Contents

Chapter 1

Introduction

In [Bruin93], the specification and implementation of an interpreter for the CSK VDM-SL specification language is described. Because the interpreter is a commercial product, we could not include the complete specification as an appendix in the thesis. The document you are reading is only available to CSK and the Overture Tools Group.

In this report, we describe an abstract state machine semantics for the CSK VDM-SL language. This language supports both the complete ISO/VDM-SL language (see [BSIVDM92]), and a structuring mechanism developed at IFAD (see [Bruin93]). In this report, the following is described:

- The complete abstract syntax of the CSK VDM-SL language.
- The semantic value domain used in the evaluation of a specification in CSK VDM-SL.
- The global definitions, types and state used in the evaluation.
- The complete dynamic semantics for the language.

All parts are specified in the CSK VDM-SL language. Therefore, some syntactical differences must be taken into account when reading the specifications. For a overview of the differences, see also [Bruin93].

This report is based on an existing document, written also at IFAD. This document, [Lassen&91], describes an abstract state machine semantics for the IPTES Meta-IV subset of ISO/VDM-SL. Therefore, not all parts of the operation semantics described in this document are the result of the MSc. project. The following are additions/changes to the semantics described in [Lassen&91]:

- In [Lassen92], a list of constructs is described that is not part of the old version of the CSK VDM-SL language. The semantics of these constructs are added to the old operation semantics. These constructs include the support for higher order functions, local function definitions, lambda expressions, and the use of exception handling.

- Due to some changes in the abstract syntax, some of the old specification had to be changed to comply to the new abstract syntax.
- The complete specification of the structuring mechanism is added to the semantics.

1.1 The Overall Stack Machine Compiler

This module specifies how ASTs representing top-level definitions such as functions and operations are translated into the stack machine instructions defined in module *STKM*.

1.1.1 Translating an AST to stack instructions

Since the core stack evaluator is naturally “stack-based”, the sequence of stack instructions generated from the AST must be in *Reverse Polish* notation. The approach taken when translating the AST representation of expressions and statements is to traverse the abstract syntax tree in *Post-order*. That is:

> When generating instructions for a node in the tree, first genrate instructions for all the childrens (in a fixed order, left to right for instance), and finally generate the instruction(s) for the node itself.

In this way the simple expression $((3 + 4 * 2) * 5)$ *mod* 5 will give an instruction sequence like: `[PUSH(3), PUSH(4), PUSH(2), MUL, ADD, PUSH(5), MUL, PUSH(5), MOD]`.

This strategy will be applied when translating VDM ASTs to stack instructions.

1.1.2 Looking Up Identifiers

When identifiers are present in expressions or statements, their immediate value must be looked up in the symbol table to evaluate the expression or statement. To make sure that identifiers are looked up at the right time during the evaluation, they are looked up simply by pushing the identifier name (*AS‘Name*) onto the stack followed by the `LOOKUP` instruction. This instruction pops one item of the stack (the identifier), looks up its value, and pushes the result back onto the stack. For instance the expression $4 * b + c$ will translate to these stack instructions `[PUSH(4), PUSH('b'), LOOKUP, MUL, PUSH('c'), LOOKUP, ADD]`.

1.1.3 Handling of Break Points, Position Information, Coverage Information and Run Time Errors

To be able to handle break points, position information, coverage information and run time errors, the generated stack instruction sequence must be annotated with some kind of debug and/or coverage information. The applied solution must fulfill the following requirements:

Break Points: It must be possible to suspend the evaluation at a certain node of the abstract syntax tree such that the stack instructions coresponding to the sub-tree(s) of this node is *not* yet evaluated.

Furthermore it must be possible to model the *Step*, *Single Step* and *Step In* functionality of the debugger. Notice that a *Single Step* does not necessarily correspond to evalute one single stack instruction. I.e. one *Single Step* may very well involve the evaluation of several stack instructions.

Position information: When the evaluation is suspended at a break point the exact position must be reported to the user interface.

Coverage information: During evaluation coverage information must be collected.

Run time errors: When a run time error is discovered the exact position of the VDM construct that caused the error must be identified.

The following solution meets several of the above requirements; The generated instruction sequence will be annotated with context information using the (void) instruction *INSTRTP‘CONTEXT*. Each context instruction is associated with a node of the AST by the unique context id (*CI‘ContextId*) of the AST node. Context instructions are generated while traversing the AST, like all other instructions, but they are inserted in the instruction sequence in *pre-order* as opposed to the post-order insertion used to generate all other instructions.

Break Points

Based on the context instruction break points can be handled as follows. Whenever a context instruction is evaluated, it is tested if the context id of the instruction is in the set of active break points. If this is the case, the evaluation is suspended. Since context instructions are inserted in pre-order, the instructions resulting from the sub-trees of the AST node that suspended the evaluation is not yet evaluated.

When the evaluation is suspended at a break point the exact file position is contained in the context information associated with the context instruction causing the suspension.

A *Single Step* and *Step In* can then be evaluated by simply evaluating all stack instructions until the next context instruction is met. The *Step* debugger command has the effect of evaluating the entire sub-trees of the current AST node, so here it is not sufficient to simply evaluate until the next context instruction — the evaluation must continue until the *right* context instruction is met. . . . We will later see how the current idea can be extended to handle the *Step* command as well.

Coverage Information

If coverage information should be collected during evaluation, the evaluation of each *INSTRTP‘CONTEXT* instruction should simply update the coverage information of the context information related to the instruction.

Run Time Errors

***** THE DOCUMENTATION BELOW IS OUTDATED *******

If, during evaluation, a run time error occurs the position of the VDM construct that caused the error must be reported to the user. For example a *division by zero* run time error will result if the divisor of a division expression equals zero, and the position of the division expression must be reported. However, since context instructions are generated in pre-order, and since run time errors are not discovered until the instructions of sub-trees are fully evaluated, it is not simple how to locate the right context instruction, and thereby the right position of the AST node that caused the run time error.

The problem is solved by maintaining a stack of context id's during evaluation. The stack instructions are extended with an instruction to pop a context id of the context stack *INSTRTP‘POPCONTEXT*, and the instruction *INSTRTP‘CONTEXT* is extended to implicitly push its context id on the context stack. The instruction sequence must the be generated to ensure that during evaluation the top item of the context stack *always* refers to the AST node currently being evaluated. This can be achieved by generating *INSTRTP‘CONTEXT* instructions in pre-order and *INSTRTP‘POPCONTEXT* instructions in post-order. I.e. the instructions for any expression or statement should be surrounded by instructions that push, respectively pop, context information.

Using this approach, whenever a run time error is detected, the position is contained on the top of the context stack.

We saw previously that the *Step* debugger command could not be realized solely on top of the *INSTRTP‘CONTEXT* instruction. However, with the additional *INSTRTP‘POPCONTEXT* instruction this realization is simple. Each instruction to pop context information will contain the context id of the AST node it represents. With this information a *Step* can simply be realized by recording the context id of the the current *INSTRTP‘CONTEXT* instruction and then evaluate stack instructions until an *INSTRTP‘POPCONTEXT* with the right context id is met.

***** THE DOCUMENTATION ABOVE IS OUTDATED *******
module *CMPL*

imports

1.0 from AS all ,

2.0 from CI all ,

3.0 from IO all ,

4.0 from PAT all ,

5.0 from REP all ,

6.0 from SEM all ,

7.0 from $FREE$ all ,

8.0 from $STKM$ all ,

9.0 from $CEXPR$ all ,

10.0 from $CSTMT$ all ,

11.0 from $RTERR$ all ,

12.0 from $STATE$ all ,

13.0 from $GLOBAL$ all ,

14.0 from $SCHDTP$ all ,

15.0 from $INSTRTP$ all ,

16.0 from $TIMEMAP$ all ,

17.0 from $TIMEPARSER$ all

exports all

definitions
types

18.0 $ProgramTable = AS`Name \xrightarrow{m} ModuleProgramTable;$

19.0 $ModuleProgramTable :: tbl : STKM`SubProgram^*$
.1 $old\text{-}id : STKM`SubProgramId$

20.0 state $sigma$ of
.1 $DebugInfo : \mathbb{B}$
.2 $curr\text{-}cl\text{-}mod : [AS`Name]$
.3 $program\text{-}table : ProgramTable$
.4 $timem : TIMEMAP`Timemap$
.5 init $s \triangleq s =$ mk-$sigma$ (true, nil , $\{\mapsto\}$,
.6 $\{\mapsto\}$)
.7 end

The field *curr_cl_mod* of the state *sigma* holds the name of the module or class currently being compiled. The operations *GetClMod* and *SetClMod* are used to read or modify this state component.

The state field *program_table* is used to maintain a mapping from unique sub-program identifiers *SubProgramId* to instruction sequences *SubProgram*. During compilation all functions, operations and lambda expressions are inserted into this table by using the operation *InsertProgram* and retrieved again by calling *GetProgram*. *InsertProgram* is used internally in this module by the three main entry point operations *CompileFnOpDef*, *CompilePrePostExpr* and *CompileLambdaBody*, while *GetProgram* is used outside this module, in particular by module *STKM*.

The function *CompileFnOpDef* converts a function or operation definition into the corresponding instruction code. Furthermore the instruction sequence generated is inserted in the program table and the index for this entry is returned.

functions

```
CompileFnOpDef : AS`FnDef | AS`OpDef → STKM`SubProgramId
CompileFnOpDef (fndef) △
  let instr =
        cases true :
          (is-AS`ExplFnDef (fndef)) →
              FnDef2I (fndef.fnpre, fndef.fnpost, fndef.body, [],
                       fndef.nm, fndef.parms, PARAMETERSLIST),
          (is-AS`ExtExplFnDef (fndef)) →
              FnDef2I (fndef.fnpre, fndef.fnpost, fndef.body,
                       fndef.resnmtps, fndef.nm, fndef.partps, PARAMETERTYPES),
          (is-AS`ImplFnDef (fndef)) →
              ImplFnDef2I (fndef.cid),
          (is-AS`ExplOpDef (fndef)) →
              OpDef2I (fndef.oppre, fndef.oppost, fndef.body, [],
                       fndef.nm, fndef.parms, fndef.constr,
                       PARAMETERS),
          (is-AS`ExtExplOpDef (fndef)) →
              OpDef2I (fndef.oppre, fndef.oppost, fndef.body,
                       fndef.resnmtps, fndef.nm, fndef.partps,
                       fndef.constr, PARAMETERTYPES),
          (is-AS`ImplOpDef (fndef)) →
              ImplOpDef2I (fndef.cid)
        end in
  InsertProgram (GetClMod (), SetContext (fndef.cid, false) ⁀ instr ⁀
                 [mk-INSTRTP`RETURN ()]);
```

The function *CompilePrePostExpr* is used to compile pre and post expressions into instructions *and* insert in the generated instruction sequence in the program table. The function returns the index of the table entry.

```
CompilePrePostExpr : AS`Expr → STKM`SubProgramId
CompilePrePostExpr (e) △
  let instr = CEXPR`E2I (e) ⁀ [mk-INSTRTP`RETURN ()] in
  InsertProgram (GetClMod (), instr);
```

When lambda expressions are handled by the compiler, the instruction sequence for the lambda expression is inserted in a semantic function value which is later pushed onto the evaluation stack when the lambda function is applied. For this reason all lambda functions must also be inserted in the program table. The function *CompileLambdaBody* will translate the expression passed to it (the body of the lambda function) and insert the instruction sequence in the program table.

23.0 $\mathit{CompileLambdaBody} : \mathit{AS}`\mathit{Expr} \to \mathit{STKM}`\mathit{SubProgramId}$
.1 $\mathit{CompileLambdaBody}\,(e) \triangleq$
.2 $\quad \mathsf{let}\ \mathit{instr} = \mathit{CEXPR}`\mathit{E2I}\,(e) \curvearrowright [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{RETURN}\,()]\ \mathsf{in}$
.3 $\quad \mathit{InsertProgram}\,(\mathit{GetClMod}\,(), \mathit{instr})$

The function *FnDef2I* compiles the pre- and postcondition and the definition of a function, being explicit or extended explicit.

operations

24.0 $\mathit{FnDef2I} : [\mathit{AS}`\mathit{Expr}] \times [\mathit{AS}`\mathit{Expr}] \times \mathit{AS}`\mathit{FnBody} \times \mathit{AS}`\mathit{NameType}^{*} \times \mathit{AS}`\mathit{Name} \times (\mathit{AS}`\mathit{ParametersList} \mid \mathit{AS}`\mathit{ParameterTypes}) \times (\textsc{ParametersList} \mid \textsc{ParameterTypes} \mid \textsc{Parameters}) \xrightarrow{o}$
.1 $\qquad \mathit{STKM}`\mathit{SubProgram}$
.2 $\mathit{FnDef2I}\,(\mathit{fnpre}, \mathit{fnpost}, \mathit{body}, \mathit{resnmtps}, \mathit{fnname}, \mathit{parms}, \mathit{type}) \triangleq$
.3 $\quad (\mathsf{if}\ \mathit{body}.\mathit{body} = \mathrm{SUBRESP}$
.4 $\quad \mathsf{then}\ \mathit{STATE}`\mathit{AddAbstract}(\mathit{GetClMod}\,())\,;$
.5 $\quad \mathsf{let}\ \mathit{i\text{-}body} = \mathsf{if}\ \mathit{body}.\mathit{body} = \mathrm{NOTYETSPEC} \lor$
.6 $\qquad\qquad \mathit{body}.\mathit{body} = \mathrm{SUBRESP}$
.7 $\qquad\qquad \mathsf{then\ if}\ \mathit{body}.\mathit{body} = \mathrm{NOTYETSPEC} \land$
.8 $\qquad\qquad\qquad \mathit{STATE}`\mathit{IsDLClass}\,(\mathit{GetClMod}\,())$
.9 $\qquad\qquad\qquad \mathsf{then}\ [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{DLCALL}\,(\mathit{GetClMod}\,(), \mathit{fnname})]$
.10 $\qquad\qquad\qquad \mathsf{else}\ [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{NOBODY}\ (\mathsf{if}\ \mathit{body}.\mathit{body} = \mathrm{NOTYETSPEC}$
.11 $\qquad\qquad\qquad\qquad \mathsf{then}\ \mathit{RTERR}`\mathit{NOTYETSPECFCT}$
.12 $\qquad\qquad\qquad\qquad \mathsf{else}\ \mathit{RTERR}`\mathit{SUBRESP},$
.13 $\qquad\qquad\qquad\qquad \mathit{GetClMod}\,(), \mathit{fnname},$
.14 $\qquad\qquad\qquad\qquad \mathit{MakeParmList}\,(\mathit{parms}, \mathit{type}))]$
.15 $\qquad\qquad \mathsf{else}\ \mathit{CEXPR}`\mathit{E2I}\,(\mathit{body}.\mathit{body}),$
.16 $\qquad \mathit{i\text{-}fnpre} = \mathit{PrePost2I}\,(\mathit{fnpre}, \mathsf{true}),$
.17 $\qquad \mathit{i\text{-}fnpost} = \mathsf{if}\ \mathit{fnpost} = \mathsf{nil}$
.18 $\qquad\qquad \mathsf{then}\ []$
.19 $\qquad\qquad \mathsf{else\ let}\ \mathit{pr\text{-}i} = [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{COPYVAL}\,()] \curvearrowright$
.20 $\qquad\qquad\qquad [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{POSTENV}\,(\mathit{resnmtps}, \mathit{fnpost}.\mathit{cid})],$
.21 $\qquad\qquad\qquad \mathit{po\text{-}i} = [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{POPBLKENV}\,()]\ \mathsf{in}$
.22 $\qquad\qquad\quad \mathit{pr\text{-}i} \curvearrowright \mathit{PrePost2I}\,(\mathit{fnpost}, \mathsf{false}) \curvearrowright \mathit{po\text{-}i}\ \mathsf{in}$
.23 $\quad \mathsf{return}\ \mathit{i\text{-}fnpre} \curvearrowright \mathit{i\text{-}body} \curvearrowright \mathit{i\text{-}fnpost}\ )$

functions

25.0 $\mathit{MakeParmList} : (\mathit{AS}`\mathit{ParametersList} \mid \mathit{AS}`\mathit{ParameterTypes} \mid \mathit{AS}`\mathit{Parameters}) \times (\textsc{ParametersList} \mid \textsc{ParameterTypes} \mid \textsc{Parameters}) \to$
.1 $\qquad \mathit{AS}`\mathit{Parameters}$
.2 $\mathit{MakeParmList}\,(\mathit{list}, \mathit{type}) \triangleq$
.3 $\quad \mathsf{if}\ \mathit{type} = \textsc{Parameters}$
.4 $\quad \mathsf{then}\ \mathit{list}$
.5 $\quad \mathsf{elseif}\ \mathit{type} = \textsc{ParametersList}$
.6 $\quad \mathsf{then\ if\ len}\ \mathit{list} = 0$
.7 $\qquad \mathsf{then}\ []$
.8 $\qquad \mathsf{else\ hd}\ \mathit{list}$
.9 $\quad \mathsf{else\ conc}\ [\mathit{list}\,(i).\mathit{pats} \mid i \in \mathsf{inds}\ \mathit{list}];$

The function *PrePost2I* compiles pre and post conditions to instruction code.

26.0 $\mathit{PrePost2I} : [\mathit{AS}`\mathit{Expr}] \times \mathbb{B} \to \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{PrePost2I}\,(\mathit{cond}, \mathit{precond}) \triangleq$
.2 $\quad (\mathsf{if}\ \mathit{cond} = \mathsf{nil}$
.3 $\quad \mathsf{then}\ []$
.4 $\quad \mathsf{else\ let}\ i\text{-}\mathit{cond} = \mathit{CEXPR}`\mathit{E2I}\,(\mathit{cond}),$
.5 $\qquad \mathit{prepost} = \mathsf{if}\ \mathit{precond}$
.6 $\qquad\qquad \mathsf{then}\ [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{PRE}\,()]$
.7 $\qquad\qquad \mathsf{else}\ [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{POST}\,()],$
.8 $\qquad \mathit{errMsg} = \mathsf{if}\ \mathit{precond}$
.9 $\qquad\qquad \mathsf{then}\ \mathit{CompileRunTime}\,(\mathit{RTERR}`\mathit{EVAL\text{-}PRE\text{-}GIVES\text{-}FALSE}, \mathit{cond}.\mathit{cid})$
.10 $\qquad\qquad \mathsf{else}\ \mathit{CompileRunTime}\,(\mathit{RTERR}`\mathit{EVAL\text{-}POST\text{-}GIVES\text{-}FALSE}, \mathit{cond}.\mathit{cid}),$
.11 $\qquad \mathit{exc} = [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{CBR}\,(\mathsf{len}\ \mathit{errMsg})] \curvearrowright \mathit{errMsg}\ \mathsf{in}$
.12 $\quad \mathit{prepost} \curvearrowright [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{CNBR}\,(\mathsf{len}\ i\text{-}\mathit{cond} + \mathsf{len}\ \mathit{exc})] \curvearrowright i\text{-}\mathit{cond} \curvearrowright \mathit{exc});$

27.0 $\mathit{ImplFnDef2I} : \mathit{CI}`\mathit{ContextId} \to \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{ImplFnDef2I}\,(\mathit{cid}) \triangleq$
.2 $\quad \mathit{CompileRunTime}\,(\mathit{RTERR}`\mathit{IMPL\text{-}FN\text{-}APPLY}, \mathit{cid})$

operations

```
28.0 OpDef2I : [AS`Expr] × [AS`Expr] × AS`OpBody × AS`NameType* × AS`Name ×
(AS`Parameters | AS`ParameterTypes) × 𝔹 × (ParametersList | ParameterTypes |
Parameters) -o->
            STKM`SubProgram

OpDef2I (precond, postcond, body, resnmtps, fnname, parms, constr, type) ≜
  (if body.body = SUBRESP
   then STATE`AddAbstract(GetClMod ()) ;
   let i-body = if body.body = NOTYETSPEC ∨
                   body.body = SUBRESP
                then if body.body = NOTYETSPEC ∧
                        STATE`IsDLClass (GetClMod ())
                     then [mk-INSTRTP`DLCALL (GetClMod (), fnname)]
                     else [mk-INSTRTP`NOBODY (if body.body = NOTYETSPEC
                                              then RTERR`NOTYETSPECOP
                                              else RTERR`SUBRESP,
                                              GetClMod (), fnname,
                                              MakeParmList (parms, type))]
                else (CSTMT`S2I (body.body) ⁀
                     if constr
                     then [mk-INSTRTP`POP (1), mk-INSTRTP`SELFEXPR ()]
                     else []),
       i-fnpre = PrePost2I (precond, true),
       i-fnpost = if postcond = nil
                  then []
                  else let pr-i = [mk-INSTRTP`COPYVAL ()] ⁀
                                  [mk-INSTRTP`POSTENV (resnmtps, postcond.cid)],
                           po-i = [mk-INSTRTP`POPBLKENV ()] in
                       pr-i ⁀ PrePost2I (postcond, false) ⁀ po-i in
   return i-fnpre ⁀ i-body ⁀ i-fnpost )
```

functions

```
29.0 ImplOpDef2I : CI`ContextId → STKM`SubProgram

ImplOpDef2I (cid) ≜
  CompileRunTime (RTERR`IMPL-OP-CALL, cid)
```

1.1.4 Context Manipulation Operations

operations

```
30.0 SetContext : CI`ContextId × 𝔹 -o-> STKM`SubProgram

SetContext (cid, isStmt) ≜
  (if cid ≠ CI`NilContextId
   then CI`SetBreakable(cid) ;
   if DebugInfo
   then return [mk-INSTRTP`CONTEXT (cid, isStmt)]
   else return [] );
```

31.0 $SetDebugInfo : \mathbb{B} \xrightarrow{o} ()$
.1 $SetDebugInfo\,(b) \triangleq$
.2 $DebugInfo := b;$

32.0 $SetClMod : AS`Name \xrightarrow{o} ()$
.1 $SetClMod\,(nm) \triangleq$
.2 $curr\text{-}cl\text{-}mod := nm;$

33.0 $GetClMod : () \xrightarrow{o} AS`Name$
.1 $GetClMod\,() \triangleq$
.2 $\mathsf{return}\ curr\text{-}cl\text{-}mod$
.3 $\mathsf{pre}\ \ curr\text{-}cl\text{-}mod \neq \mathsf{nil}\ ;$

The following operations are used by the stack instruction compiler. They are used to maintain a table of instruction sequences (*STKM'SubProgram*) for each module or class of the specification. The table is (currently) a part of the state of this module. A semantic value for a function or operation will then contain an index to this program table instead of a copy of the instruction sequence.

