
 - calledby PARSE-Loop, 344
 - calledby PARSE-Option, 314
 - calledby PARSE-ScriptItem, 333
 - calledby PARSE-Sexpr1, 335
 - calledby PARSE-SpecialCommand, 311
 - calledby PARSE-Statement, 314
 - calledby PARSE-TokenCommandTail, 312
 - calledby PARSE-TokenList, 312
 - calls pop-stack-1, 359
 - calls push-reduction, 359
 - calls stack-size, 359
 - defmacro, 359
- step
 - calledby floatexpid, 357
- storeblanks, 93
 - calledby preparseReadLine, 85
 - calledby skip-ifblock, 86
 - defun, 93
- strconc
 - calledby compDefine1, 223
 - calledby compDefineFunctor1, 167
 - calledby compileSpad2Cmd, 382
 - calledby decodeScripts, 297
 - calledby mkCategoryPackage, 146
 - calledby preparseReadLine1, 87
 - calledby quote-if-string, 348

- calledby unget-tokens, 350
- String, 250
 - defplist, 250
- string-not-greaterp
 - calledby initial-substring-p, 348
- string2BootTree
 - calledby prepareReadLine, 85
 - calledby skip-ifblock, 86
- string2id-n
 - calledby infixtok, 373
- stringimage
 - calledby substituteCategoryArguments, 192
- stringPrefix?
 - calledby comp3, 406
- stripUnionTags
 - calledby augModemapsFromDomain, 190
- strposl[5]
 - called by prepare1, 81
- SubDomain, 250
 - defplist, 250
- sublis
 - calledby augLisplibModemapsFromCategory, 152
 - calledby compDefineCategory2, 149
 - calledby compDefineFunctor1, 167
 - calledby compForm2, 416
 - calledby getModemap, 193
 - calledby mkOpVec, 183
 - calledby substituteCategoryArguments, 192
- sublislis
 - calledby mkCategoryPackage, 147
- subseq
 - calledby match-string, 346
 - calledby read-a-line, 88
- SubsetCategory, 252
 - defplist, 252
- substituteCategoryArguments, 192
 - calledby augModemapsFromDomain1, 191
 - calls internl, 192
 - calls msubst, 192
 - calls stringimage, 192
 - calls sublis, 192
 - defun, 192
- substNames, 202
 - calledby evalAndSub, 201
 - calledby genDomainOps, 182
 - calls eqsubstlist, 202
 - calls isCategoryPackageName, 202
 - calls nreverse0, 202
 - calls substq, 202
 - uses \$FormalMapVariableList, 202
 - defun, 202
- substq
 - calledby substNames, 202
- suffix
 - calledby addclose, 370
- systemCommand[5]
 - called by PARSE-CommandTail, 313
 - called by PARSE-TokenCommandTail, 312
- systemError
 - calledby compReduce1, 238
 - calledby errhuh, 301
 - calledby getOperationAlist, 201
- systemErrorHere
 - calledby addEltModemap, 197
 - calledby compCategory, 212
 - calledby compColon, 216
 - calledby getSlotFromCategoryForm, 163
 - calledby getSlotFromFunctor, 165
 - calledby postIn, 281
 - calledby postin, 280
- take
 - calledby compColon, 216
 - calledby compDefineCategory2, 149
 - calledby compForm2, 416
 - calledby compWithMappingMode1, 422
 - calledby drop, 371
 - calledby getSlotFromCategoryForm, 163
- terminateSystemCommand[5]
 - called by compileSpad2Cmd, 382
 - called by compileSpadLispCmd, 431
- terpri
 - calledby s-process, 396
- throwKeyedMsg
 - calledby compileSpad2Cmd, 382
 - calledby compileSpadLispCmd, 432
 - calledby compiler, 379
 - calledby loadLibIfNecessary, 119
- throwkeyedmsg
 - calledby postMDef, 283
- tmptok, 308