The operation *ResetProgramTable* resets the program table associated with the module passed to the operation.

34.0 $ResetProgramTable : AS`Name \xrightarrow{o} ()$
.1 $ResetProgramTable\,(modnm) \triangleq$
.2 $(program\text{-}table := program\text{-}table \dagger \{modnm \mapsto \mathsf{mk}\text{-}ModuleProgramTable\,([], 0)\});$

The *InsertProgram* operation inserts the instruction sequence *instr* into the program table of module *modnm*. The index for this new entry is returned.

35.0 $InsertProgram : AS`Name \times STKM`SubProgram \xrightarrow{o} STKM`SubProgramId$
.1 $InsertProgram\,(modnm, instr) \triangleq$
.2 $\mathsf{let}\ \mathsf{mk}\text{-}ModuleProgramTable\,(tbl, id) = program\text{-}table\,(modnm)\ \mathsf{in}$
.3 $(program\text{-}table := program\text{-}table \dagger$
.4 $\{modnm \mapsto \mathsf{mk}\text{-}ModuleProgramTable\,(tbl \curvearrowright [instr], id + 1)\};$
.5 $\mathsf{return}\ id + 1\ );$

36.0 $GetProgram : AS`Name \times STKM`SubProgramId \xrightarrow{o} STKM`SubProgram$

.1 $GetProgram\,(modnm, id) \triangleq$

.2 let mk-$ModuleProgramTable\,(tbl, \text{-}) = program\text{-}table\,(modnm)$ in

.3 if $id >$ len tbl

.4 then error

.5 else return $tbl\,(id)$;

37.0 $CopyProgram : AS`Name \times AS`Name \xrightarrow{o} ()$

.1 $CopyProgram\,(old, new) \triangleq$

.2 $program\text{-}table := program\text{-}table \dagger \{new \mapsto program\text{-}table\,(old)\}$

1.1.5 internal debug-enabeling functions

functions

38.0 $IStart : \mathsf{char}^* \times CI`ContextId \rightarrow STKM`SubProgram$

.1 $IStart\,(name, cid) \triangleq$

.2 [mk-$INSTRTP`ISTART\,(name, cid)$];

39.0 $IEnd : \mathsf{char}^* \rightarrow STKM`SubProgram$

.1 $IEnd\,(name) \triangleq$

.2 [mk-$INSTRTP`IEND\,(name)$];

1.1.6 Run Time Error functions

40.0 $CompileRunTime : RTERR`ERR \times CI`ContextId \rightarrow STKM`SubProgram$

.1 $CompileRunTime\,(rterr, cid) \triangleq$

.2 $SetContext\,(cid, \mathsf{false}) \curvearrowright$ [mk-$INSTRTP`ERRINST\,(rterr)$]

1.1.7 The Time map

operations

41.0 $SetTM : TIMEMAP`Timemap \xrightarrow{o} ()$

.1 $SetTM\,(newtm) \triangleq$

.2 $timem := newtm;$

42.0 $GetTM : () \xrightarrow{o} TIMEMAP`Timemap$

.1 $GetTM\ () \triangleq$

.2 return $timem$

end $CMPL$

Test Suite : rtinfo.ast
Module : CMPL

Name	#Calls	Coverage
CMPL‘IEnd	**undefined**	**undefined**
CMPL‘GetTM	**undefined**	**undefined**
CMPL‘SetTM	**undefined**	**undefined**
CMPL‘IStart	**undefined**	**undefined**
CMPL‘FnDef2I	**undefined**	**undefined**
CMPL‘OpDef2I	**undefined**	**undefined**
CMPL‘GetClMod	**undefined**	**undefined**
CMPL‘SetClMod	**undefined**	**undefined**
CMPL‘PrePost2I	**undefined**	**undefined**
CMPL‘GetProgram	**undefined**	**undefined**
CMPL‘SetContext	**undefined**	**undefined**
CMPL‘CopyProgram	**undefined**	**undefined**
CMPL‘ImplFnDef2I	**undefined**	**undefined**
CMPL‘ImplOpDef2I	**undefined**	**undefined**
CMPL‘MakeParmList	**undefined**	**undefined**
CMPL‘SetDebugInfo	**undefined**	**undefined**
CMPL‘InsertProgram	**undefined**	**undefined**
CMPL‘CompileFnOpDef	**undefined**	**undefined**
CMPL‘CompileRunTime	**undefined**	**undefined**
CMPL‘CompileLambdaBody	**undefined**	**undefined**
CMPL‘ResetProgramTable	**undefined**	**undefined**
CMPL‘CompilePrePostExpr	**undefined**	**undefined**
Total Coverage		**0%**

1.2 Compilation of Expressions

This module specifies how ASTs representing expressions are translated into the stack machine instructions defined in module *STKM*.
module $CEXPR$

imports

from AS all ,

from CI all ,

from IO all ,

from AUX all ,

from PAT all ,

from REP all ,

from SEM all ,

from $CMPL$ all ,

from $CPAT$ all ,

from $FREE$ all ,

from $STKM$ all ,

from $TIME$

functions $MkBr : () \to STKM`SubProgram;$
$MkCbr : () \to STKM`SubProgram;$
$E2Time : AS`Expr \to STKM`SubProgram;$
$MkMatchPattern : () \to STKM`SubProgram;$
$MkMapCompInsert : () \to STKM`SubProgram;$
$MkSeqCompInsert : () \to STKM`SubProgram;$
$MkSetCompInsert : () \to STKM`SubProgram;$
$IsRuntimeBinaryOp : AS`BinaryOp \to \mathbb{B};$
$IsRuntimePrefixOp : AS`UnaryOp \to \mathbb{B};$
$MkRuntimeBinaryOp : AS`BinaryOp \to STKM`SubProgram;$
$MkRuntimePrefixOp : AS`UnaryOp \to STKM`SubProgram;$
$MkRuntimeSetSeqMap : AS`SetRangeExpr \mid AS`SubSequenceExpr \mid$
$AS`SeqModifyMapOverrideExpr \to$
$STKM`SubProgram$

operations $GetCompilingTime : () \xrightarrow{o} \mathbb{B}$,

from $RTERR$ all ,

from $STATE$

operations $GetInhStrct : AS`Name \xrightarrow{o} AS`Name\text{-set}^*;$
$GetInstVars : AS`Name \xrightarrow{o} AS`InstAssignDef^*;$
$GetDefaultCons : AS`Name \xrightarrow{o} \mathbb{B}$,

from $GLOBAL$ all ,

from $SCHDTP$ all ,

from $INSTRTP$ all ,

from $TIMEMAP$

types $Timemap$

exports all

definitions

1.2.1 Introduction to Functions and Apply Expression

Consider the apply expression below:

```
f(mk_(3,4), a)
```

The apply expression can in be a map, sequence or function apply.

The corresponding instruction code for this should be:

```
[ EMPTYLIST,
  E2I("mk_(3,4)"), APPENDESTCK,
  E2I("a"), APPENDESTCK,            // Now a sequence of semantic values
                                  // is pushed on the evaluation stack.
  "f", LOOKUP,                    // Now the semantic value of "f"
                                  // is pushed on the evaluation stack.
  APPLY
]
```

The `Apply` instruction pops two elements from the evaluation stack: the semantic value of the "applicator" and its argumentlist. See the description of the `APPLY` instruction in Section **??**. For function application the APPLY instruction sets-up the block-environment and scope environment. It pushes the semantic value of the function on the call stack.

Returning from the function application consists of:

- Cleaning the environment for the function application, that is, the block environment and object scope.
- Pop the instruction set of the call stack.

See the `RETURN` instruction (Section **??**) and the compilation order for a function body (Section **??**) for a further description.

Say that the function *f* in the example performs a simple addition:

```
f: (nat * nat) * bool -> nat
f( mk_(a,b), -) ==
  a+b
```

The call stack looks like:

[.., EMPTYLIST, ..., APPLY , ...]	3
...	..

Say that the PC=45 when executing the APPLY instruction, the PC is set to PC=(1) and the function stack looks like:

["a", LOOKUP, "b", LOOKUP, mk_BINOP(<ADD>), RETURN]	45
[..., EMPTYLIST, ..., APPLY, ...]	3

Executing the instruction RETURN will pop from the call stack which sets the PC to 45 again.

The function *E2I* takes an expression and generates corresponding instruction code.
functions

```
E2I : AS`Expr → STKM`SubProgram
E2I (e) △
  let mk- (name, expr-l) =
          cases true :
            (is-AS`BracketedExpr (e)) → mk- ("BracketedExpr", E2I (e.expr)),
            (is-AS`DefExpr (e)) → mk- ("DefExpr", CompileDefExpr (e)),
            (is-AS`LetExpr (e)) → mk- ("LetExpr", CompileLetExpr (e)),
            (is-AS`LetBeSTExpr (e)) → mk- ("LetBeSTExpr", CompileLetBeSTExpr (e)),
            (is-AS`AllOrExistsExpr (e)) → mk- ("AllOrExistsExpr", CompileAllOrExistsExpr (e)),
            (is-AS`ExistsUniqueExpr (e)) → mk- ("ExistsUniqueExpr", CompileExistsUniqueExpr (e)),
            (is-AS`IotaExpr (e)) → mk- ("IotaExpr", CompileIotaExpr (e)),
            (is-AS`ApplyExpr (e)) → mk- ("ApplyExpr", CompileApplyExpr (e)),
            (is-AS`FieldSelectExpr (e)) → mk- ("FieldSelectExpr", CompileFieldSelectExpr (e)),
            (is-AS`IfExpr (e)) → mk- ("IfExpr", CompileIfExpr (e)),
            (is-AS`CasesExpr (e)) → mk- ("CasesExpr", CompileCasesExpr (e)),
            (is-AS`MapInverseExpr (e)) → mk- ("MapInverseExpr", CompileMapInverseExpr (e)),
            (is-AS`PrefixExpr (e)) → mk- ("PrefixExpr", CompileUnaryExpr (e)),
            (is-AS`BinaryExpr (e)) → mk- ("BinaryExpr", CompileBinaryExpr (e)),
            (is-AS`SetRangeExpr (e)) → mk- ("SetRangeExpr", CompileSetRangeExpr (e)),
            (is-AS`SubSequenceExpr (e)) → mk- ("SubSequenceExpr", CompileSubSequenceExpr (e)),
            (is-AS`SetEnumerationExpr (e)) → mk- ("SetEnumerationExpr", CompileSetEnumExpr (e)),
            (is-AS`SeqEnumerationExpr (e)) → mk- ("SeqEnumerationExpr", CompileSeqEnumExpr (e)),
            (is-AS`MapEnumerationExpr (e)) → mk- ("MapEnumerationExpr", CompileMapEnumExpr (e)),
            (is-AS`SetComprehensionExpr (e)) → mk- ("SetComprehensionExpr", CompileSetComprehension
            (is-AS`SeqComprehensionExpr (e)) → mk- ("SeqComprehensionExpr", CompileSeqComprehensio
            (is-AS`MapComprehensionExpr (e)) → mk- ("MapComprehensionExpr", CompileMapComprehen
            (is-AS`TupleConstructorExpr (e)) → mk- ("TupleConstructorExpr", CompileTupleConstructorEx
            (is-AS`RecordConstructorExpr (e)) → mk- ("RecordConstructorExpr", CompileRecordConstructor
            (is-AS`RecordModifierExpr (e)) → mk- ("RecordModifierExpr", CompileRecordModifierExpr (e)),
            (is-AS`SeqModifyMapOverrideExpr (e)) → mk- ("SeqModifyMapOverrideExpr",
                                      CompileSeqModifyMapOverrideExpr (e)),
            (is-AS`LambdaExpr (e)) → mk- ("LambdaExpr", CompileLambdaExpr (e)),
            (is-AS`FctTypeInstExpr (e)) → mk- ("FctTypeInstExpr", CompileFctTypeInstExpr (e)),
            (is-AS`IsExpr (e)) → mk- ("IsExpr", CompileIsExpr (e)),
            (is-AS`TokenConstructorExpr (e)) →
                mk- ("TokenConstructorExpr", CompileTokenConstructorExpr (e)),
            (is-AS`TupleSelectExpr (e)) → mk- ("TupleSelectExpr", CompileTupleSelectExpr (e)),
            (is-AS`TypeJudgementExpr (e)) → mk- ("TypeJudgementExpr", CompileTypeJudgementExpr (e))
            (is-AS`PreConditionApplyExpr (e)) → mk- ("PreConditionApplyExpr", CompilePreConditionApp
            (is-AS`Name (e)) → mk- ("Name", [mk-INSTRTP`LOOKUP (e)]),
            (is-AS`OldName (e)) → mk- ("OldName", [mk-INSTRTP`LOOKUP (e)]),
            (is-AS`BoolLit (e)) → mk- ("BoolLit", [mk-INSTRTP`PUSH (mk-SEM`BOOL (e.val))]),
            (is-AS`NilLit (e)) → mk- ("NilLit", [mk-INSTRTP`PUSH (mk-SEM`NIL ())]),
            (is-AS`RealLit (e)) → mk- ("RealLit", [mk-INSTRTP`PUSH (mk-SEM`NUM (e.val))]),
            (is-AS`TextLit (e)) → mk- ("TextLit", [mk-INSTRTP`PUSH (mk-SEM`SEQ ([mk-SEM`CHAR (e.

                                                          i ∈
inds e.val]))]),
            (is-AS`CharLit (e)) → mk- ("CharLit", [mk-INSTRTP`PUSH (mk-SEM`CHAR (e.val))]),
            (is-AS`QuoteLit (e)) → mk- ("QuoteLit", [mk-INSTRTP`PUSH (mk-SEM`QUOTE (e.val))]),
            (is-AS`UndefinedExpr (e)) → mk- ("UndefinedExpr",
                              [mk-INSTRTP`ERRINST (RTERR`UNDEFINED-EXPRESS
            (is-AS`LastRes (e)) → mk- ("LastRes", [mk-INSTRTP`LASTRES ()]),
            (is-AS`SelfExpr (e)) → mk- ("SelfExpr", CompileSelfExpr (e)),
            (is-AS`NewExpr (e)) → mk- ("NewExpr", CompileNewExpr (e, nil )),
            (is-AS`IsOfClassExpr (e)) → mk- ("IsOfClassExpr", CompileIsOfClassExpr (e)),
            (is-AS`IsOfBaseClassExpr (e)) → mk- ("IsOfBaseClassExpr", CompileIsOfBaseClassExpr (e)),
            (is-AS`SameBaseClassExpr (e)) → mk- ("SameBaseClassExpr", CompileSameBaseClassExpr (e))
            (is-AS`SameClassExpr (e)) → mk- ("SameClassExpr", CompileSameClassExpr (e)),
            (is-AS`ThreadIdExpr (e)) → mk- ("ThreadIdExpr", CompileThreadIdExpr (e)),
```

1.2.2 Define Expression

operations

```
CompileDefExpr : AS`DefExpr -o-> STKM`SubProgram
CompileDefExpr (mk-AS`DefExpr (def-l, in-e, -)) ≜
  (dcl sp : STKM`SubProgram := [mk-INSTRTP`EMPTYBLKENV (READ_ONLY)];
   for mk- (pb, expr) in def-l
   do let p-instr = CPAT`PB2I (pb) in
        sp := sp ↷ E2I (expr) ↷ p-instr ↷
                 [mk-INSTRTP`MATCHANDBIND ()];
   sp := sp ↷ E2I (in-e) ↷ [mk-INSTRTP`POPBLKENV ()];
   return sp );
```

1.2.3 Let Expression

```
CompileLetExpr : AS`LetExpr -o-> STKM`SubProgram
CompileLetExpr (mk-AS`LetExpr (localdef, In, -)) ≜
  (dcl sp : STKM`SubProgram := [mk-INSTRTP`EMPTYBLKENV (READ_ONLY)];
   for elm in localdef
   do if is-AS`ExplFnDef (elm) ∨ is-AS`ImplFnDef (elm) ∨ is-AS`ExtExplFnDef (elm)
      then let mk- (blkenv, b-m) = PAT`ConstructFN (elm) in
             sp := sp ↷ [mk-INSTRTP`CLOSENV (blkenv, b-m)]
      else let dtc = if elm.tp = nil
                     then []
                     else [mk-INSTRTP`DTC (elm.tp)] in
             sp := sp ↷ E2I (elm.val) ↷ dtc ↷ CPAT`P2I (elm.pat) ↷
                      [mk-INSTRTP`MATCHANDBIND ()];
   sp := sp ↷ E2I (In) ↷ [mk-INSTRTP`POPBLKENV ()];
   return sp );
```

1.2.4 Let be such that Expression

If a let-be-expression contains a type binding a run-time error must be raised. Otherwise instructions for the "such that" part must be constructed seperately. Since it is neccesary to loop through the alternative values in the set a general strategy is used: First instructions before the loop are gathered. Then the instructions inside the loop are collected and finally the instructions clealing up after the loop are produced. These three components are combined by the CombRepeatUntil function. This means that a VDM expression such as:

```
let pat in set setexpr be st pred
in
  inexpr
```

will produce an instruction sequence like:

```
[E2I(setexpr),
 mk_INSTRTP‘EMPTYBLKENV(<READ_ONLY>), -- this one is simpy there because the
                                -- loop must start by popping a blockenv.
 -- outer loop part
 mk_INSTRTP‘POPBLKENV(),
 mk_INSTRTP‘EMPTYBLKENV(<READ_ONLY>),
 mk_INSTRTP‘SELELEM()],
 CPAT‘P2I(lhs.pat),
 mk_INSTRTP‘TRYMATCH(),
 mk_INSTRTP‘CBR(3),
 mk_INSTRTP‘PUSH(mk_SEM‘BOOL(false)), -- take next elem
 mk_INSTRTP‘BR("until end of outer loop")

 -- inner loop part
 E2I(pred),
 mk_INSTRTP‘CBR(8),
 mk_INSTRTP‘ISEMPTYSET(2),
 mk_INSTRTP‘CBR(3)
 mk_INSTRTP‘PUSH(mk_SEM‘BOOL(true)),
 mk_INSTRTP‘SELBLKENV(2),
 mk_INSTRTP‘BR("until end of inner loop")

 mk_INSTRTP‘PUSH(mk_SEM‘BOOL(false)),
 mk_INSTRTP‘PUSH(mk_SEM‘BOOL(false)),
 mk_INSTRTP‘BR("until end of inner loop")
 mk_INSTRTP‘PUSH(mk_SEM‘BOOL(true)),
 -- end inner loop part

 mk_INSTRTP‘CNBR( -len inner_loop_part),
 mk_INSTRTP‘REMSTACKELEM(2), -- get rid of the set of blkenvs
 -- end outer loop part
 mk_INSTRTP‘CNBR(-len outer_loop_part),
 mk_INSTRTP‘POP(1), -- get rid of the rest of the set value from the stack
 E2I(inexpr),
 mk_INSTRTP‘POPBLKENV()]
```

prep_instr: This part is preparing the remaining instruction evaluation. The value stack will be extended with the set value and the environment list will be extended with an empty block environment.

out_loop_instr: The outer loop instructions start by checking whether the set value on top of the evaluation stack is non-empty. If this is the case an element from the set is selected and the pattern is matched against this element. If this succeeds the inner loop is entered. After completing the inner loop it is necessary to get rid of the set of environments again. Thus, the instructions in the outer loop does not add anything to the evaluation stack upon completion.

in_loop_instr: The inner loop instructions assumes that the evaluation stack contains a set of environments at the top and that the next element is a set value. It starts by checking whether the predicate is satisfied for the selected environment. In case it is satisfied we need to leave both loops. Otherwise we must select a new environment from the top of the evaluation stack in case there is more and then repeat the inner loop again. In case there is no more environments left we should leave the inner loop and run the outer loop again.

postt_instr: The post-instructions are the part which removes the remaning set from the evaluation stack, evaluates the instructions for the in-expression and finally pops the block environment which was pushed for the matching between the pattern and the selected element from the set.

```
68.0  CompileLetBeSTExpr : AS`LetBeSTExpr -o-> STKM`SubProgram
 .1   CompileLetBeSTExpr (mk-AS`LetBeSTExpr (lhs, St, In, -)) ≜
 .2     (if is-AS`TypeBind (lhs)
 .3      then return [mk-INSTRTP`ERRINST (RTERR`TYPE-BIND-EVAL)] ;
 .4      let St-instr = if St = nil
 .5                     then [mk-INSTRTP`PUSH (mk-SEM`BOOL (true))]
 .6                     else E2I (St),
 .7          succ-instr = [mk-INSTRTP`PUSH (mk-SEM`BOOL (true))],
 .8          fail-instr = [mk-INSTRTP`PUSH (mk-SEM`BOOL (false))],
 .9          St-failed = ConcIfThenElse ([mk-INSTRTP`ISEMPTYSET (1, nil )],
 .10                                     fail-instr ⁀ succ-instr,
 .11                                     [mk-INSTRTP`SELBLKENV (1)]⁀fail-instr) in
 .12     let prep-instr = E2I (lhs.Set)⁀[mk-INSTRTP`EMPTYBLKENV (READ_ONLY)],
 .13         in-loop-instr = ConcIfThenElse (St-instr,
 .14                                         succ-instr ⁀ succ-instr,
 .15                                         St-failed),
 .16         postt-instr = [mk-INSTRTP`POP (1)] ⁀ E2I (In) ⁀
 .17                       [mk-INSTRTP`POPBLKENV ()],
 .18         empty-instr = ConcIfThenElse ([mk-INSTRTP`ISEMPTYSET (1, SEM)],
 .19                                       [mk-INSTRTP`ERRINST (RTERR`EMPTY-ENV-S)],
 .20                                       [mk-INSTRTP`SELELEM ()] ⁀
 .21                                       CPAT`P2I (lhs.pat) ⁀
 .22                                       [mk-INSTRTP`TRYMATCH ()]) in
 .23     let out-loop-instr = ConcIfThenElse (empty-instr,
 .24                                          CombRepeatUntil (in-loop-instr) ⁀
 .25                                          [mk-INSTRTP`REMSTACKELEM (2)],
 .26                                          fail-instr) in
 .27     return prep-instr ⁀ CombRepeatUntil (out-loop-instr) ⁀ postt-instr )
```

functions

```
69.0  CombRepeatUntil : STKM`SubProgram → STKM`SubProgram
 .1   CombRepeatUntil (loop-instr) ≜
 .2     loop-instr ⁀ [mk-INSTRTP`CNBR (− len loop-instr − 1)];
```

70.0 $CombWhileLoop : STKM`SubProgram \to STKM`SubProgram$

.1 $CombWhileLoop\,(loop\text{-}instr) \triangleq$

.2 $[\mathsf{mk}\text{-}INSTRTP`CNBR\,(\mathsf{len}\ loop\text{-}instr + 1)] \curvearrowright loop\text{-}instr \curvearrowright$

.3 $[\mathsf{mk}\text{-}INSTRTP`CBR\,(-\ \mathsf{len}\ loop\text{-}instr - 1)];$

71.0 $CombWhileLoopWithCond : STKM`SubProgram \times STKM`SubProgram \to STKM`SubProgram$

.1 $CombWhileLoopWithCond\,(condition, body) \triangleq$

.2 $(\mathsf{let}\ while\text{-}body = condition \curvearrowright [\mathsf{mk}\text{-}INSTRTP`CNBR\,(\mathsf{len}\ body + 1)] \curvearrowright body,$

.3 $goto\text{-}start = [\mathsf{mk}\text{-}INSTRTP`BR\,(-\ \mathsf{len}\ while\text{-}body - 1)]\ \mathsf{in}$

.4 $while\text{-}body \curvearrowright goto\text{-}start)$

1.2.5 Quantified Expressions

A quantified expression such as:

```
forall a in set {1,2} & a > 1
```

will compile into code such as:
operations

72.0 $CompileAllOrExistsExpr : AS`AllOrExistsExpr \xrightarrow{o} STKM`SubProgram$

.1 $CompileAllOrExistsExpr\,(\mathsf{mk}\text{-}AS`AllOrExistsExpr\,(quant, bind\text{-}l, pred, \text{-})) \triangleq$

.2 $(\mathsf{let}\ prep\text{-}instr = CPAT`CompileMultBindL\,(bind\text{-}l, \mathrm{DONT\_PARTITION}),$

.3 $q\text{-}i = [\mathsf{mk}\text{-}INSTRTP`PUSH\,(\mathsf{mk}\text{-}SEM`BOOL\,(quant = \mathrm{ALL}))],$

.4 $succ\text{-}instr = [\mathsf{mk}\text{-}INSTRTP`PUSH\,(\mathsf{mk}\text{-}SEM`BOOL\,(\mathsf{true}))],$

```
.5           fail-instr = [mk-INSTRTP`PUSH (mk-SEM`BOOL (false))] in
.6       let body-instr =
.7               [mk-INSTRTP`SELBLKENV (1)] ⁀
.8               E2I (pred) ⁀
.9               [mk-INSTRTP`POPBLKENV ()] ⁀
.10              ConcIfThenElse ([], if quant = ALL
.11                                  then fail-instr
.12                                  else [mk-INSTRTP`POP (1)] ⁀
.13                                       succ-instr ⁀
.14                                       succ-instr,
.15                                  if quant = ALL
.16                                  then [mk-INSTRTP`POP (1)] ⁀
.17                                       fail-instr ⁀
.18                                       succ-instr
.19                                  else fail-instr) in
.20      let loop-instr = ConcIfThenElse ([mk-INSTRTP`ISEMPTYSET (1, nil )],
.21                                       [mk-INSTRTP`POP (1)] ⁀ q-i ⁀ succ-instr,
.22                                       body-instr) in
.23      return prep-instr ⁀ CombRepeatUntil (loop-instr) );


73.0  CompileEUandICommon : AS`Bind × AS`Expr -o-> STKM`SubProgram
.1    CompileEUandICommon (bind, expr) △
.2      if is-AS`TypeBind (bind)
.3      then return [mk-INSTRTP`ERRINST (RTERR`TYPE-BIND-EVAL)]
.4      else let succ-instr = [mk-INSTRTP`PUSH (mk-SEM`BOOL (true))],
.5               fail-instr = [mk-INSTRTP`PUSH (mk-SEM`BOOL (false))],
.6               mk-AS`SetBind (pat, Set, ci) = bind,
.7               bind-l = [mk-AS`MultSetBind ([pat], Set, ci)],
.8               prep-instr = CPAT`CompileMultBindL (bind-l, DO_PARTITION) ⁀
.9                            [mk-INSTRTP`PUSH ({})],
.10              pred-instr = E2I (expr),
.11              more-than-one-pred = [mk-INSTRTP`ADDTOBLKENV ()] ⁀
.12                                   ConcIfThenElse ([mk-INSTRTP`SIZE (2)],
.13                                                   succ-instr, fail-instr),
.14              loop-body =
.15                  [mk-INSTRTP`SELBLKENV (2)] ⁀ pred-instr ⁀
.16                  ConcIfThenElse ([], more-than-one-pred, fail-instr) ⁀
.17                  [mk-INSTRTP`POPBLKENV ()],
.18              loop-instr = ConcIfThenElse ([mk-INSTRTP`ISEMPTYSET (2, nil )], succ-instr, loop-body) in
.19          return prep-instr ⁀ CombRepeatUntil (loop-instr) ;


74.0  CompileExistsUniqueExpr : AS`ExistsUniqueExpr -o-> STKM`SubProgram
.1    CompileExistsUniqueExpr (mk-AS`ExistsUniqueExpr (bind, pred-e, -)) △
.2      let new-bind = PAT`DoCarePattern (bind, mk-AS`Name (["1"], CI`NilContextId)),
.3          common = CompileEUandICommon (new-bind, pred-e) in
.4      return common⁀[mk-INSTRTP`SIZE (1), mk-INSTRTP`REMSTACKELEM (2), mk-INSTRTP`REMSTAC
```

75.0 $\mathit{CompileIotaExpr} : \mathit{AS}`\mathit{IotaExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{CompileIotaExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{IotaExpr}\,(\mathit{bind}, \mathit{pred\text{-}e}, \text{-})) \triangleq$

.2 $\quad \textsf{let } \mathit{new\text{-}bind} = \mathit{PAT}`\mathit{DoCarePattern}\,(\mathit{bind}, \textsf{mk-}\mathit{AS}`\mathit{Name}\,([\texttt{"1"}], \mathit{CI}`\mathit{NilContextId})),$

.3 $\qquad \mathit{common} = \mathit{CompileEUandICommon}\,(\mathit{new\text{-}bind}, \mathit{pred\text{-}e}),$

.4 $\qquad \mathit{one\text{-}found} = [\textsf{mk-}\mathit{INSTRTP}`\mathit{SELBLKENV}\,(1)] \curvearrowright \mathit{E2I}\,(\mathit{PAT}`\mathit{GetExpr}\,(\mathit{PAT}`\mathit{SelPattern}\,(\mathit{new\text{-}bind}))) \curvearrowright$

.5 $\qquad\qquad\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{POPBLKENV}\,()] \textsf{ in}$

.6 $\quad \textsf{return } \mathit{common} \curvearrowright \mathit{ConcIfThenElse}\,([\textsf{mk-}\mathit{INSTRTP}`\mathit{SIZE}\,(1)],$

.7 $\qquad\qquad\qquad\qquad \mathit{one\text{-}found},$

.8 $\qquad\qquad\qquad\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{ERRINST}\,(\mathit{RTERR}`\mathit{NO\text{-}UNIQ\text{-}ELEM})]) \curvearrowright$

.9 $\qquad\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{REMSTACKELEM}\,(2), \textsf{mk-}\mathit{INSTRTP}`\mathit{REMSTACKELEM}\,(2)];$

1.2.6 Apply Expression

76.0 $\mathit{CompileApplyExpr} : \mathit{AS}`\mathit{ApplyExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{CompileApplyExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{ApplyExpr}\,(\mathit{fct}, \mathit{arg\text{-}l}, \text{-})) \triangleq$

.2 $\quad (\textsf{dcl } \mathit{sp} : \mathit{STKM}`\mathit{SubProgram} := \mathit{E2I}\,(\mathit{fct});$

.3 $\quad \mathit{sp} := \mathit{sp} \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{EMPTYLIST}\,()];$

.4 $\quad \textsf{for } \mathit{arg} \textsf{ in } \mathit{arg\text{-}l}$

.5 $\quad \textsf{do } \mathit{sp} := \mathit{sp} \curvearrowright \mathit{E2I}\,(\mathit{arg}) \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{APPENDESTCK}\,()];$

.6 $\quad \mathit{sp} := \mathit{sp} \curvearrowright$

.7 $\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{GUARD}\,()] \curvearrowright$

.8 $\qquad \mathit{CMPL}`\mathit{SetContext}\,(\mathit{fct}.\mathit{cid}, \textsf{false}) \curvearrowright$

.9 $\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{APPLY}\,()];$

.10 $\quad \textsf{return } \mathit{sp}\ );$

1.2.7 Field Select Expression

77.0 $\mathit{CompileFieldSelectExpr} : \mathit{AS}`\mathit{FieldSelectExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{CompileFieldSelectExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{FieldSelectExpr}\,(\mathit{rec\text{-}e}, \mathit{field}, \text{-})) \triangleq$

.2 $\quad \textsf{return } \mathit{E2I}\,(\mathit{rec\text{-}e}) \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{PUSH}\,(\mathit{field}),$

.3 $\qquad \textsf{mk-}\mathit{INSTRTP}`\mathit{FIELDSEL}\,(),$

.4 $\qquad \textsf{mk-}\mathit{INSTRTP}`\mathit{POLYTYPEINST}\,(\mathit{CMPL}`\mathit{GetClMod}\,())];$

1.2.8 Set Range Expression

78.0 $CompileSetRangeExpr : AS‘SetRangeExpr \xrightarrow{o} STKM‘SubProgram$
.1 $CompileSetRangeExpr\,(e) \triangleq$
.2 let mk-$AS‘SetRangeExpr\,(lb, ub, \text{-}) = e$ in
.3 (dcl $sp : STKM‘SubProgram := [\,]$;
.4 $sp := sp \curvearrowright E2I\,(lb) \curvearrowright E2I\,(ub) \curvearrowright$
.5 $TIME‘MkRuntimeSetSeqMap\,(e) \curvearrowright$
.6 [mk-$INSTRTP‘SETRNG\,()$];
.7 return sp);

1.2.9 Subsequence Expression

79.0 $CompileSubSequenceExpr : AS‘SubSequenceExpr \xrightarrow{o} STKM‘SubProgram$
.1 $CompileSubSequenceExpr\,(e) \triangleq$
.2 let mk-$AS‘SubSequenceExpr\,(seq\text{-}e, lb, ub, \text{-}) = e$ in
.3 (dcl $sp : STKM‘SubProgram := E2I\,(seq\text{-}e)$;
.4 $sp := sp \curvearrowright E2I\,(lb) \curvearrowright E2I\,(ub) \curvearrowright$
.5 $TIME‘MkRuntimeSetSeqMap\,(e) \curvearrowright$
.6 [mk-$INSTRTP‘SUBSEQ\,()$];
.7 return sp);

1.2.10 Mapping Enumeration

80.0 $CompileMapEnumExpr : AS‘MapEnumerationExpr \xrightarrow{o} STKM‘SubProgram$
.1 $CompileMapEnumExpr$ (mk-$AS‘MapEnumerationExpr\,(els\text{-}l, \text{-})) \triangleq$
.2 (dcl $sp : STKM‘SubProgram :=$ [mk-$INSTRTP‘PUSH$ (mk-$SEM‘MAP\,(\{\mapsto\})$)];
.3 for mk-$AS‘Maplet\,(mapdom, maprng, \text{-})$ in $els\text{-}l$
.4 do $sp := sp \curvearrowright E2I\,(mapdom) \curvearrowright E2I\,(maprng) \curvearrowright$[mk-$INSTRTP‘APPENDMAP\,()$];
.5 return sp);

1.2.11 Sequence Enumeration

81.0 $CompileSeqEnumExpr : AS‘SeqEnumerationExpr \xrightarrow{o} STKM‘SubProgram$
.1 $CompileSeqEnumExpr$ (mk-$AS‘SeqEnumerationExpr\,(els\text{-}l, \text{-})) \triangleq$
.2 (dcl $sp : STKM‘SubProgram :=$ [mk-$INSTRTP‘PUSH$ (mk-$SEM‘SEQ\,([\,])$)];
.3 for $elem$ in $els\text{-}l$
.4 do $sp := sp \curvearrowright E2I\,(elem) \curvearrowright$ [mk-$INSTRTP‘APPENDSEQ\,()$];
.5 return sp);

1.2.12 Set Enumeration

$$
\begin{array}{l}
\mathit{CompileSetEnumExpr} : \mathit{AS}`\mathit{SetEnumerationExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}\\
\mathit{CompileSetEnumExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{SetEnumerationExpr}\,(\mathit{els\text{-}l}, \text{-})) \triangleq\\
\quad (\textsf{dcl } \mathit{sp} : \mathit{STKM}`\mathit{SubProgram} := [\textsf{mk-}\mathit{INSTRTP}`\mathit{PUSH}\,(\textsf{mk-}\mathit{SEM}`\mathit{SET}\,(\{\}))];\\
\quad\ \textsf{for } \mathit{elem} \textsf{ in } \mathit{els\text{-}l}\\
\quad\ \textsf{do } \mathit{sp} := \mathit{sp} \curvearrowright \mathit{E2I}\,(\mathit{elem}) \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{ADDSET}\,()];\\
\quad\ \textsf{return } \mathit{sp}\ );
\end{array}
$$

1.2.13 Set Comprehension

$$
\begin{array}{l}
\mathit{CompileSetComprehensionExpr} : \mathit{AS}`\mathit{SetComprehensionExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}\\
\mathit{CompileSetComprehensionExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{SetComprehensionExpr}\,(\mathit{elem}, \mathit{bind}, \mathit{pred}, \text{-})) \triangleq\\
\quad \textsf{let } \mathit{pred\text{-}instr} = \textsf{if } \mathit{pred} = \textsf{nil}\\
\qquad\qquad\qquad\quad \textsf{then } [\textsf{mk-}\mathit{INSTRTP}`\mathit{PUSH}\,(\textsf{mk-}\mathit{SEM}`\mathit{BOOL}\,(\textsf{true}))]\\
\qquad\qquad\qquad\quad \textsf{else } \mathit{E2I}\,(\mathit{pred}) \textsf{ in}\\
\quad \textsf{let } \mathit{prep\text{-}instr} = \mathit{CPAT}`\mathit{CompileMultBindL}\,(\mathit{bind}, \mathrm{DO\_PARTITION}) \curvearrowright\\
\qquad\qquad\qquad\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{PUSH}\,(\textsf{mk-}\mathit{SEM}`\mathit{SET}\,(\{\}))],\\
\qquad \mathit{loop\text{-}instr} = \mathit{ConcIfThenElse}\,([\textsf{mk-}\mathit{INSTRTP}`\mathit{ISEMPTYSET}\,(2, \textsf{nil}\ )],\\
\qquad\qquad\qquad\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{PUSH}\,(\textsf{mk-}\mathit{SEM}`\mathit{BOOL}\,(\textsf{true}))],\\
\qquad\qquad\qquad\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{SELBLKENV}\,(2)] \curvearrowright\\
\qquad\qquad\qquad\qquad \mathit{ConcIfThenElse}\,(\mathit{pred\text{-}instr},\\
\qquad\qquad\qquad\qquad\qquad \mathit{E2I}\,(\mathit{elem}) \curvearrowright\\
\qquad\qquad\qquad\qquad\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{ADDSET}\,()] \curvearrowright\\
\qquad\qquad\qquad\qquad\qquad \mathit{TIME}`\mathit{MkSetCompInsert}\,(),\\
\qquad\qquad\qquad\qquad\qquad []) \curvearrowright\\
\qquad\qquad\qquad\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{POPBLKENV}\,()] \curvearrowright\\
\qquad\qquad\qquad\qquad [\textsf{mk-}\mathit{INSTRTP}`\mathit{PUSH}\,(\textsf{mk-}\mathit{SEM}`\mathit{BOOL}\,(\textsf{false}))]) \textsf{ in}\\
\quad \textsf{return } \mathit{prep\text{-}instr} \curvearrowright \mathit{CombRepeatUntil}\,(\mathit{loop\text{-}instr}) \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{REMSTACKELEM}\,(2)];
\end{array}
$$