- usedby PARSE-Operation, 319
 - defvar, 308
- tok, 308
 - usedby PARSE-GlyphTok, 336
 - usedby PARSE-NBGlyphTok, 335
 - defvar, 308
- token, 96
 - defstruct, 96
- token-install, 98
 - uses \$token, 98
 - defun, 98
- token-lookahead-type, 347
 - calledby skip-blanks, 346
 - uses Escape-Character, 347
 - defun, 347
- token-nonblank
 - calledby PARSE-FloatBasePart, 329
 - calledby quote-if-string, 348
 - calledby unget-tokens, 350
- token-print, 98
 - uses \$token, 98
 - defun, 98
- token-symbol
 - calledby PARSE-SpecialKeyWord, 310
 - calledby match-token, 352
 - calledby parse-argument-designator, 367
 - calledby parse-identifier, 365
 - calledby parse-keyword, 366
 - calledby parse-number, 366
 - calledby parse-spadstring, 364
 - calledby parse-string, 365
 - calledby quote-if-string, 348
- token-type
 - calledby match-token, 351
 - calledby quote-if-string, 348
- TPDHERE
 - See LocalAlgebra for an example call, 252
 - test with BASTYPE, 141
- transformOperationAlist, 163
 - calledby getCategoryOpsAndAtts, 162
 - calledby getFunctorOpsAndAtts, 165
 - calls assoc, 164
 - calls insertAlist, 164
 - calls keyedSystemError, 164
 - calls lassq, 164
 - calls member, 164
 - local ref \$functionLocations, 164
 - defun, 163
- transIs, 110
 - calledby parseIsnt, 128
 - calledby parseIs, 127
 - calledby parseLET, 130
 - calledby parseLhs, 110
 - calledby transIs1, 110
 - calls isListConstructor, 110
 - calls transIs1, 110
 - defun, 110
- transIs1, 110
 - calledby transIs, 110
 - calledby transIs, 110
 - calls nreverse0, 110
 - calls pairp, 110
 - calls qcar, 110
 - calls qcdr, 110
 - calls transIs1, 110
 - calls transIs, 110
 - defun, 110
- translabel, 361
 - calledby PARSE-Data, 334
 - calls translabel1, 361
 - defun, 361
- translabel1, 361
 - calledby translabel1, 361
 - calledby translabel, 361
 - calls lassoc, 361
 - calls maxindex, 361
 - calls refvecp, 361
 - calls translabel1, 361
 - defun, 361
- transSeq
 - calledby parseSeq, 136
- TruthP
 - calledby mergeModemap, 199
- try-get-token, 353
 - calledby advance-token, 354
 - calledby current-token, 353
 - calledby next-token, 354
 - calls get-token, 353
 - uses valid-tokens, 353
 - defun, 353
- tuple2List, 368

- calledby postCollect,finish, 270
 - calledby postConstruct, 275
 - calledby postInSeq, 280
 - calledby tuple2List, 368
 - calls postTranSegment, 368
 - calls postTran, 368
 - calls tuple2List, 368
 - uses \$InteractiveMode, 368
 - uses \$boot, 368
 - defun, 368
- TupleCollect, 290
 - defplist, 290
- unabbrevAndLoad
 - calledby parseHasRhs, 118
 - calledby parseHas, 116
- unAbbreviateKeyword[5]
 - called by PARSE-SpecialKeyWord, 310
- Undef
 - usedby mkOpVec, 183
- underscore, 350
 - calledby quote-if-string, 348
 - calls vector-push, 350
 - defun, 350
- unErrorRef
 - calledby addModemap1, 198
- unget-tokens, 350
 - calledby match-string, 346
 - calls line-current-segment, 350
 - calls line-new-line, 350
 - calls line-number, 350, 351
 - calls quote-if-string, 350
 - calls strconc, 350
 - calls token-nonblank, 350
 - uses valid-tokens, 351
 - defun, 350
- Union, 211
 - defplist, 211
- union
 - calledby compDefWhereClause, 184
 - calledby compJoin, 231
 - calledby makeFunctorArgumentParameters, 179
- UnionCategory, 222
 - defplist, 222
- unionq
 - calledby freelist, 430
- unknownTypeError
 - calledby addDomain, 187
 - calledby compColon, 216
- unloadOneConstructor
 - calledby compDefineLisplib, 155
- unTuple, 301
 - calledby postMapping, 282
 - calledby postTran, 258
 - calledby postType, 267
 - defun, 301
- updateCategoryFrameForCategory, 121
 - calledby compDefineLisplib, 155
 - calledby isFunctor, 188
 - calledby loadLibIfNecessary, 119
 - calls addModemap, 121
 - calls getdatabase, 121
 - calls put, 121
 - local def \$CategoryFrame, 121
 - local ref \$CategoryFrame, 121
 - defun, 121
- updateCategoryFrameForConstructor, 120
 - calledby compDefineLisplib, 155
 - calledby isFunctor, 188
 - calledby loadLibIfNecessary, 119
 - calls addModemap, 120
 - calls convertOpAlist2compilerInfo, 120
 - calls getdatabase, 120
 - calls put, 120
 - local def \$CategoryFrame, 120
 - local ref \$CategoryFrame, 120
 - defun, 120
- updateSourceFiles[5]
 - called by compileSpad2Cmd, 382
- userError
 - calledby compDefWhereClause, 184
 - calledby compOrCroak1, 402
 - calledby compReturn, 242
- valid-tokens, 98
 - usedby advance-token, 354
 - usedby current-token, 353
 - usedby next-token, 354
 - usedby try-get-token, 353
 - usedby unget-tokens, 351
 - uses \$token, 98

- defvar, 98
- vcons, 137
 - defplist, 137
- vector, 254
 - defplist, 254
- vector-push
 - calledby underscore, 350
- VectorCategory, 222
 - defplist, 222
- vmisp::optionlist
 - usedby print-defun, 399
- where, 137, 255, 291
 - defplist, 137, 255, 291
- with, 292
 - defplist, 292
- wrapDomainSub
 - calledby comp.Join, 231
- wrapSEQExit
 - calledby makeSimplePredicateOrNil, 364
- writeLib1, 160
 - calledby initializeLisplib, 159
 - calls rdefiostream, 160
 - defun, 160
- XTokenReader, 355
 - calledby get-token, 355
 - usedby get-token, 355
 - defvar, 355