1.2.14 Seqence Comprehension

The instruction code for a sequence comprehension:

```
[i | i in set {1,2}]
```

will look like:

```
mk_INSTRTP‘ISTART(
     "SeqComprehensionExpr" ),
  mk_INSTRTP‘CONTEXT( 41943063,false ),
```

```
  mk_INSTRTP`PUSH(
     mk_STKM`PatternName(
        mk_AS`Name(
           [ "i" ],
           41943057 ),
        41943058 ) ),
  .. E2I {1,2} ...

  mk_INSTRTP`SEQCOMPBIND(  ), -- Push the pattern and the sequence of the set
                              -- that we are going to traverse.
  mk_INSTRTP`PUSH(
     mk_SEM`SEQ( [  ] ) ),    -- Push the "result" of the sequence comprehension.
                                      [ mk_SEM`SEQ([]),
                                        [mk_SEM`Val(1), mk_SEM`Val(2)],
                                        mk_AS`PatternName(mk_AS`Name("i")) ]

label: loop
  mk_INSTRTP`ISEMPTYSEQ( 2,nil ), -- Check if the sequence that we traverse
                                  -- is empty. The sequence is the second
                                  -- element on the evaluation stack.
  mk_INSTRTP`UNOP( <NOT> ),
  mk_INSTRTP`CNBR( 14 ),          -- If the sequence is empty we are done,
                                  -- Jump to the end of the

  mk_INSTRTP`EMPTYBLKENV( <READ_ONLY> ), -- otherwise: Push a block environment
                                         -- for the free variables in the
                                         -- pattern.
  mk_INSTRTP`SEQELEMMATCH( 3 ),          -- Bind the pattern to the first
                                         -- element of the sequence at the
                                         -- second place in the evaluation
                                         -- stack. The binding is introduced in
                                         -- the block-environment.
                                         -- From the sequence that we traverse
                                         -- the element that we used in
                                         -- the pattern match is removed.
                                         -- Thus after the first traversion the
                                         -- evaluation stack looks like:
                                         -- [ mk_SEM`([]),
                                         --   [ mk_SEM`Val(2)],
                                         --   mk_AS`PatternName(mk_AS`Name("i"))]
  mk_INSTRTP`PUSH(                       -- Evaluate the predicate. (In this
                                         -- case always true)

     mk_SEM`BOOL( true ) ),
  mk_INSTRTP`CBR( 1 ),
  mk_INSTRTP`BR( 7 ),
  mk_INSTRTP`ISTART(                     -- Compute the expression.
     "Name" ),
  mk_INSTRTP`CONTEXT( 41943056,false ),
  mk_INSTRTP`PUSH(
```

```
   mk_AS‘Name(
      [ "i" ],
      41943056 ) ),
mk_INSTRTP‘LOOKUP(  ),
mk_INSTRTP‘POPCONTEXT( 41943056,false ),
mk_INSTRTP‘IEND(
   "Name" ),
mk_INSTRTP‘APPENDSEQ(  ),             -- and append it with the result sequence
                                      -- on the evaluation stack.
mk_INSTRTP‘POPBLKENV(  ),             -- Remove the temporary block
                                      -- environment
mk_INSTRTP‘BR( -17 ),                 -- Jump to the next traversion.
                                      -- After the first traversion the
                                      -- evaluation stack looks like:
                                      -- [ mk_SEM‘SEQ([mk_SEM‘Val(1)]),
                                      --   [ mk_SEM‘Val(1)],
                                      --   mk_AS‘PatternName(mk_AS‘("i"))]


mk_INSTRTP‘REMSTACKELEM( 2 ),         -- After the evaluation:
                                      -- The evaluation stack looks like:
                                      -- [ mk_SEM‘SEQ([mk_SEM..(1), mk_SEM..(2))],
                                      --   [],
                                      --   mk_AS‘PatternName(mk_AS‘Name("i"))].
                                      -- REMSTACKELEM removes the
                                      -- second element on the evaluation
                                      -- stack. That is, the empty sequence.

mk_INSTRTP‘REMSTACKELEM( 2 ),         -- Removes the pattern from the
                                      -- evaluation stack.
mk_INSTRTP‘POPCONTEXT( 41943063,false ),
mk_INSTRTP‘IEND(
   "SeqComprehensionExpr" ),
```

84.0 $CompileSeqComprehensionExpr : AS`SeqComprehensionExpr \xrightarrow{o} STKM`SubProgram$

.1 $CompileSeqComprehensionExpr\,(\mathsf{mk}\text{-}AS`SeqComprehensionExpr\,(elem, bind, pred, \text{-})) \triangleq$

.2 $\quad \mathsf{let}\ pred\text{-}instr = \mathsf{if}\ pred = \mathsf{nil}$

.3 $\qquad\qquad \mathsf{then}\ [\mathsf{mk}\text{-}INSTRTP`PUSH\,(\mathsf{mk}\text{-}SEM`BOOL\,(\mathsf{true}))]$

.4 $\qquad\qquad \mathsf{else}\ E2I\,(pred),$

.5 $\qquad \mathsf{mk}\text{-}AS`SetBind\,(pat, set\text{-}e, \text{-}) = bind\ \mathsf{in}$

.6 $\quad \mathsf{let}\ prep\text{-}instr =$

.7 $\qquad\quad CPAT`P2I\,(pat) \curvearrowright E2I\,(set\text{-}e) \curvearrowright$

.8 $\qquad\quad [\mathsf{mk}\text{-}INSTRTP`SEQCOMPBIND\,()] \curvearrowright$

.9 $\qquad\quad [\mathsf{mk}\text{-}INSTRTP`PUSH\,(\mathsf{mk}\text{-}SEM`SEQ\,([]))],$

.10 $\qquad condition = [\mathsf{mk}\text{-}INSTRTP`ISEMPTYSEQ\,(2, \mathsf{nil}\,), \mathsf{mk}\text{-}INSTRTP`UNOP\,(\mathrm{NOT})],$

```
    body =
          [mk-INSTRTP`EMPTYBLKENV (READ_ONLY),
           mk-INSTRTP`SEQELEMMATCH (3)] ⌒
          ConcIfThenElse (pred-instr,
                          E2I (elem) ⌒
                          [mk-INSTRTP`APPENDSEQ ()] ⌒
                          TIME`MkSeqCompInsert (),
                          []) ⌒
          [mk-INSTRTP`POPBLKENV ()],
    clean-stack = [mk-INSTRTP`REMSTACKELEM (2), mk-INSTRTP`REMSTACKELEM (2)] in
let res =
          prep-instr ⌒
          CombWhileLoopWithCond (condition, body) ⌒
          clean-stack in
return res;
```

1.2.15 Map Comprehension

```
85.0 CompileMapComprehensionExpr : AS`MapComprehensionExpr -o-> STKM`SubProgram
CompileMapComprehensionExpr (mk-AS`MapComprehensionExpr (elem, bind, pred, -)) △
  let pred-instr = if pred = nil
                   then [mk-INSTRTP`PUSH (mk-SEM`BOOL (true))]
                   else E2I (pred) in
  let prep-instr = CPAT`CompileMultBindL (bind, DO_PARTITION) ⌒
                   [mk-INSTRTP`PUSH (mk-SEM`MAP ({↦}))],
      loop-instr = ConcIfThenElse ([mk-INSTRTP`ISEMPTYSET (2, nil )],
                                   [mk-INSTRTP`PUSH (mk-SEM`BOOL (true))],
                                   [mk-INSTRTP`SELBLKENV (2)] ⌒
                                   ConcIfThenElse (pred-instr,
                                                   E2I (elem.mapdom) ⌒
                                                   E2I (elem.maprng) ⌒
                                                   [mk-INSTRTP`APPENDMAP ()] ⌒

                                                   TIME`MkMapCompInsert (),
                                                   []) ⌒
                                   [mk-INSTRTP`POPBLKENV ()] ⌒
                                   [mk-INSTRTP`PUSH (mk-SEM`BOOL (false))]) in
  return prep-instr ⌒ CombRepeatUntil (loop-instr) ⌒
         [mk-INSTRTP`REMSTACKELEM (2)];
```

1.2.16 Tuple Expressions

86.0 $\mathit{CompileTupleConstructorExpr} : \mathit{AS}`\mathit{TupleConstructorExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{CompileTupleConstructorExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{TupleConstructorExpr}\,(\mathit{fields}, \textsf{-})) \triangleq$
.2 $(\textsf{dcl}\ \mathit{sp} : \mathit{STKM}`\mathit{SubProgram} := [\textsf{mk-}\mathit{INSTRTP}`\mathit{PUSH}\,(\textsf{mk-}\mathit{SEM}`\mathit{TUPLE}\,([]))];$
.3 $\textsf{for}\ \mathit{elem}\ \textsf{in}\ \mathit{fields}$
.4 $\textsf{do}\ \mathit{sp} := \mathit{sp} \curvearrowright \mathit{E2I}\,(\mathit{elem}) \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{APPENDTUP}\,()];$
.5 $\textsf{return}\ \mathit{sp}\ );$

1.2.17 Record Expressions

87.0 $\mathit{CompileRecordConstructorExpr} : \mathit{AS}`\mathit{RecordConstructorExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{CompileRecordConstructorExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{RecordConstructorExpr}\,(\mathit{tag}, \mathit{fields}, \textsf{-})) \triangleq$
.2 $(\textsf{dcl}\ \mathit{sp} : \mathit{STKM}`\mathit{SubProgram} := [];$
.3 $\textsf{for}\ \mathit{elem}\ \textsf{in}\ \mathit{fields}$
.4 $\textsf{do}\ \mathit{sp} := \mathit{sp} \curvearrowright \mathit{E2I}\,(\mathit{elem});$
.5 $\mathit{sp} := \mathit{sp} \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{RECCONS}\,(\mathit{tag}, \textsf{len}\ \mathit{fields})];$
.6 $\textsf{return}\ \mathit{sp}\ );$

88.0 $\mathit{CompileRecordModifierExpr} : \mathit{AS}`\mathit{RecordModifierExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{CompileRecordModifierExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{RecordModifierExpr}\,(\mathit{rec\text{-}e}, \mathit{modifiers}, \textsf{-})) \triangleq$
.2 $(\textsf{dcl}\ \mathit{sp} : \mathit{STKM}`\mathit{SubProgram} := \mathit{E2I}\,(\mathit{rec\text{-}e});$
.3 $\textsf{for}\ \textsf{mk-}\mathit{AS}`\mathit{RecordModification}\,(\mathit{field}, \mathit{new}, \textsf{-})\ \textsf{in}\ \mathit{modifiers}$
.4 $\textsf{do}\ \mathit{sp} := \mathit{sp} \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{PUSH}\,(\mathit{field})] \curvearrowright \mathit{E2I}\,(\mathit{new});$
.5 $\mathit{sp} := \mathit{sp} \curvearrowright [\textsf{mk-}\mathit{INSTRTP}`\mathit{RECMOD}\,(2 \times \textsf{len}\ \mathit{modifiers})];$
.6 $\textsf{return}\ \mathit{sp}\ );$

1.2.18 Sequence and Map Modifier Expression

89.0 $\mathit{CompileSeqModifyMapOverrideExpr} : \mathit{AS}`\mathit{SeqModifyMapOverrideExpr} \xrightarrow{o}$
.1 $\mathit{STKM}`\mathit{SubProgram}$
.2 $\mathit{CompileSeqModifyMapOverrideExpr}\,(e) \triangleq$
.3 $\textsf{let}\ \textsf{mk-}\mathit{AS}`\mathit{SeqModifyMapOverrideExpr}\,(\mathit{sm\text{-}e}, \mathit{m\text{-}e}, \textsf{-}) = e\ \textsf{in}$
.4 $\textsf{return}\ \mathit{E2I}\,(\mathit{sm\text{-}e}) \curvearrowright \mathit{E2I}\,(\mathit{m\text{-}e}) \curvearrowright$
.5 $\mathit{TIME}`\mathit{MkRuntimeSetSeqMap}\,(e) \curvearrowright$
.6 $[\textsf{mk-}\mathit{INSTRTP}`\mathit{SEQMAPOVER}\,()]$

1.2.19 If-Then-Else Expression

To be able to evaluate conditional expressions (amongst others) in the context of a stack machine we must have an instruction that allows the program to branch — perform jumps. Therefore we introduce the conditional branch instruction `CBR(n)`. The instruction pops one item of the stack and branches according to the boolean value of this item. If *true* the instuction increments the PC with the value of the argument for the instruction n, and consequently it branches relatively to the PC with this offset. If the item popped of the stack is *false* the program evaluation continues unaffected with the next instruction.

Furthermore it seems natural to add an instruction to perform an unconditional branch. The instruction `BR` pops one item, the offset, of the stack and branches unconditionally with this offset.

Using the branch instructions just introduced, the following if expression

if *cond_expr* then *expr_1* else *expr_2*

would be translated to the following (pseudo) stack instructions:

```
[cond_expr, CBR(len expr_2 + 1),
 expr_2, BR(len expr_1 + 1), expr_1 ]
```

where `cond_expr`, `expr_1` and `expr_2` are the complete stack instructions for the coresponding expressions *cond_expr*, *expr_1* and *expr_2*. Furthermore `len` is used to denote the length of an instruction sequence to compute the offset for the branch instructions.

For a more concrete example, the following if-then-else expression

if 3+7 = a then 3+42*5 else 17

will result in these instructions

```
[ 3, 7, ADD, 'a', LOOKUP, EQUAL, CBR(2),
17, BR(8),
3, 42, 5, MUL, ADD]
```

The following table shows the evolution of the stack and PC as the program is interpreted (assuming that the identifier 'a' equals 42):

Stack	PC
-	1
8	2
8,3	3
8,3,7	4
8,10	5
8,10,'a'	6
8,10,42	7
8,FALSE	8
-	16
17	-

If-then-elseif expressions can be handled by treating the *elseif* part of the expression as an if-the-else expression.

functions

```
90.0 CompileIfExpr : AS`IfExpr → STKM`SubProgram
  .1 CompileIfExpr (e) △
  .2   let mk-AS`IfExpr (cond-e, expr1-e, elif-l, altn-e, -) = e,
  .3       cond-l = E2I (cond-e),
  .4       expr1-l =
  .5           E2I (expr1-e) ⁀
  .6           TIME`MkCbr () in
  .7   if len elif-l = 0
  .8   then ConcIfThenElse (cond-l,
  .9                        expr1-l,
 .10                        E2I (altn-e) ⁀
 .11                        TIME`MkBr ())
 .12   else let mk-AS`ElseifExpr (elif-cond, elif-expr, elif-cid) = hd elif-l,
 .13            altn-l =
 .14                E2I (mk-AS`IfExpr
 .15                        (
 .16                         elif-cond,
 .17                         elif-expr,
 .18                         tl elif-l,
 .19                         altn-e,
 .20                         elif-cid)) ⁀
 .21                TIME`MkCbr () in
 .22        ConcIfThenElse (cond-l, expr1-l, altn-l);
```

```
91.0 ConcIfThenElse : STKM`SubProgram × STKM`SubProgram × STKM`SubProgram
  .1                  → STKM`SubProgram
  .2 ConcIfThenElse (cond-l, exp1-l, altn-l) △
  .3   cond-l ⁀ [mk-INSTRTP`CBR (len altn-l + 1)] ⁀
  .4   altn-l ⁀ [mk-INSTRTP`BR (len exp1-l)] ⁀
  .5   exp1-l
```

1.2.20 Cases Expression

operations

```
92.0 CompileCasesExpr : AS`CasesExpr ⥲ STKM`SubProgram
  .1 CompileCasesExpr (mk-AS`CasesExpr (sel, altns, others-e, -)) △
  .2   (dcl sp : STKM`SubProgram := if others-e = nil
  .3                    then [mk-INSTRTP`ERRINST (RTERR`NO-OTHERS-EXPR)]
  .4                    else E2I (others-e) ⁀
  .5                         TIME`MkBr ();
```

.6 for mk-$AS`CaseAltn\,(match\text{-}lp, body\text{-}e, \text{-})$ in $altns$
.7 do let $body\text{-}instr =$
.8 $E2I\,(body\text{-}e) \curvearrowright$
.9 $TIME`MkCbr\,()$ in
.10 for pat in $match\text{-}lp$
.11 do let $test\text{-}instr =$
.12 [mk-$INSTRTP`COPYVAL\,()] \curvearrowright$
.13 $CPAT`P2I\,(pat) \curvearrowright$
.14 [mk-$INSTRTP`TRYANYMATCH\,()] \curvearrowright$
.15 $TIME`MkMatchPattern\,()$ in
.16 $(sp := sp \curvearrowright TIME`MkBr\,();$
.17 $sp := ConcIfThenElse\,(test\text{-}instr, body\text{-}instr, sp));$
.18 return [mk-$INSTRTP`EMPTYBLKENV\,(\mathrm{READ\_ONLY})] \curvearrowright$
.19 $E2I\,(sel) \curvearrowright$
.20 $sp \curvearrowright$[mk-$INSTRTP`POPBLKENV\,()$, mk-$INSTRTP`REMSTACKELEM\,(2)]$
)

1.2.21 Unary Expressions

functions

93.0 $CompileUnaryExpr : AS`PrefixExpr \rightarrow STKM`SubProgram$
.1 $CompileUnaryExpr\,($mk-$AS`PrefixExpr\,(opr, arg\text{-}e, \text{-})) \triangleq$
.2 if $TIME`IsRuntimePrefixOp\,(opr)$
.3 then $E2I\,(arg\text{-}e) \curvearrowright TIME`MkRuntimePrefixOp\,(opr) \curvearrowright$[mk-$INSTRTP`UNOP\,(opr)]$
.4 else $E2I\,(arg\text{-}e) \curvearrowright$ [mk-$INSTRTP`UNOP\,(opr)];$

94.0 $CompileMapInverseExpr : AS`MapInverseExpr \rightarrow STKM`SubProgram$
.1 $CompileMapInverseExpr\,($mk-$AS`MapInverseExpr\,(op, \text{-})) \triangleq$
.2 $E2I\,(op) \curvearrowright$ [mk-$INSTRTP`UNOP\,(\mathrm{MAPINVERSE})];$

1.2.22 Binary Expressions

```
95.0 CompileBinaryExpr : AS`BinaryExpr → STKM`SubProgram
     CompileBinaryExpr (mk-AS`BinaryExpr (left-e, opr, right-e, cid)) △
       cases opr :
         AND,
         OR,
         IMPLY → CompileLogBinaryExpr (left-e, opr, right-e),
         others → cases opr :
                    EQ,
                    NE,
                    EQUIV,
                    NUMPLUS,
                    NUMMINUS,
                    NUMMULT,
                    NUMDIV,
                    NUMREM,
                    INTDIV,
                    NUMLT,
                    NUMLE,
                    NUMGT,
                    NUMGE,
                    NUMMOD,
                    INSET,
                    NOTINSET,
                    SETUNION,
                    SETINTERSECT,
                    SETMINUS,
                    SUBSET,
                    PROPERSUBSET,
                    SEQCONC,
                    MAPMERGE,
                    MAPDOMRESTTO,
                    MAPDOMRESTBY,
                    MAPRNGRESTTO,
                    MAPRNGRESTBY,
                    COMPOSE,
                    NUMEXP → CompileOrdinaryBinaryExpr (left-e, opr, right-e, cid),
                    others → undefined
                  end
       end;
```

96.0 $\mathit{CompileLogBinaryExpr} : \mathit{AS}`\mathit{Expr} \times \mathit{AS}`\mathit{BinaryOp} \times \mathit{AS}`\mathit{Expr} \to \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{CompileLogBinaryExpr}\,(\mathit{left\text{-}e}, \mathit{op}, \mathit{right\text{-}e}) \triangleq$
.2 let $\mathit{sp\text{-}left} = \mathit{E2I}\,(\mathit{left\text{-}e})$,
.3 $\mathit{sp\text{-}right} = \mathit{E2I}\,(\mathit{right\text{-}e})$,
.4 $\mathit{def\text{-}true} = [$mk-$\mathit{INSTRTP}`\mathit{PUSH}\,($mk-$\mathit{SEM}`\mathit{BOOL}\,($true$))]$,
.5 $\mathit{def\text{-}false} = [$mk-$\mathit{INSTRTP}`\mathit{PUSH}\,($mk-$\mathit{SEM}`\mathit{BOOL}\,($false$))]$ in
.6 cases op :
.7 $\mathrm{AND} \to \mathit{ConcIfThenElse}\,(\mathit{sp\text{-}left}, \mathit{sp\text{-}right}, \mathit{def\text{-}false})$,
.8 $\mathrm{OR} \to \mathit{ConcIfThenElse}\,(\mathit{sp\text{-}left}, \mathit{def\text{-}true}, \mathit{sp\text{-}right})$,
.9 $\mathrm{IMPLY} \to \mathit{ConcIfThenElse}\,(\mathit{sp\text{-}left}, \mathit{sp\text{-}right}, \mathit{def\text{-}true})$
.10 end;

97.0 $\mathit{CompileOrdinaryBinaryExpr} : \mathit{AS}`\mathit{Expr} \times \mathit{AS}`\mathit{BinaryOp} \times \mathit{AS}`\mathit{Expr} \times \mathit{CI}`\mathit{ContextId} \to \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{CompileOrdinaryBinaryExpr}\,(\mathit{left\text{-}e}, \mathit{opr}, \mathit{right\text{-}e}, \text{-}) \triangleq$
.2 let $\mathit{left\text{-}l} = \mathit{E2I}\,(\mathit{left\text{-}e})$,
.3 $\mathit{right\text{-}l} = \mathit{E2I}\,(\mathit{right\text{-}e})$ in
.4 if $\mathit{TIME}`\mathit{IsRuntimeBinaryOp}\,(\mathit{opr})$
.5 then $\mathit{left\text{-}l} \curvearrowright \mathit{right\text{-}l} \curvearrowright \mathit{TIME}`\mathit{MkRuntimeBinaryOp}\,(\mathit{opr}) \curvearrowright$
.6 $[$mk-$\mathit{INSTRTP}`\mathit{BINOP}\,(\mathit{opr})]$
.7 else $\mathit{left\text{-}l} \curvearrowright \mathit{right\text{-}l} \curvearrowright [$mk-$\mathit{INSTRTP}`\mathit{BINOP}\,(\mathit{opr})]$

1.2.23 Token Constructor Expression

operations

98.0 $\mathit{CompileTokenConstructorExpr} : \mathit{AS}`\mathit{TokenConstructorExpr} \xrightarrow{o}$
.1 $\mathit{STKM}`\mathit{SubProgram}$

.2 $\mathit{CompileTokenConstructorExpr}\,($mk-$\mathit{AS}`\mathit{TokenConstructorExpr}\,(\mathit{expr}, \text{-})) \triangleq$
.3 return $\mathit{E2I}\,(\mathit{expr}) \curvearrowright [$mk-$\mathit{INSTRTP}`\mathit{TOKENVAL}\,()]$;

1.2.24 Tuple Selection Expressions

99.0 $\mathit{CompileTupleSelectExpr} : \mathit{AS}`\mathit{TupleSelectExpr} \xrightarrow{o} \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{CompileTupleSelectExpr}\,($mk-$\mathit{AS}`\mathit{TupleSelectExpr}\,(\mathit{tuple}, \mathit{no}, \text{-})) \triangleq$
.2 return $\mathit{E2I}\,(\mathit{tuple}) \curvearrowright [$mk-$\mathit{INSTRTP}`\mathit{TUPSEL}\,(\mathit{no}.\mathit{val})]$;

1.2.25 Type Judgements

```
100.0 CompileTypeJudgementExpr : AS`TypeJudgementExpr →o STKM`SubProgram
      CompileTypeJudgementExpr (mk-AS`TypeJudgementExpr (ex, tp, -)) ≜
        return E2I (ex) ⁀ [mk-INSTRTP`TYPEJUDGE (tp)];
```

1.2.26 Pre-condition Application Expressions

```
101.0 CompilePreConditionApplyExpr : AS`PreConditionApplyExpr →o STKM`SubProgram
      CompilePreConditionApplyExpr (mk-AS`PreConditionApplyExpr (fct, arg, -)) ≜
        return [mk-INSTRTP`ERRINST (RTERR`PRE-COND-APPLY-EXPR)];
```

1.2.27 Lambda Expressions

This operation returns a semantic function value that represents the given lambda expression. A closure environment is created to preserve the bindings for the free variables in the body of the lambda expression. The range type of the function denotes all possible types.

```
102.0 CompileLambdaExpr : AS`LambdaExpr →o STKM`SubProgram
      CompileLambdaExpr (mk-AS`LambdaExpr (type-l, body, -)) ≜
        (dcl id-s : AS`Name-set := {},
             pat-l : AS`Pattern* := [],
             tp-l : AS`Type* := [];
         for mk-AS`TypeBind (pat, tp, -) in type-l
         do (id-s := id-s ∪ FREE`IdentInPattern (pat);
             pat-l := pat-l ⁀ [pat];
             tp-l := tp-l ⁀ [tp]);
         let clmodName = CMPL`GetClMod (),
             body-prog = CMPL`CompileLambdaBody (body),
             pi-l = [CPAT`P2P (pat-l (i)) | i ∈ inds pat-l] in
         return [mk-INSTRTP`PUSH
                    (
                     SEM`CompFN
                        (
                         mk-SEM`ExplFN (mk-AS`LambdaFnType (tp-l, mk-AS`AllType ()),
                                        [pi-l], body-prog,
                                        mk-SEM`BlkEnv ({↦}, READ_ONLY), {↦},
                                        nil , clmodName, nil , nil ))),
                 mk-INSTRTP`UPDATECLOSENV (body)] );
```

1.2.28 Polymorphic Function Instantiation

103.0 $CompileFctTypeInstExpr : AS`FctTypeInstExpr \xrightarrow{o} STKM`SubProgram$

.1 $CompileFctTypeInstExpr\,(\textsf{mk-}AS`FctTypeInstExpr\,(polyfct, inst, \textsf{-})) \triangleq$

.2 $\quad \textsf{return}\ [\textsf{mk-}INSTRTP`LOOKUP\,(polyfct),$

.3 $\qquad\qquad \textsf{mk-}INSTRTP`POLYTYPEINST\,(CMPL`GetClMod\,()), \textsf{mk-}INSTRTP`POLYINST\,(inst)];$

1.2.29 Is Expression

104.0 $CompileIsExpr : AS`IsExpr \xrightarrow{o} STKM`SubProgram$

.1 $CompileIsExpr\,(\textsf{mk-}AS`IsExpr\,(Type, arg\text{-}e, \textsf{-})) \triangleq$

.2 $\quad \textsf{let}\ renType =$

.3 $\qquad\qquad Type\ \textsf{in}$

.4 $\quad \textsf{return}\ E2I\,(arg\text{-}e) \curvearrowright [\textsf{mk-}INSTRTP`ISCHECK\,(renType)];$

105.0 $CompileSelfExpr : AS`SelfExpr \xrightarrow{o} STKM`SubProgram$

.1 $CompileSelfExpr\,(\textsf{mk-}AS`SelfExpr\,(\textsf{-})) \triangleq$

.2 $\quad \textsf{return}\ [\textsf{mk-}INSTRTP`SELFEXPR\,()];$

There is a lot of different ways in which the new-expressions can be used and while debugging it is worth to notice that there are a number of rules which are followed for the evaluation and the value stack for the different cases. These can be summarised as:

1. `new A()` Without any constructor in A this must leave one value on top of the evaluation stack (the reference to this new A object).
2. `new A(7)` When A has a constructor it must leave exactly the same on top of the evaluation stack (note that the constructor makes a major difference in the way it gets evaluated).
3. `new A().op()` Without constructor this expression must leave the result of evaluating the op operation only on top of the evaluation stack.
4. `new A(7).op()` With a constructor this expression must just like above only leave the operation result on top of the stack. Note that because of the constructor the expression will be evaluated by first creating the instance, then calling the constructor and finally calling the operation op.
5. `new A(7).op(9)` Same as above. The complications are related to when the arguments for the constuctor and the operation are taken from the evaluation stack. This is further complicated by the presence of overloading.
6. Note that in all cases above when a new instance of a class is being created, all instance variables in all superclasses must also be initialised. Each of these can also have constructors. When each of these initialisations are completed they must not leave any values on the evaluation stack. In this initialisation the statically declared instance variables must also be looked up.

106.0 $CompileNewExpr : AS`NewExpr \times [\mathsf{token}] \xrightarrow{o} STKM`SubProgram$
.1 $CompileNewExpr\,(\mathsf{mk\text{-}}AS`NewExpr\,(nm, exprs, \text{-}), dlobject) \triangleq$
.2 $(\mathsf{dcl}\ sp : STKM`SubProgram := [\,];$
.3 $\mathsf{for}\ expr\ \mathsf{in}\ exprs$
.4 $\mathsf{do}\ sp := sp \curvearrowright E2I\,(expr);$
.5 $sp := sp \curvearrowright [\mathsf{mk\text{-}}INSTRTP`NEWOBJ\,(nm, dlobject)];$
.6 $\mathsf{return}\ sp \curvearrowright [\mathsf{mk\text{-}}INSTRTP`INITCLASS\,(nm, \mathsf{len}\ exprs),$
.7 $\mathsf{mk\text{-}}INSTRTP`NEWCOMPL\,(\mathsf{true})] \curvearrowright$
.8 $\mathsf{if}\ exprs \neq [\,]$
.9 $\mathsf{then}\ [\mathsf{mk\text{-}}INSTRTP`POP\,(1)]$
.10 $\mathsf{else}\ [\,]\ );$

107.0 $CompileIsOfClassExpr : AS`IsOfClassExpr \xrightarrow{o} STKM`SubProgram$
.1 $CompileIsOfClassExpr\,(\mathsf{mk\text{-}}AS`IsOfClassExpr\,(clnm, arg, \text{-})) \triangleq$
.2 $\mathsf{return}\ E2I\,(arg) \curvearrowright [\mathsf{mk\text{-}}INSTRTP`ISOFCLASS\,(clnm)];$

108.0 $CompileIsOfBaseClassExpr : AS`IsOfBaseClassExpr \xrightarrow{o} STKM`SubProgram$
.1 $CompileIsOfBaseClassExpr\,(\mathsf{mk\text{-}}AS`IsOfBaseClassExpr\,(clnm, arg, \text{-})) \triangleq$
.2 $\mathsf{return}\ E2I\,(arg) \curvearrowright [\mathsf{mk\text{-}}INSTRTP`ISOFBASECLASS\,(clnm)];$

109.0 $CompileSameBaseClassExpr : AS`SameBaseClassExpr \xrightarrow{o} STKM`SubProgram$
.1 $CompileSameBaseClassExpr\,(\mathsf{mk\text{-}}AS`SameBaseClassExpr\,(expr1, expr2, \text{-})) \triangleq$
.2 $\mathsf{return}\ E2I\,(expr1) \curvearrowright E2I\,(expr2) \curvearrowright [\mathsf{mk\text{-}}INSTRTP`SAMEBASECLASS\,()];$

110.0 $CompileSameClassExpr : AS`SameClassExpr \xrightarrow{o} STKM`SubProgram$
.1 $CompileSameClassExpr\,(\mathsf{mk\text{-}}AS`SameClassExpr\,(expr1, expr2, \text{-})) \triangleq$
.2 $\mathsf{return}\ E2I\,(expr1) \curvearrowright E2I\,(expr2) \curvearrowright [\mathsf{mk\text{-}}INSTRTP`SAMECLASS\,()];$

111.0 $CompileThreadIdExpr : AS`ThreadIdExpr \xrightarrow{o} STKM`SubProgram$
.1 $CompileThreadIdExpr\,(\mathsf{mk\text{-}}AS`ThreadIdExpr\,(\text{-})) \triangleq$
.2 $\mathsf{return}\ [\mathsf{mk\text{-}}INSTRTP`THREADID\,()];$

112.0 $CompileHistoryExpr : AS`Name^* \times INSTRTP`HistoryKind \xrightarrow{o} STKM`SubProgram$

.1 $CompileHistoryExpr\,(mthds, tp) \triangleq$

.2 let $qualified\text{-}names = [$let $nm = mthds\,(i)$,

.3 mk-$AS`Name\,(ids, cid) = nm$ in

.4 if len $ids = 2$

.5 then nm

.6 else $AUX`ConstructDoubleName\,(CMPL`GetClMod\,(), nm) \mid$

.7 $i \in$ inds $mthds]$ in

.8 return [mk-$INSTRTP`HISTORY\,(tp, qualified\text{-}names)$]

end $CEXPR$

Test Suite : rtinfo.ast
Module : CEXPR

Name	#Calls	Coverage
CEXPR`E2I	**undefined**	**undefined**
CEXPR`CombWhileLoop	**undefined**	**undefined**
CEXPR`CompileIfExpr	**undefined**	**undefined**
CEXPR`CompileIsExpr	**undefined**	**undefined**
CEXPR`CompileDefExpr	**undefined**	**undefined**
CEXPR`CompileLetExpr	**undefined**	**undefined**
CEXPR`CompileNewExpr	**undefined**	**undefined**
CEXPR`ConcIfThenElse	**undefined**	**undefined**
CEXPR`CombRepeatUntil	**undefined**	**undefined**
CEXPR`CompileIotaExpr	**undefined**	**undefined**
CEXPR`CompileSelfExpr	**undefined**	**undefined**
CEXPR`CompileApplyExpr	**undefined**	**undefined**
CEXPR`CompileCasesExpr	**undefined**	**undefined**
CEXPR`CompileUnaryExpr	**undefined**	**undefined**
CEXPR`CompileBinaryExpr	**undefined**	**undefined**
CEXPR`CompileLambdaExpr	**undefined**	**undefined**
CEXPR`CompileHistoryExpr	**undefined**	**undefined**
CEXPR`CompileLetBeSTExpr	**undefined**	**undefined**
CEXPR`CompileMapEnumExpr	**undefined**	**undefined**
CEXPR`CompileSeqEnumExpr	**undefined**	**undefined**
CEXPR`CompileSetEnumExpr	**undefined**	**undefined**
CEXPR`CompileEUandICommon	**undefined**	**undefined**
CEXPR`CompileSetRangeExpr	**undefined**	**undefined**
CEXPR`CompileThreadIdExpr	**undefined**	**undefined**
CEXPR`CompileIsOfClassExpr	**undefined**	**undefined**
CEXPR`CompileLogBinaryExpr	**undefined**	**undefined**
CEXPR`CompileSameClassExpr	**undefined**	**undefined**
CEXPR`CombWhileLoopWithCond	**undefined**	**undefined**
CEXPR`CompileMapInverseExpr	**undefined**	**undefined**
CEXPR`CompileAllOrExistsExpr	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
CEXPR‘CompileFctTypeInstExpr	**undefined**	**undefined**
CEXPR‘CompileFieldSelectExpr	**undefined**	**undefined**
CEXPR‘CompileSubSequenceExpr	**undefined**	**undefined**
CEXPR‘CompileTupleSelectExpr	**undefined**	**undefined**
CEXPR‘CompileExistsUniqueExpr	**undefined**	**undefined**
CEXPR‘CompileIsOfBaseClassExpr	**undefined**	**undefined**
CEXPR‘CompileSameBaseClassExpr	**undefined**	**undefined**
CEXPR‘CompileTypeJudgementExpr	**undefined**	**undefined**
CEXPR‘CompileOrdinaryBinaryExpr	**undefined**	**undefined**
CEXPR‘CompileRecordModifierExpr	**undefined**	**undefined**
CEXPR‘CompileMapComprehensionExpr	**undefined**	**undefined**
CEXPR‘CompileSeqComprehensionExpr	**undefined**	**undefined**
CEXPR‘CompileSetComprehensionExpr	**undefined**	**undefined**
CEXPR‘CompileTokenConstructorExpr	**undefined**	**undefined**
CEXPR‘CompileTupleConstructorExpr	**undefined**	**undefined**
CEXPR‘CompilePreConditionApplyExpr	**undefined**	**undefined**
CEXPR‘CompileRecordConstructorExpr	**undefined**	**undefined**
CEXPR‘CompileSeqModifyMapOverrideExpr	**undefined**	**undefined**
Total Coverage		**0%**

1.3 Compilation of Statements

This module specifies how ASTs representing statements are translated into the stack machine instructions defined in module *STKM*.
module *CSTMT*

imports

113.0 from AS all ,

114.0 from CI all ,

115.0 from IO all ,

116.0 from PAT all ,

117.0 from REP all ,

118.0 from SEM all ,

119.0 from $CMPL$ all ,

120.0 from $CPAT$ all ,

121.0 from $FREE$ all ,

122.0 from $STKM$ all ,

123.0 from $TIME$

124.0 functions $MkBr : () \to STKM\text{‘}SubProgram;$
.1 $MkCbr : () \to STKM\text{‘}SubProgram;$
.2 $S2Time : AS\text{‘}Stmt \to STKM\text{‘}SubProgram;$
.3 $MkLoopBind : \mathbb{N} \to STKM\text{‘}SubProgram;$
.4 $MkMatchPattern : () \to STKM\text{‘}SubProgram;$
.5 $MkRuntimeStartList : () \to STKM\text{‘}SubProgram$

125.0 operations $GetCompilingTime : () \xrightarrow{o} \mathbb{B}$,

126.0 from $CEXPR$ all ,

127.0 from $RTERR$ all ,

128.0 from $GLOBAL$ all ,

129.0 from $SCHDTP$ all ,

130.0 from $INSTRTP$ all ,

131.0 from $TIMEMAP$

132.0 types $Timemap$

exports all

definitions

133.0 state $TRAP$ of
.1 $trapno : \mathbb{N}$
.2 init $trapst \triangleq trapst =$ mk-$TRAP\,(0)$
.3 end

operations

134.0 $GetNewTrapNo : () \xrightarrow{o} \mathbb{N}_1$
.1 $GetNewTrapNo\,() \triangleq$
.2 $(trapno := trapno + 1;$
.3 return $trapno$);

```
135.0 S2I : AS`Stmt -o-> STKM`SubProgram
S2I (stmt) △
  let mk- (name, stmt-l) =
          cases true :
            (is-AS`DefStmt (stmt)) → mk- ("DefStmt", CompileDefStmt (stmt)),
            (is-AS`LetStmt (stmt)) → mk- ("LetStmt", CompileLetStmt (stmt)),
            (is-AS`LetBeSTStmt (stmt)) → mk- ("LetBeSTStmt", CompileLetBeSTStmt (stmt)),
            (is-AS`AssignStmt (stmt)) → mk- ("AssignStmt", CompileAssignStmt (stmt)),
            (is-AS`AtomicAssignStmt (stmt)) → mk- ("AtomicAssignStmt", CompileAtomicAssignStmt (stmt
            (is-AS`SeqForLoopStmt (stmt)) → mk- ("SeqForLoopStmt", CompileSeqForLoopStmt (stmt)),
            (is-AS`SetForLoopStmt (stmt)) → mk- ("SetForLoopStmt", CompileSetForLoopStmt (stmt)),
            (is-AS`IndexForLoopStmt (stmt)) → mk- ("IndexForLoopStmt", CompileIndexForLoopStmt (stmt)
            (is-AS`WhileLoopStmt (stmt)) → mk- ("WhileLoopStmt", CompileWhileLoopStmt (stmt)),
            (is-AS`CallStmt (stmt)) → mk- ("CallStmt", CompileCallStmt (stmt)),
            (is-AS`ReturnStmt (stmt)) → mk- ("ReturnStmt", CompileReturnStmt (stmt)),
            (is-AS`IfStmt (stmt)) → mk- ("IfStmt", CompileIfStmt (stmt)),
            (is-AS`CasesStmt (stmt)) → mk- ("CasesStmt", CompileCasesStmt (stmt)),
            (is-AS`ErrorStmt (stmt)) → mk- ("ErrorStmt", CompileErrorStmt (stmt)),
            (is-AS`ExitStmt (stmt)) → mk- ("ExitStmt", CompileExitStmt (stmt)),
            (is-AS`AlwaysStmt (stmt)) → mk- ("AlwaysStmt", CompileAlwaysStmt (stmt)),
            (is-AS`TrapStmt (stmt)) → mk- ("TrapStmt", CompileTrapStmt (stmt)),
            (is-AS`RecTrapStmt (stmt)) → mk- ("RecTrapStmt", CompileRecTrapStmt (stmt)),
            (is-AS`StartStmt (stmt)) → mk- ("StartStmt", CompileStartStmt (stmt)),
            (is-AS`StartListStmt (stmt)) → mk- ("StartListStmt", CompileStartListStmt (stmt)),
            (is-AS`DurationStmt (stmt)) → mk- ("DurationStmt", CompileDurationStmt (stmt)),
            (is-AS`BlockStmt (stmt)) → mk- ("BlockStmt", CompileBlockStmt (stmt)),
            (is-AS`NonDetStmt (stmt)) → mk- ("NonDetStmt", CompileNonDetStmt (stmt)),
            (is-AS`IdentStmt (stmt)) → mk- ("Ident", [mk-INSTRTP`PUSH (mk-SEM`CONT ())]),
            (is-AS`SpecificationStmt (stmt)) → mk- ("SpecificationStmt", [mk-INSTRTP`ERRINST (RTERR`
            others → undefined
          end in
  return CMPL`IStart (name, stmt.cid) ⁀
         CMPL`SetContext (stmt.cid, true) ⁀
         stmt-l ⁀
         CMPL`IEnd (name) ⁀
         (if TIME`GetCompilingTime ()
          then TIME`S2Time (stmt)
          else []);
```

1.3.1 Def Statement

```
136.0 CompileDefStmt : AS`DefStmt -o-> STKM`SubProgram
CompileDefStmt (mk-AS`DefStmt (def-l, In, -)) △
  (dcl sp : STKM`SubProgram := [mk-INSTRTP`EMPTYBLKENV (READ_ONLY)];
   for mk- (pb, expr) in def-l
```

```
.4      do let p-instr = CPAT`PB2I (pb) in
.5          sp := sp ⌢ CEXPR`E2I (expr) ⌢ p-instr ⌢
.6                  [mk-INSTRTP`MATCHANDBIND ()];
.7      sp := sp ⌢ S2I (In) ⌢ [mk-INSTRTP`POPBLKENV ()];
.8      return sp );
```

1.3.2 Let Statement

```
137.0   CompileLetStmt : AS`LetStmt -o-> STKM`SubProgram
.1      CompileLetStmt (mk-AS`LetStmt (localdef, In, -)) ≜
.2       (dcl sp : STKM`SubProgram := [mk-INSTRTP`EMPTYBLKENV (READ_ONLY)];
.3        for elm in localdef
.4        do if is-AS`ExplFnDef (elm) ∨ is-AS`ImplFnDef (elm) ∨ is-AS`ExtExplFnDef (elm)
.5           then let mk- (blkenv, b-m) = PAT`ConstructFN (elm) in
.6                  sp := sp ⌢ [mk-INSTRTP`CLOSENV (blkenv, b-m)]
.7           else let dtc = if elm.tp = nil
.8                          then []
.9                          else [mk-INSTRTP`DTC (elm.tp)] in
.10                sp := sp ⌢ CEXPR`E2I (elm.val) ⌢ dtc ⌢ CPAT`P2I (elm.pat) ⌢
.11                      [mk-INSTRTP`MATCHANDBIND ()];
.12       sp := sp ⌢ S2I (In) ⌢ [mk-INSTRTP`POPBLKENV ()];
.13       return sp );
```

1.3.3 Let be such that Expression

```
138.0   CompileLetBeSTStmt : AS`LetBeSTStmt -o-> STKM`SubProgram
.1      CompileLetBeSTStmt (mk-AS`LetBeSTStmt (lhs, St, In, -)) ≜
.2       (if is-AS`TypeBind (lhs)
.3        then return [mk-INSTRTP`ERRINST (RTERR`TYPE-BIND-EVAL)] ;
.4        let St-instr = if St = nil
.5                       then [mk-INSTRTP`PUSH (mk-SEM`BOOL (true))]
.6                       else CEXPR`E2I (St),
.7            succ-instr = [mk-INSTRTP`PUSH (mk-SEM`BOOL (true))],
.8            fail-instr = [mk-INSTRTP`PUSH (mk-SEM`BOOL (false))],
.9            St-failed = CEXPR`ConcIfThenElse ([mk-INSTRTP`ISEMPTYSET (1, nil )],
.10                                            fail-instr ⌢ succ-instr,
.11                                            [mk-INSTRTP`SELBLKENV (1)]⌢fail-instr) in
.12       let prep-instr = CEXPR`E2I (lhs.Set)⌢[mk-INSTRTP`EMPTYBLKENV (READ_ONLY)],
.13           in-loop-instr = CEXPR`ConcIfThenElse (St-instr,
.14                                                 succ-instr ⌢ succ-instr,
.15                                                 St-failed),
```

```
      empty-instr = CEXPR‘ConcIfThenElse ([mk-INSTRTP‘ISEMPTYSET (1, SEM)],
                                          [mk-INSTRTP‘ERRINST (RTERR‘EMPTY-ENV-S)],
                                          [mk-INSTRTP‘SELELEM ()] ↷
                                          CPAT‘P2I (lhs.pat) ↷
                                          [mk-INSTRTP‘TRYMATCH ()]),
      postt-instr = [mk-INSTRTP‘POP (1)] ↷ S2I (In) ↷
                    [mk-INSTRTP‘POPBLKENV ()] in
  let out-loop-instr = CEXPR‘ConcIfThenElse (empty-instr,
                                             CEXPR‘CombRepeatUntil (in-loop-instr)↷

                                             [mk-INSTRTP‘REMSTACKELEM (2)],
                                             fail-instr) in
  return prep-instr ↷ CEXPR‘CombRepeatUntil (out-loop-instr) ↷ postt-instr );
```

1.3.4 Assign Statements

```
139.0 CompileAtomicAssignStmt : AS‘AtomicAssignStmt o→ STKM‘SubProgram
      CompileAtomicAssignStmt (mk-AS‘AtomicAssignStmt (assstmtl, -)) ≜
        (dcl sp : STKM‘SubProgram := [];
         for mk-AS‘AssignStmt (lhs, rhs, -) in assstmtl
         do sp := sp ↷ CEXPR‘E2I (rhs) ↷ CPAT‘SD2I (lhs);
         sp := sp ↷ [mk-INSTRTP‘ATOMIC (len assstmtl),
                mk-INSTRTP‘PUSH (mk-SEM‘CONT ())];
         return sp );
```

```
140.0 CompileAssignStmt : AS‘AssignStmt o→ STKM‘SubProgram
      CompileAssignStmt (mk-AS‘AssignStmt (lhs, rhs, -)) ≜
        let rhs-instr = CEXPR‘E2I (rhs),
            lhs-instr = CPAT‘SD2I (lhs),
            cont-check = CEXPR‘ConcIfThenElse
                           (
                            [mk-INSTRTP‘ISCONT ()],
                            [mk-INSTRTP‘ERRINST (RTERR‘OP-RETURNED-CONT)],
                            lhs-instr ↷ [mk-INSTRTP‘ASSIGNSD (),
                             mk-INSTRTP‘PUSH (mk-SEM‘CONT ())]) in
        return rhs-instr ↷ cont-check;
```

1.3.5 Loop Statements

```
141.0 CompileSeqForLoopStmt : AS‘SeqForLoopStmt o→ STKM‘SubProgram
      CompileSeqForLoopStmt (mk-AS‘SeqForLoopStmt (cv, dirn, fseq, body, -)) ≜
        let succ-instr = [mk-INSTRTP‘PUSH (mk-SEM‘BOOL (true))],
```

.3 *fail-instr* = [mk-*INSTRTP*‘*PUSH* (mk-*SEM*‘*BOOL* (false))],
.4 *prep-instr* = *CEXPR*‘*E2I* (*fseq*) $\curvearrowright$ [mk-*INSTRTP*‘*PUSH* (mk-*SEM*‘*CONT* ()),
.5 mk-*INSTRTP*‘*ISEMPTYSEQ* (2, SEM),
.6 mk-*INSTRTP*‘*UNOP* (NOT)],
.7 *sel-and-test* = [mk-*INSTRTP*‘*SELSEQELEM* (*dirn*)] $\curvearrowright$
.8 *CPAT*‘*PB2I* (*cv*),
.9 *bind-pat* =
.10 [mk-*INSTRTP*‘*TRYANYMATCH* ()] $\curvearrowright$
.11 *CEXPR*‘*ConcIfThenElse* ([], [], [mk-*INSTRTP*‘*ERRINST* (*RTERR*‘*EMPTY-ENV-S*)]) $\curvearrowright$

.12 *TIME*‘*MkLoopBind* (1),
.13 *remove-last-res* = [mk-*INSTRTP*‘*POP* (1)],
.14 *loop-body* =
.15 *remove-last-res* $\curvearrowright$ *sel-and-test* $\curvearrowright$ [mk-*INSTRTP*‘*EMPTYBLKENV* (READ_ONLY)] $\curvearrowright$

.16 *bind-pat* $\curvearrowright$
.17 *S2I* (*body*) $\curvearrowright$ [mk-*INSTRTP*‘*POPBLKENV* ()] $\curvearrowright$
.18 *CEXPR*‘*ConcIfThenElse* ([mk-*INSTRTP*‘*ISCONT* ()],
.19 [mk-*INSTRTP*‘*ISEMPTYSEQ* (2, SEM), mk-*INSTRTP*‘*UNOP* (NOT)],
.20 *fail-instr*) $\curvearrowright$
.21 *TIME*‘*MkCbr* (),
.22 *clean-up* = [mk-*INSTRTP*‘*REMSTACKELEM* (2)] in
.23 return *prep-instr* $\curvearrowright$ *CEXPR*‘*CombWhileLoop* (*loop-body*) $\curvearrowright$ *clean-up*;

142.0 *CompileSetForLoopStmt* : *AS*‘*SetForLoopStmt* $\xrightarrow{o}$ *STKM*‘*SubProgram*
.1 *CompileSetForLoopStmt* (mk-*AS*‘*SetForLoopStmt* (*cv*, *fset*, *body*, *ci*)) $\triangleq$
.2 let *succ-instr* = [mk-*INSTRTP*‘*PUSH* (mk-*SEM*‘*BOOL* (true))],
.3 *fail-instr* = [mk-*INSTRTP*‘*PUSH* (mk-*SEM*‘*BOOL* (false))],
.4 *bind-l* = [mk-*AS*‘*MultSetBind* ([*cv*], *fset*, *ci*)],
.5 *prep-instr* =
.6 *CPAT*‘*CompileMultBindL* (*bind-l*, DO_PARTITION) $\curvearrowright$ [mk-*INSTRTP*‘*PUSH* (mk-*SEM*‘*CONT* ()

.7 *TIME*‘*MkLoopBind* (1),
.8 *testCont* = *CEXPR*‘*ConcIfThenElse* ([mk-*INSTRTP*‘*ISCONT* ()], *fail-instr*, *succ-instr*),
.9 *loop-body* =
.10 [mk-*INSTRTP*‘*POP* (1), mk-*INSTRTP*‘*SELBLKENV* (1)] $\curvearrowright$ *S2I* (*body*) $\curvearrowright$
[mk-*INSTRTP*‘*POPBLKENV* ()] $\curvearrowright$ *testCont* $\curvearrowright$
.11 *TIME*‘*MkCbr* (),
.12 *loop* = *CEXPR*‘*ConcIfThenElse* ([mk-*INSTRTP*‘*ISEMPTYSET* (2, nil)], *succ-instr*, *loop-body*) in
.13 return *prep-instr* $\curvearrowright$ *CEXPR*‘*CombRepeatUntil* (*loop*) $\curvearrowright$ [mk-*INSTRTP*‘*REMSTACKELEM* (2)];

143.0 *CompileIndexForLoopStmt* : *AS*‘*IndexForLoopStmt* $\xrightarrow{o}$ *STKM*‘*SubProgram*
.1 *CompileIndexForLoopStmt* (mk-*AS*‘*IndexForLoopStmt* (*cv*, *lb-e*, *ub-e*, *by-e*, *body*, -)) $\triangleq$
.2 let *succ-instr* = [mk-*INSTRTP*‘*PUSH* (mk-*SEM*‘*BOOL* (true))],
.3 *fail-instr* = [mk-*INSTRTP*‘*PUSH* (mk-*SEM*‘*BOOL* (false))],

```
.4       step-e = if by-e = nil
.5                then [mk-INSTRTP‘PUSH (mk-SEM‘NUM (1))]
.6                else CEXPR‘E2I (by-e),
.7       args = step-e ⁀ CEXPR‘E2I (ub-e) ⁀ CEXPR‘E2I (lb-e) ⁀ [mk-INSTRTP‘VERIFYINDEXARGS ()],
.8       prep-instr =
.9            args ⁀ [mk-INSTRTP‘PUSH (mk-SEM‘CONT ()),
.10            mk-INSTRTP‘EMPTYBLKENV (READ_ONLY),
.11            mk-INSTRTP‘TESTCOUNTER ()] ⁀
.12           TIME‘MkLoopBind (1),
.13      bindArgs = [mk-INSTRTP‘COPYVAL (),
.14                  mk-INSTRTP‘PUSH (mk-STKM‘PatternName (cv, cv.cid)),
.15                  mk-INSTRTP‘TRYANYMATCH (),
.16                  mk-INSTRTP‘POP (1)],
.17      loop =
.18           [mk-INSTRTP‘POP (1)] ⁀ bindArgs ⁀ S2I (body) ⁀
.19           CEXPR‘ConcIfThenElse ([mk-INSTRTP‘ISCONT ()],
.20                                 [mk-INSTRTP‘INCRCOUNTER (), mk-INSTRTP‘TESTCOUNTER ()],
.21                                 fail-instr) ⁀
.22           TIME‘MkCbr (),
.23      cleanUp = [mk-INSTRTP‘REMSTACKELEM (2), mk-INSTRTP‘REMSTACKELEM (2),
.24                 mk-INSTRTP‘REMSTACKELEM (2), mk-INSTRTP‘POPBLKENV ()] in
.25   return prep-instr ⁀ CEXPR‘CombWhileLoop (loop) ⁀ cleanUp;
```

```
144.0 CompileWhileLoopStmt : AS‘WhileLoopStmt →(o) STKM‘SubProgram
.1    CompileWhileLoopStmt (mk-AS‘WhileLoopStmt (test, body, -)) △
.2      let succ-instr = [mk-INSTRTP‘PUSH (mk-SEM‘BOOL (true))],
.3          fail-instr = [mk-INSTRTP‘PUSH (mk-SEM‘BOOL (false))],
.4          testI =
.5              CEXPR‘E2I (test) ⁀ [mk-INSTRTP‘COPYVAL (),
.6              mk-INSTRTP‘ISCHECK (mk-AS‘BasicType (BOOLEAN, CI‘NilContextId))] ⁀

.7              CEXPR‘ConcIfThenElse ([], [], [mk-INSTRTP‘ERRINST (RTERR‘LOOP-EXPR-NOT-AN-EXPR
.8          bodyI = S2I (body),
.9          prep-instr = [mk-INSTRTP‘PUSH (mk-SEM‘CONT ())] ⁀ testI,
.10         loop =
.11             [mk-INSTRTP‘POP (1)] ⁀ bodyI ⁀
.12             CEXPR‘ConcIfThenElse ([mk-INSTRTP‘ISCONT ()],
.13                                   testI, fail-instr) ⁀
.14             TIME‘MkCbr () in
.15     return prep-instr ⁀ CEXPR‘CombWhileLoop (loop);
```

1.3.6 Call Statement

The use of a state designator is NOT taken into account and we believe that it will not be taken into account at all. Thus, since the code generator does not support it and no of the examples in the examples repository uses it we recommend to get rid of it entirely in VDM-SL.

```
CompileCallStmt : AS`CallStmt ⁰→ STKM`SubProgram
CompileCallStmt (mk-AS`CallStmt (obj, oprt, args, -)) △
  (dcl sp : STKM`SubProgram := [];
   if obj ≠ nil
   then sp := sp ⁀ CEXPR`E2I (obj);
   sp := sp ⁀ [mk-INSTRTP`EMPTYLIST ()];
   for arg in args
   do sp := sp ⁀ CEXPR`E2I (arg) ⁀ [mk-INSTRTP`APPENDESTCK ()];
   return sp⁀[mk-INSTRTP`CALLGUARD (obj ≠ nil , oprt)]⁀CMPL`SetContext (oprt.cid, false)⁀
[mk-INSTRTP`PPCALL ()] );
```

1.3.7 Return Statement

```
CompileReturnStmt : AS`ReturnStmt ⁰→ STKM`SubProgram
CompileReturnStmt (mk-AS`ReturnStmt (expr, -)) △
  if expr = nil
  then return [mk-INSTRTP`PUSH (mk-SEM`RETURN ())]
  else return CEXPR`E2I (expr) ;
```

1.3.8 If Statement

```
CompileIfStmt : AS`IfStmt ⁰→ STKM`SubProgram
CompileIfStmt (mk-AS`IfStmt (cond-e, cons-s, elif-l, altn-s, -)) △
  let cond-l =
          CEXPR`E2I (cond-e) ⁀ [mk-INSTRTP`COPYVAL (),
           mk-INSTRTP`ISCHECK (mk-AS`BasicType (BOOLEAN, CI`NilContextId))]⁀

          CEXPR`ConcIfThenElse ([], [], [mk-INSTRTP`ERRINST (RTERR`TEST-EXPR-NOT-AN-EXPR
      cons-l =
          S2I (cons-s) ⁀
          TIME`MkCbr () in
  if len elif-l = 0
  then return CEXPR`ConcIfThenElse (cond-l, cons-l,
                                    if altn-s ≠ nil
                                    then S2I (altn-s) ⁀
                                         TIME`MkBr ()
                                    else [mk-INSTRTP`PUSH (mk-SEM`CONT ())])
  else let mk-AS`ElseifStmt (elif-cond, elif-s, elif-cid) = hd elif-l,
```

```
        altn-l =
              S2I (mk-AS`IfStmt (elif-cond, elif-s, tl elif-l,
                                         altn-s, elif-cid)) ⁀
              TIME`MkCbr () in
    return CEXPR`ConcIfThenElse (cond-l, cons-l, altn-l) ;
```

1.3.9 Cases Statement

```
CompileCasesStmt : AS`CasesStmt →(o) STKM`SubProgram
CompileCasesStmt (mk-AS`CasesStmt (sel, altns, others-s, -)) ≜
  (dcl sp : STKM`SubProgram := if others-s = nil
                then [mk-INSTRTP`PUSH (mk-SEM`CONT ())]
                else S2I (others-s) ⁀
                     TIME`MkBr ();
   for mk-AS`CasesStmtAltn (match-lp, body-s, -) in altns
   do let body-instr =
              S2I (body-s) ⁀
              TIME`MkCbr () in
      for pat in match-lp
      do let test-instr =
                 [mk-INSTRTP`COPYVAL ()] ⁀
                 CPAT`P2I (pat) ⁀
                 [mk-INSTRTP`TRYANYMATCH ()] ⁀
                 TIME`MkMatchPattern () in
         (sp := sp ⁀ TIME`MkBr ();
          sp := CEXPR`ConcIfThenElse (test-instr, body-instr, sp));
   return CEXPR`E2I (sel) ⁀
          [mk-INSTRTP`EMPTYBLKENV (READ_ONLY)] ⁀
          sp ⁀ [mk-INSTRTP`POPBLKENV ()] ⁀
          [mk-INSTRTP`REMSTACKELEM (2)] );
```

1.3.10 Error Statement

```
CompileErrorStmt : AS`ErrorStmt →(o) STKM`SubProgram
CompileErrorStmt (-) ≜
  return [mk-INSTRTP`ERRINST (RTERR`ERROR-STATEMENT)];
```

1.3.11 Exception Handling Statements

1.3.12 Exit Statement

150.0 $CompileExitStmt : AS`ExitStmt \xrightarrow{o} STKM`SubProgram$

.1 $CompileExitStmt\,(\mathsf{mk}\text{-}AS`ExitStmt\,(expr, \text{-})) \triangleq$

.2 $\quad \mathsf{if}\ expr = \mathsf{nil}$

.3 $\quad \mathsf{then\ return}\ [\mathsf{mk}\text{-}INSTRTP`MKEXIT\,(\mathsf{true}), \mathsf{mk}\text{-}INSTRTP`EXITVAL\,()]$

.4 $\quad \mathsf{else\ return}\ CEXPR`E2I\,(expr) \curvearrowright [\mathsf{mk}\text{-}INSTRTP`MKEXIT\,(\mathsf{false}), \mathsf{mk}\text{-}INSTRTP`EXITVAL\,()]$

;

151.0 $CompileAlwaysStmt : AS`AlwaysStmt \xrightarrow{o} STKM`SubProgram$

.1 $CompileAlwaysStmt\,(\mathsf{mk}\text{-}AS`AlwaysStmt\,(Post, body, \text{-})) \triangleq$

.2 $\quad \mathsf{let}\ hand\text{-}no = GetNewTrapNo\,(),$

.3 $\qquad b\text{-}instr = S2I\,(body),$

.4 $\qquad p\text{-}instr = S2I\,(Post)\ \mathsf{in}$

.5 $\quad \mathsf{return}\ [\mathsf{mk}\text{-}INSTRTP`PUSHTH\,(hand\text{-}no)] \curvearrowright b\text{-}instr \curvearrowright$

.6 $\qquad [\mathsf{mk}\text{-}INSTRTP`HANDID\,(hand\text{-}no)] \curvearrowright$

.7 $\qquad [\mathsf{mk}\text{-}INSTRTP`POPTH\,()] \curvearrowright$

.8 $\qquad p\text{-}instr \curvearrowright$

.9 $\qquad [\mathsf{mk}\text{-}INSTRTP`POP\,(1)] \curvearrowright$

.10 $\qquad CEXPR`ConcIfThenElse\,([\mathsf{mk}\text{-}INSTRTP`ISEXIT\,()],$

.11 $\qquad\qquad [\mathsf{mk}\text{-}INSTRTP`EXITVAL\,()], []);$

152.0 $CompileTrapStmt : AS`TrapStmt \xrightarrow{o} STKM`SubProgram$

.1 $CompileTrapStmt\,(\mathsf{mk}\text{-}AS`TrapStmt\,(pb, Post, body, \text{-})) \triangleq$

.2 $\quad \mathsf{let}\ hand\text{-}no = GetNewTrapNo\,(),$

.3 $\qquad b\text{-}instr = S2I\,(body),$

.4 $\qquad p\text{-}instr = S2I\,(Post),$

.5 $\qquad pat\text{-}instr = CPAT`PB2I\,(pb)\ \mathsf{in}$

.6 $\quad \mathsf{let}\ han\text{-}instr =$

.7 $\qquad [\mathsf{mk}\text{-}INSTRTP`EMPTYBLKENV\,(\mathrm{READ\_ONLY}), \mathsf{mk}\text{-}INSTRTP`REMEXITVAL\,()] \curvearrowright$

.8 $\qquad pat\text{-}instr \curvearrowright$

.9 $\qquad [\mathsf{mk}\text{-}INSTRTP`TRYANYMATCH\,()] \curvearrowright$

.10 $\qquad CEXPR`ConcIfThenElse$

.11 $\qquad\quad ($

.12 $\qquad\quad [],$

.13 $\qquad\quad [\mathsf{mk}\text{-}INSTRTP`POP\,(1)] \curvearrowright$

.14 $\qquad\quad p\text{-}instr \curvearrowright$

.15 $\qquad\quad [\mathsf{mk}\text{-}INSTRTP`POPBLKENV\,()],$

.16 $\qquad\quad [\mathsf{mk}\text{-}INSTRTP`EXITVAL\,()]),$

.17 $\qquad use\text{-}handler =$

.18 $\qquad [\mathsf{mk}\text{-}INSTRTP`HANDID\,(hand\text{-}no),$

.19 $\qquad \mathsf{mk}\text{-}INSTRTP`POPTH\,()] \curvearrowright$

.20 $\qquad CEXPR`ConcIfThenElse\,([\mathsf{mk}\text{-}INSTRTP`ISNEEXIT\,()], han\text{-}instr,$

.21 $\qquad\qquad [\mathsf{mk}\text{-}INSTRTP`EXITVAL\,()])\ \mathsf{in}$

.22 $\quad \mathsf{return}\ [\mathsf{mk}\text{-}INSTRTP`PUSHTH\,(hand\text{-}no)] \curvearrowright b\text{-}instr \curvearrowright$

.23 $\qquad [\mathsf{mk}\text{-}INSTRTP`POPTH\,(), \mathsf{mk}\text{-}INSTRTP`BR\,(\mathsf{len}\ use\text{-}handler)] \curvearrowright$

.24 $\qquad use\text{-}handler;$

```
153.0  CompileRecTrapStmt : AS‘RecTrapStmt →(o) STKM‘SubProgram
   .1  CompileRecTrapStmt (mk-AS‘RecTrapStmt (traps, body, -)) ≜
   .2    (dcl sp : STKM‘SubProgram := [mk-INSTRTP‘POPTH (), mk-INSTRTP‘EXITVAL ()];
   .3     let hand-no = GetNewTrapNo (),
   .4         b-instr = S2I (body)⁀[mk-INSTRTP‘EMPTYBLKENV (READ_ONLY)] in
   .5     (for mk-AS‘Trap (pb, Post, -) in traps
   .6      do let p-instr = S2I (Post) ⁀ [mk-INSTRTP‘POPBLKENV ()],
   .7             pat-instr = CPAT‘PB2I (pb) in
   .8         let test-instr =
   .9                 [mk-INSTRTP‘REMEXITVAL (),
  .10                  mk-INSTRTP‘REMSTACKELEM (2)] ⁀
  .11                 pat-instr ⁀
  .12                 [mk-INSTRTP‘EMPTYBLKENV (READ_ONLY),
  .13                  mk-INSTRTP‘TRYANYMATCH ()] in
  .14         sp := CEXPR‘ConcIfThenElse (test-instr, p-instr, sp);
  .15      let loop-instr = CEXPR‘ConcIfThenElse ([mk-INSTRTP‘ISNEEXIT ()], sp,
  .16                                            [mk-INSTRTP‘POPTH (), mk-INSTRTP‘EXITVAL ()]),
  .17          hand-body = [mk-INSTRTP‘HANDID (hand-no)] ⁀ loop-instr in
  .18      return [mk-INSTRTP‘PUSHTH (hand-no)] ⁀ b-instr ⁀
  .19             [mk-INSTRTP‘BR (len hand-body)] ⁀
  .20             hand-body ⁀ [mk-INSTRTP‘POPTH ()] ));
```

1.3.13 Start and Start List Statements

```
154.0  CompileStartStmt : AS‘StartStmt →(o) STKM‘SubProgram
   .1  CompileStartStmt (mk-AS‘StartStmt (expr, -)) ≜
   .2    return CEXPR‘E2I (expr)⁀[mk-INSTRTP‘STARTLIST (false), mk-INSTRTP‘PUSH (mk-SEM‘CONT ())]
```

```
155.0  CompileStartListStmt : AS‘StartListStmt →(o) STKM‘SubProgram
   .1  CompileStartListStmt (mk-AS‘StartListStmt (expr, -)) ≜
   .2    return CEXPR‘E2I (expr) ⁀
   .3           TIME‘MkRuntimeStartList () ⁀
   .4           [mk-INSTRTP‘STARTLIST (true), mk-INSTRTP‘PUSH (mk-SEM‘CONT ())];
```

```
156.0  CompileDurationStmt : AS‘DurationStmt →(o) STKM‘SubProgram
   .1  CompileDurationStmt (mk-AS‘DurationStmt (num, stmt, -)) ≜
   .2    (dcl sp : STKM‘SubProgram := [];
   .3     sp := sp ⁀ [mk-INSTRTP‘PUSHDURATION ()];
   .4     sp := sp ⁀ S2I (stmt);
   .5     sp := sp ⁀ [mk-INSTRTP‘POPDURATION (num.val)];
   .6     return sp );
```

1.3.14 Block Statement

```
157.0 CompileBlockStmt : AS`BlockStmt -o-> STKM`SubProgram
   .1 CompileBlockStmt (mk-AS`BlockStmt (dcl-l, stmt-l, -)) ≜
   .2   (dcl sp : STKM`SubProgram := S2I (stmt-l (len stmt-l));
   .3    for stmt in stmt-l (1, ..., len stmt-l − 1)
   .4    do sp := CEXPR`ConcIfThenElse (S2I (stmt) ⌢ [mk-INSTRTP`ISCONT ()],
   .5                                   [mk-INSTRTP`POP (1)] ⌢ sp,
   .6                                   []);
   .7    for mk-AS`AssignDef (id, tp, Iinit, -) in dcl-l
   .8    do let t =
   .9              tp in
  .10       sp := (if Iinit = nil
  .11              then [mk-INSTRTP`PUSH (mk-SEM`UNDEF ())]
  .12              else CEXPR`E2I (Iinit) ⌢
  .13                   [mk-INSTRTP`DTC (t)]) ⌢
  .14             [mk-INSTRTP`APPENDBLKENV (id, t)] ⌢
  .15             sp;
  .16    sp := [mk-INSTRTP`EMPTYBLKENV (READ_WRITE)]⌢sp⌢[mk-INSTRTP`POPBLKENV ()];
  .17    return sp );
```

1.3.15 Nondeterministic Statement

The approach chosen to compile nondeterministic statements is that a sequence with all instruction code for all element statements is produced. This sequence is used to create a list of all the relative start positions of the different element statements. At the end of each of the element statements there is a jump back to a special non-determistic instruction. This non-deterministic instruction assumes that the evaluation stack first contains the value of the executed element statement and then a sequence of the element statement instruction code. In case the ramdom order is set by the user the sequence is permuted with respect to the given seed. Thus, the permutation is done before the special non-determistic instruction is reached.

```
158.0 CompileNonDetStmt : AS`NonDetStmt -o-> STKM`SubProgram
   .1 CompileNonDetStmt (mk-AS`NonDetStmt (stmts, -)) ≜
   .2   (dcl sp-l : STKM`SubProgram* := [S2I (stmts (i)) | i ∈ inds stmts];
   .3    let reljumps = [0] ⌢ RelJumpLengths (0, sp-l) in
   .4    (sp-l := [sp-l (i) ⌢ [mk-INSTRTP`BR (− reljumps (i + 1) − 2)] |
   .5              i ∈ inds sp-l];
   .6     return [mk-INSTRTP`PUSH (reljumps (1, ..., len reljumps − 1))] ⌢
   .7            [mk-INSTRTP`RANDOM ()] ⌢
   .8            [mk-INSTRTP`PUSH (mk-SEM`CONT ())] ⌢
   .9            [mk-INSTRTP`NONDETSTMT ()] ⌢
  .10            [mk-INSTRTP`BR (reljumps (len reljumps))] ⌢
  .11            conc sp-l ))
```

functions

159.0 $RelJumpLengths : \mathbb{N} \times STKM`SubProgram^* \to \mathbb{N}^*$

.1 $RelJumpLengths\,(length, sp\text{-}l) \triangleq$

.2 if $sp\text{-}l = []$

.3 then $[]$

.4 else let $newlength = 1 + length +$ len hd $sp\text{-}l$ in

.5 $[newlength] \curvearrowright RelJumpLengths\,(newlength,$ tl $sp\text{-}l)$

end $CSTMT$

Test Suite : rtinfo.ast
Module : CSTMT

Name	**#Calls**	**Coverage**
CSTMT‘S2I	**undefined**	**undefined**
CSTMT‘GetNewTrapNo	**undefined**	**undefined**
CSTMT‘CompileIfStmt	**undefined**	**undefined**
CSTMT‘CompileDefStmt	**undefined**	**undefined**
CSTMT‘CompileLetStmt	**undefined**	**undefined**
CSTMT‘RelJumpLengths	**undefined**	**undefined**
CSTMT‘CompileCallStmt	**undefined**	**undefined**
CSTMT‘CompileExitStmt	**undefined**	**undefined**
CSTMT‘CompileTrapStmt	**undefined**	**undefined**
CSTMT‘CompileBlockStmt	**undefined**	**undefined**
CSTMT‘CompileCasesStmt	**undefined**	**undefined**
CSTMT‘CompileErrorStmt	**undefined**	**undefined**
CSTMT‘CompileStartStmt	**undefined**	**undefined**
CSTMT‘CompileAlwaysStmt	**undefined**	**undefined**
CSTMT‘CompileAssignStmt	**undefined**	**undefined**
CSTMT‘CompileNonDetStmt	**undefined**	**undefined**
CSTMT‘CompileReturnStmt	**undefined**	**undefined**
CSTMT‘CompileLetBeSTStmt	**undefined**	**undefined**
CSTMT‘CompileRecTrapStmt	**undefined**	**undefined**
CSTMT‘CompileDurationStmt	**undefined**	**undefined**
CSTMT‘CompileStartListStmt	**undefined**	**undefined**
CSTMT‘CompileWhileLoopStmt	**undefined**	**undefined**
CSTMT‘CompileSeqForLoopStmt	**undefined**	**undefined**
CSTMT‘CompileSetForLoopStmt	**undefined**	**undefined**
CSTMT‘CompileAtomicAssignStmt	**undefined**	**undefined**
CSTMT‘CompileIndexForLoopStmt	**undefined**	**undefined**
Total Coverage		**0%**

1.4 Compilation of Patterns

This module specifies how ASTs representing patterns are translated into the stack machine instructions defined in module *STKM*.

module *CPAT*

imports

160.0 from *AS* all ,

161.0 from *CI* all ,

162.0 from *IO* all ,

163.0 from *PAT* all ,

164.0 from *REP* all ,

165.0 from *SEM* all ,

166.0 from *FREE* all ,

167.0 from *STKM* all ,

168.0 from *CEXPR* all ,

169.0 from *RTERR* all ,

170.0 from *GLOBAL* all ,

171.0 from *SCHDTP* all ,

172.0 from *INSTRTP* all

exports all

definitions

operations

173.0 $CompileMultBindL : AS`BindList \times PAT`PARTITION \xrightarrow{o} STKM`SubProgram$

.1 $CompileMultBindL\,(bind\text{-}l, part) \triangleq$

.2 (dcl $sp : STKM`SubProgram := []$,

.3 $length : \mathbb{N} := 0$;

.4 if $\exists\, bind \in$ elems $bind\text{-}l \cdot$ is-$AS`MultTypeBind\,(bind)$

.5 then return [mk-$INSTRTP`ERRINST\,(RTERR`TYPE\text{-}BIND\text{-}EVAL)$] ;

.6 for mk-$AS`MultSetBind\,(pat\text{-}l, set\text{-}e, \text{-})$ in $bind\text{-}l$

.7 do $(length := length + 1 +$ len $pat\text{-}l$;

.8 $sp := sp \curvearrowright CEXPR`E2I\,(set\text{-}e)$;

.9 for $pat\text{-}p$ in $pat\text{-}l$

.10 do $sp := sp \curvearrowright P2I\,(pat\text{-}p))$;

.11 return $sp \curvearrowright$ [mk-$INSTRTP`MULTBINDL\,(length, part)$]);

174.0 $PB2I : AS`PatternBind \xrightarrow{o} STKM`SubProgram$
.1 $PB2I\,(pb) \triangleq$
.2 return if is-$AS`TypeBind\,(pb)$
.3 then [mk-$INSTRTP`DTC\,(pb.tp)$] $\curvearrowright$ $P2I\,(pb.pat)$
.4 elseif is-$AS`SetBind\,(pb)$
.5 then $CEXPR`E2I\,(pb.Set)$ $\curvearrowright$ [mk-$INSTRTP`DTCSET\,()$] $\curvearrowright$
.6 $P2I\,(pb.pat)$
.7 else $P2I\,(pb)$;

175.0 $P2I : AS`Pattern \xrightarrow{o} STKM`SubProgram$
.1 $P2I\,(pat) \triangleq$
.2 return if $AnyMatchVals\,(pat)$
.3 then $PStack2I\,(pat)$
.4 else $PDirect2I\,(pat)$

Note that the match values are translated directly to instruction code which produces a value without having any *MatchVal* tag with an additional context id around it. This could naturally easily be changed here.
functions

176.0 $PStack2I : AS`Pattern \to STKM`SubProgram$
.1 $PStack2I\,(pat) \triangleq$
.2 cases true :
.3 (is-$AS`PatternName\,(pat)$) $\to$
.4 [mk-$INSTRTP`PUSH$ (mk-$STKM`PatternName\,(pat.nm, pat.cid)$)],
.5 (is-$AS`MatchVal\,(pat)$) $\to$ $CEXPR`E2I\,(pat.val)$ $\curvearrowright$
.6 [mk-$INSTRTP`MATCHVAL\,()$],
.7 (is-$AS`SetEnumPattern\,(pat)$),
.8 (is-$AS`SeqEnumPattern\,(pat)$) $\to$
.9 $Enum2I\,(pat)$,
.10 (is-$AS`SetUnionPattern\,(pat)$) $\to$
.11 $PStack2I\,(pat.lp) \curvearrowright PStack2I\,(pat.rp) \curvearrowright$ [mk-$INSTRTP`SETUNION\,(pat.cid)$],
.12 (is-$AS`SeqConcPattern\,(pat)$) $\to$
.13 $PStack2I\,(pat.lp) \curvearrowright PStack2I\,(pat.rp) \curvearrowright$ [mk-$INSTRTP`SEQCONC\,(pat.cid)$],
.14 (is-$AS`TuplePattern\,(pat)$) $\to$
.15 $Tuple2I\,(pat)$,
.16 (is-$AS`RecordPattern\,(pat)$) $\to$
.17 $Record2I\,(pat)$
.18 end;

177.0 $PDirect2I : AS`Pattern \to STKM`SubProgram$

.1 $PDirect2I\,(pat) \triangleq$

.2 if is-$AS`MatchVal\,(pat)$

.3 then $CEXPR`E2I\,(pat.val) \curvearrowright [$mk-$INSTRTP`MATCHVAL\,()]$

.4 else $[$mk-$INSTRTP`PUSH\,(P2P\,(pat))]$;

178.0 $PL2PL : AS`Pattern^* \to STKM`Pattern^*$

.1 $PL2PL\,(pl) \triangleq$

.2 $[P2P\,(pl\,(i)) \mid i \in$ inds $pl]$;

```
179.0 P2P : AS`Pattern → STKM`Pattern
P2P (pat) △
  cases true :
    (is-AS`PatternName (pat)) →
        mk-STKM`PatternName (pat.nm, pat.cid),
    (is-AS`MatchVal (pat)) →
        if IsLiteral (pat.val)
        then let v = cases pat.val :
                        mk-AS`BoolLit (b,-) → mk-SEM`BOOL (b),
                        mk-AS`NilLit (-) → mk-SEM`NIL (),
                        mk-AS`RealLit (r,-) → mk-SEM`NUM (r),
                        mk-AS`CharLit (c,-) → mk-SEM`CHAR (c),
                        mk-AS`TextLit (str,-) →
                            mk-SEM`SEQ ([mk-SEM`CHAR (str (i)) |
                                              i ∈ inds str]),
                        mk-AS`QuoteLit (quo,-) →
                            mk-SEM`QUOTE ([quo (i) | i ∈ inds quo])
                      end in
              mk-STKM`MatchVal (v)
        else mk-STKM`PatternName (nil , pat.cid),
    (is-AS`SetEnumPattern (pat)) →
        let els = [P2P (pat.Elems (i)) | i ∈ inds pat.Elems] in
        mk-STKM`SetEnumPattern (els, pat.cid),
    (is-AS`SetUnionPattern (pat)) →
        let lp = P2P (pat.lp),
            rp = P2P (pat.rp) in
        mk-STKM`SetUnionPattern (lp, rp, pat.cid),
    (is-AS`SeqEnumPattern (pat)) →
        let els = [P2P (pat.els (i)) | i ∈ inds pat.els] in
        mk-STKM`SeqEnumPattern (els, pat.cid),
    (is-AS`SeqConcPattern (pat)) →
        let lp = P2P (pat.lp),
            rp = P2P (pat.rp) in
        mk-STKM`SeqConcPattern (lp, rp, pat.cid),
    (is-AS`TuplePattern (pat)) →
        let els = [P2P (pat.fields (i)) | i ∈ inds pat.fields] in
        mk-STKM`TuplePattern (els, pat.cid),
    (is-AS`RecordPattern (pat)) →
        let els = [P2P (pat.fields (i)) | i ∈ inds pat.fields] in
        let pnm =
                pat.nm in
        mk-STKM`RecordPattern (pnm, els, pat.cid)
  end
```

operations

180.0 $Enum2I : AS`SetEnumPattern \mid AS`SeqEnumPattern \xrightarrow{o} STKM`SubProgram$
.1 $Enum2I\,(pat) \triangleq$
.2 (dcl $sp : STKM`SubProgram :=$ if is-$AS`SetEnumPattern\,(pat)$
.3 then [mk-$INSTRTP`PUSH$ (mk-$STKM`SetEnumPattern\,([], pat.cid)$)]
.4 else [mk-$INSTRTP`PUSH$ (mk-$STKM`SeqEnumPattern\,([], pat.cid)$)];
.5 let $els =$ if is-$AS`SetEnumPattern\,(pat)$
.6 then $pat.Elems$
.7 else $pat.els$ in
.8 for pat in els
.9 do ($sp := sp \curvearrowright PStack2I\,(pat)$;
.10 $sp := sp \curvearrowright$ [mk-$INSTRTP`ENUMAPPEND\,()$]);
.11 return sp);

181.0 $Tuple2I : AS`TuplePattern \xrightarrow{o} STKM`SubProgram$
.1 $Tuple2I$ (mk-$AS`TuplePattern\,(fields, cid)$) $\triangleq$
.2 (dcl $sp : STKM`SubProgram :=$ [mk-$INSTRTP`PUSH$ (mk-$STKM`TuplePattern\,([], cid)$)];
.3 for pat in $fields$
.4 do ($sp := sp \curvearrowright PStack2I\,(pat)$;
.5 $sp := sp \curvearrowright$ [mk-$INSTRTP`FIELDSAPPEND\,()$]);
.6 return sp);

182.0 $Record2I : AS`RecordPattern \xrightarrow{o} STKM`SubProgram$
.1 $Record2I$ (mk-$AS`RecordPattern\,(nm, fields, cid)$) $\triangleq$
.2 (dcl $sp : STKM`SubProgram :=$ [mk-$INSTRTP`PUSH$ (mk-$STKM`RecordPattern\,(nm, [], cid)$)];
.3 for pat in $fields$
.4 do ($sp := sp \curvearrowright PStack2I\,(pat)$;
.5 $sp := sp \curvearrowright$ [mk-$INSTRTP`FIELDSAPPEND\,()$]);
.6 return sp)

functions

183.0 $AnyMatchVals : AS`Pattern \to \mathbb{B}$

.1 $AnyMatchVals\,(pat) \;\triangleq$

.2 $\quad$ cases true :

.3 $\qquad$ $(\text{is-}AS`PatternName\,(pat)) \to$ false,

.4 $\qquad$ $(\text{is-}AS`MatchVal\,(pat)) \to$ true,

.5 $\qquad$ $(\text{is-}AS`SetEnumPattern\,(pat)) \to$

.6 $\qquad\quad$ $\exists\, p \in \text{elems}\; pat.Elems \cdot AnyMatchVals\,(p),$

.7 $\qquad$ $(\text{is-}AS`SetUnionPattern\,(pat)) \to$

.8 $\qquad\quad$ $AnyMatchVals\,(pat.lp) \vee AnyMatchVals\,(pat.rp),$

.9 $\qquad$ $(\text{is-}AS`SeqEnumPattern\,(pat)) \to$

.10 $\qquad\quad$ $\exists\, p \in \text{elems}\; pat.els \cdot AnyMatchVals\,(p),$

.11 $\qquad$ $(\text{is-}AS`SeqConcPattern\,(pat)) \to$

.12 $\qquad\quad$ $AnyMatchVals\,(pat.lp) \vee AnyMatchVals\,(pat.rp),$

.13 $\qquad$ $(\text{is-}AS`TuplePattern\,(pat)),$

.14 $\qquad$ $(\text{is-}AS`RecordPattern\,(pat)) \to$

.15 $\qquad\quad$ $\exists\, p \in \text{elems}\; pat.fields \cdot AnyMatchVals\,(p)$

.16 $\quad$ end;

184.0 $IsLiteral : AS`Expr \to \mathbb{B}$

.1 $IsLiteral\,(expr) \;\triangleq$

.2 $\quad$ $\text{is-}AS`BoolLit\,(expr) \vee$

.3 $\quad$ $\text{is-}AS`NilLit\,(expr) \vee$

.4 $\quad$ $\text{is-}AS`RealLit\,(expr) \vee$

.5 $\quad$ $\text{is-}AS`CharLit\,(expr) \vee$

.6 $\quad$ $\text{is-}AS`TextLit\,(expr) \vee$

.7 $\quad$ $\text{is-}AS`QuoteLit\,(expr)$

1.4.1 State Designators

operations

185.0 $SD2I : AS`StateDesignator \xrightarrow{o} STKM`SubProgram$

.1 $SD2I\,(sd) \;\triangleq$

.2 $\quad$ return if $AnyExprs\,(sd)$

.3 $\qquad$ then $SDStack2I\,(sd)$

.4 $\qquad$ else $SDDirect2I\,(sd)$

functions

```
186.0 SDStack2I : AS‘StateDesignator → STKM‘SubProgram
   .1 SDStack2I (sd) ≜
   .2   cases sd :
   .3     mk-AS‘Name (-, -) → [mk-INSTRTP‘PUSH (sd)],
   .4     mk-AS‘FieldRef (var, sel, ci) →
   .5         SDStack2I (var) ⁀
   .6         [mk-INSTRTP‘PUSH (sel)] ⁀
   .7         [mk-INSTRTP‘FREF (ci)],
   .8     mk-AS‘MapOrSeqRef (var, arg, ci) →
   .9         SDStack2I (var) ⁀
   .10        CEXPR‘E2I (arg) ⁀
   .11        [mk-INSTRTP‘MOSREF (ci)]
   .12  end;

187.0 SDDirect2I : AS‘StateDesignator → STKM‘SubProgram
   .1 SDDirect2I (sd) ≜
   .2   [mk-INSTRTP‘PUSH (SD2SD (sd))]
   .3 pre ¬ AnyExprs (sd) ;

188.0 SD2SD : AS‘StateDesignator → STKM‘StateDesignator
   .1 SD2SD (sd) ≜
   .2   cases sd :
   .3     mk-AS‘Name (-, -) → sd,
   .4     mk-AS‘FieldRef (var, sel, ci) → mk-STKM‘FieldRef (SD2SD (var), sel, ci)
   .5   end
   .6 pre ¬ AnyExprs (sd) ;

189.0 AnyExprs : AS‘StateDesignator → 𝔹
   .1 AnyExprs (sd) ≜
   .2   cases sd :
   .3     mk-AS‘Name (-, -) → false,
   .4     mk-AS‘FieldRef (var, -, -) → AnyExprs (var),
   .5     mk-AS‘MapOrSeqRef (-, -, -) → true
   .6   end

end CPAT
```

Test Suite : rtinfo.ast
Module : CPAT

Name	#Calls	Coverage
CPAT‘P2I	**undefined**	**undefined**
CPAT‘P2P	**undefined**	**undefined**

Name	#Calls	Coverage
CPAT‘PB2I	**undefined**	**undefined**
CPAT‘SD2I	**undefined**	**undefined**
CPAT‘PL2PL	**undefined**	**undefined**
CPAT‘SD2SD	**undefined**	**undefined**
CPAT‘Enum2I	**undefined**	**undefined**
CPAT‘Tuple2I	**undefined**	**undefined**
CPAT‘AnyExprs	**undefined**	**undefined**
CPAT‘PStack2I	**undefined**	**undefined**
CPAT‘Record2I	**undefined**	**undefined**
CPAT‘IsLiteral	**undefined**	**undefined**
CPAT‘PDirect2I	**undefined**	**undefined**
CPAT‘SDStack2I	**undefined**	**undefined**
CPAT‘SDDirect2I	**undefined**	**undefined**
CPAT‘AnyMatchVals	**undefined**	**undefined**
CPAT‘CompileMultBindL	**undefined**	**undefined**
Total Coverage		**0%**

1.5 Scheduling VDM++ Threads

The *SCHD* module is used for the top level scheduling of threads for the concurrent execution of VDM++ specifications.

```
module SCHD

  imports
    from AS all ,
    from CI all ,
    from PAT all ,
    from REP all ,
    from SEM all ,
    from STKM all ,
    from DEBUG all ,
    from RTERR all ,
    from STATE all ,
    from GLOBAL all ,
    from SCHDTP all ,
    from DEBUGTP
      types BreakInfo ,
```

203.0 from $INSTRTP$ all ,

204.0 from $TIMEMAP$ all ,

205.0 from $SETTINGS$ all ,

206.0 from $TIMETRACE$

207.0 operations $LogThreadSwapIn : SCHDTP`ThreadId \times [SEM`OBJ\text{-}Ref] \times [AS`Name] \xrightarrow{o} ()$;

.1 $LogThreadSwapOut : SCHDTP`ThreadId \times [SEM`OBJ\text{-}Ref] \times [AS`Name] \xrightarrow{o} ()$;

.2 $LogDelayedThreadSwapIn : SCHDTP`ThreadId \times [SEM`OBJ\text{-}Ref] \times [AS`Name] \times \mathbb{R} \xrightarrow{o} ()$,

208.0 from $TIMEPARSER$ all

exports all

definitions

The state of the scheduler has a number of components:

threads: contains all the active threads.

curthread: contains the thread id that is running on the stack machine.

perthreads: a sequence of thread ids representing the periodic threads currently executing in the interpreter. The sequence is in order of the next time which each thread should be run i.e. if thread 1 should next run at absolute time 200 and thread 2 should next run at absolute time 300, thread 1 appears before thread 2 in this sequence.

secondaryAlgorithm: indicates which scheduling algorithm is currently being used. It is not envisaged that this can be changed *during* execution.

classPriorityMap: a mapping from class names to numeric priorities.

maxPriority: the maximum priority, which is always greater than the maximum priority listed in the user-supplied priority file. The maximum priority will be assigned to the debug thread, to ensure that the debug thread terminates, and that it does so as soon as possible.

next_thread_id: This is a counter that contains the next available thread identification.

209.0 state $Schedule$ of

.1 $threads : SCHDTP`Threads$

.2 $curthread : SCHDTP`ThreadId$

.3 $perthreads : SCHDTP`ThreadId^*$

.4 $secondaryAlgorithm : SCHDTP`SecondarySchedulerAlgorithm$

.5 $classPriorityMap : AS`Id \xrightarrow{m} \mathbb{N}$

.6 $maxPriority : \mathbb{N}$

.7 $next\text{-}thread\text{-}id : SCHDTP`ThreadId$

.8 $checkingGuard : \mathbb{B}$

.9 inv mk-$Schedule\,(threads, \text{-}, perthreads, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}) \triangleq$

.10 elems $perthreads \subseteq$ dom $threads \wedge$

.11 $InOrder\,([threads\,(perthreads\,(i)).next\text{-}run \mid i \in$ inds $perthreads])$

.12 init $s \triangleq s =$ mk-$Schedule\,(\{\mapsto\}, 1,$
.13 $[],$
.14 mk-$SCHDTP`RoundRobin\,(), \{\mapsto\}, 1, 1,$ false$)$
.15 end

operations

210.0 $ClearScheduler : () \xrightarrow{o} ()$
.1 $ClearScheduler\,() \triangleq$
.2 $(threads := \{\mapsto\});$

211.0 $InitScheduler : () \xrightarrow{o} ()$
.1 $InitScheduler\,() \triangleq$
.2 $(ClearScheduler()\,;$
.3 $perthreads := [];$
.4 $SetCurThread(0)\,;$
.5 $secondaryAlgorithm :=$ if $SETTINGS`PriorityBased\,()$
.6 then mk-$SCHDTP`PriorityBased\,()$
.7 else mk-$SCHDTP`RoundRobin\,();$
.8 $next\text{-}thread\text{-}id := 1;$
.9 $StartDebugThread()\,)$

The function *InOrder* is just used to specify the order of the perthreads state component in the state invariant.
functions

212.0 $InOrder : \mathbb{N}^* \to \mathbb{B}$
.1 $InOrder\,(vals) \triangleq$
.2 len $vals < 2 \lor$
.3 let $mid =$ len $vals$ div 2 in
.4 $vals\,(mid) \leq vals\,(mid + 1) \land$
.5 $InOrder\,(vals\,(1, \ldots, mid)) \land$
.6 $InOrder\,(vals\,(mid + 1, \ldots,$ len $vals))$

1.5.1 The Scheduler Evaluation

The threads in the scheduler can be divided into three kinds: The threads that are started in the specification using the start/startlist statement, a periodic thread that is created when an instance of a class defining a periodic thread (directly or inherited) is created, or a thread that corresponds to an artificial thread, namely a debug thread. A debug thread is a thread that executes the expression/statement that the user wants to evaluate the first time after he has initialised the specification.

Each thread has its own call stack. If a thread is interrupted because of a break-point, or run-time error, and the user wants to evaluate some-thing (that is using recursive-debug) the call stack of the current thread is added with a debug command. That is, it is only if there does not exists a current thread an artificial thread is created, otherwise all debug commands are evaluated in the scope of the current thread (that is, the one that is running on the stack machine).

The state of the thread is described by *ThreadStatus*. Below is the a state machine of which commands/operations that can change the status of the thread:

Figure 1.1: Thread Status and Transitions

The operation *EvalScheduler* is called by *EvalRun* in the *DEBUG* module. It is assumed that a current thread exists, that is, the stack machine is already instantiated with a thread. This has been done by InitScheduler that called StartDebugThread.

EvalScheduler will be called after the commands *print, debug, step, stepin, singlestep* and *EvalAuxCmd*. The scheduler starts the main loop in the stack machine, and elaborates the result, in terms of which status the stack machine is in:

- The maximum number of instructions has been reached. In this case we must select and run another thread.
- A permission guard needs to be checked because a call of an operation has been detected. In case there is no permission guard the guard part of the state of the stack machine is set to nil, and the main loop of the stack machine is started again. Otherwise the permission guard code is ran. If it succeeds main loop of the stack machine is started again, otherwise, another thread should be selected and the current thread should be marked as blocked.
- A breakpoint or an interrupt has been reached. In this case the result of the execution should be returned.
- Execution has succeeded. If the thread is the debug thread the result should be returned. If the call stack of the thread is greater than zero, and the thread is a normal active thread, it must mean that we are in the middle of evaluating an recursive debug session and thus the result of the previous evaluation should be returned.

  If the call stack is zero and it is not a debug thread, then the thread has been evaluated successfully, and it is therefore terminated: that is removed from the *threads* state and another thread is selected to be run.

operations

213.0 $\mathit{EvalScheduler} : () \xrightarrow{o} \mathit{STKM}`\mathit{EvaluationState} \times [\mathit{SEM}`\mathit{VAL}]$

.1 $\mathit{EvalScheduler}\,() \triangleq$

.2 (dcl $\mathit{evalres} : \mathit{STKM}`\mathit{EvaluationState} \times [\mathit{SEM}`\mathit{VAL}]$;

.3 $\mathit{STKM}`\mathit{ResetSlice}()$;

.4 $\mathit{evalres} := \mathit{STKM}`\mathit{EvalMainLoop}\,()$;

.5 while true

```
.6    do  cases  let mk- (stack-state, -) = evalres in
.7              stack-state:
.8          ,
.9          mk-STKM‘EndOfSlice () →
.10             evalres := SelAndRunThread (mk-SCHDTP‘MaxReached (), false),
.11         mk-STKM‘Guard (fullopnm, obj) →
.12             (STKM‘ResetGuard() ;
.13              let instr = STATE‘LookUpPermis (fullopnm) in
.14              if instr = nil
.15              then evalres := STKM‘EvalMainLoop ()
.16              else (if RunGuard (instr, obj)
.17                    then (evalres := STKM‘EvalMainLoop ())
.18                    else (evalres := SelAndRunThread (mk-SCHDTP‘Blocked (fullopnm,
.19                                                                          obj),
.20                                                         false)))),
.21         mk-STKM‘Breakpoint (),
.22         mk-STKM‘Interrupt () →
.23             (return evalres ),
.24         mk-STKM‘Success () →
.25             (let mk- (org-obj-ref, org-cl) = GetObjRefAndClass (CurThreadId ()) in
.26              TIMETRACE‘LogThreadSwapOut(CurThreadId (), org-obj-ref,
.27                      org-cl) ;
.28              if IsDebugThread (CurThreadId ()) ∨ STKM‘CallStackLevel () > 0
.29              then (return evalres )
.30              else (if ¬ IsPeriodicThread (CurThreadId ())
.31                    then threads := {CurThreadId ()} ⩤ threads;
.32                    evalres := SelAndRunThread (nil , true)))
.33      end ;
.34   return evalres );
```

The operation *StartDebugThread* is used to create an artificial debug thread. It is called by the *InitScheduler* operation.

The *StartDebugThread* creates a new thread id and sets it to the current thread. The state of the now current thread is set to a running thread, with no object reference and an empty SubProgram. The debug thread's priority is set to be the maximum priority.

```
214.0 StartDebugThread : () -o-> ()
   .1 StartDebugThread () △
   .2    (let threadid = AddNewThreadId (true, nil , []) in
   .3     (SetCurThread(threadid) );
   .4     SetThreadStatus(CurThreadId (), mk-SCHDTP‘Running ()) ;
   .5     SetMaxPriority(CurThreadId ()) ;
   .6     RestoreSTKM (CurThreadId ()) );
```

The *RunGuard* operation is used to execute the instruction code for a guard. Note that it deactivates all break points before the evaluation and reactivates them afterwards. The operation

FIXME....

```
RunGuard : STKM`SubProgram × [SEM`OBJ-Ref] -o-> B
RunGuard (instr, objref) ≜
  (dcl res : STKM`EvaluationState × [SEM`VAL],
       df : STKM`DebugFlag;
   if objref ≠ nil
   then STKM`PushCurObj(objref, nil , STATE`GetClFromObjRef (objref)) ;
   STKM`PushCS(mk-STKM`DebugCmd (instr), "GuardEvaluation", nil , INTERNAL);
   DEBUG`DeActivateAllBreakpoints() ;
   df := STKM`GetDebugFlag ();
   STKM`SetDebugFlag(mk-STKM`Continue ()) ;
   SetCheckingGuard(true) ;
   res := STKM`EvalMainLoop ();
   SetCheckingGuard(false) ;
   STKM`SetDebugFlag(df) ;
   DEBUG`ActivateAllBreakpoints() ;
   STKM`PopCS() ;
   if objref ≠ nil
   then STKM`PopCurObj() ;
   let mk- (main-state, eval-res) = res in
   if main-state = mk-STKM`Success ()
   then let mk-SEM`BOOL (bool-res) = eval-res in
          return bool-res
   elseif is-STKM`Guard (main-state)
   then (RTERR`Error(RTERR`OP-IN-GUARD, nil , nil , []) ;
           error)
   else error);
```

SelAndRunThread selects the next thread which can be continued and let the stack machine execute that.

The operation takes the following parameters:

- threadstatus. This is the status that the currently thread should be set to when it is to be swapped out. If the current thread is terminated this value will be nil.
- curthread_terminated: a boolean indicating if the currently thread was terminated.

```
SelAndRunThread : [SCHDTP`ThreadStatus] × 𝔹 -o->
                  STKM`EvaluationState × [SEM`VAL]

SelAndRunThread (threadstatus, curthread-terminated) ≜
  (if ¬ curthread-terminated
   then (SetThreadStatus(CurThreadId (), threadstatus) ;
         SetThreadState(CurThreadId ()) );
   if IsPeriodicThread (CurThreadId ()) ∧ curthread-terminated
   then (SetThreadState(CurThreadId ()) ;
         SetThreadStatus(CurThreadId (), mk-SCHDTP`Sleeping ()) ;
         ResetNextRunTime(CurThreadId ()) ;
         InsertInQueue(CurThreadId ()) );
   if (¬ curthread-terminated)
   then let mk- (org-obj-ref, org-cl) = GetObjRefAndClass (CurThreadId ()) in
        TIMETRACE`LogThreadSwapOut(CurThreadId (), org-obj-ref, org-cl) ;
   def ts = SETTINGS`GetTaskSwitch () in
   if ts ≠ 0
   then STKM`IncrAbsTime(ts) ;
   let nextthread = FindNextThread () in
   if nextthread = nil
   then (RTERR`InitError(RTERR`DEADLOCK-DETECTED, CI`NilContextId) ;
         error)
   else (SetCurThread(nextthread) ;
        SetThreadStatus(CurThreadId (), mk-SCHDTP`Running ()) ;
        def mk- (new-obj-ref, new-cl) = GetObjRefAndClass (CurThreadId ()) in
        if IsQueueingThread (CurThreadId ())
        then (def cur-time = STKM`GetTime () in
              if GetNextRunTime (CurThreadId ()) < cur-time
              then TIMETRACE`LogDelayedThreadSwapIn(CurThreadId (),
                            new-obj-ref, new-cl,
                            cur-time − GetNextRunTime (CurThreadId ()))
              else (AdvanceTime(CurThreadId ()) ;
                    TIMETRACE`LogThreadSwapIn(CurThreadId (), new-obj-ref,
                            new-cl) ))
        else TIMETRACE`LogThreadSwapIn(CurThreadId (), new-obj-ref, new-cl) ;
        if IsQueueingThread (CurThreadId ())
        then (DequeueThread(CurThreadId ()) ;
              RestoreInstrAndSTKM(CurThreadId ()) )
        else RestoreSTKM(CurThreadId ()) ;
        STKM`ResetSlice() ;
        STKM`EvalMainLoop() ));
```

```
217.0  GetObjRefAndClass : SCHDTP`ThreadId -o-> [SEM`OBJ-Ref] × [AS`Name]
   .1  GetObjRefAndClass (threadid) △
   .2    let obj-ref = if IsDebugThread (threadid)
   .3                  then nil
   .4                  else GetObjRef (threadid) in
   .5    let cl = if obj-ref = nil
   .6             then nil
   .7             else STATE`GetClFromObjRef (obj-ref) in
   .8    return mk- (obj-ref, cl);
```

The operation *AdvanceTime* is used to advance time to the run time of the next queueing thread, if no other threads can be run.

```
218.0  AdvanceTime : SCHDTP`ThreadId -o-> ()
   .1  AdvanceTime (threadid) △
   .2    (def run-time = GetNextRunTime (CurThreadId ()) in
   .3     STKM`SetTime(run-time) )
   .4  pre  threadid ∈ elems perthreads ;
```

```
219.0  GetObjRef : SCHDTP`ThreadId -o-> SEM`OBJ-Ref
   .1  GetObjRef (threadid) △
   .2    return threads (threadid).obj;
```

FindNextThread checks the different threads in an order determined whether the scheduling is priority based or not. For each of the threads the status of the thread is examined and if it is blocked the premission guard must be executed and if it succeeds then that thread number can be returned. In case the status says that it is not blocked it can be chosen directly.

If it is not possible to find a thread that is schedulable nil is returned.

```
220.0  FindNextThread : () -o-> [SCHDTP`ThreadId]
   .1  FindNextThread () △
   .2    (let orders = SortTheThreads () in
   .3     for order in orders
   .4     do (for threadno in order
   .5         do if is-SCHDTP`Blocked (GetThreadStatus (threadno))
   .6            then let mk-SCHDTP`Blocked (fullopnm, obj) = GetThreadStatus (threadno),
```

```
                 instr = STATE`LookUpPermis (fullopnm) in
               (RestoreSTKM(threadno) ;
                let org-thread = CurThreadId () in
                (SetCurThread(threadno) ;
                 let guardOK = RunGuard (instr, obj) in
                 (SetCurThread(org-thread) ;
                  if guardOK
                  then return threadno
                  else skip)))
            elseif IsQueueingThread (threadno)
            then (if STKM`GetTime () ≥ GetNextRunTime (threadno)
                  then return threadno
                  else skip)
            else return threadno );
    if AreQueueingThreads ()
    then return NextQueueingThread () ;
    return nil );
```

The operation *SortTheThreads* sorts the current threads in the system based on the current scheduling algorithm being used. If round robin is being used, all of the threads are sorted at once using *SortSelectedThreads*. If priority based scheduling is being used, threads of equal priority are sorted individually and the result is returned in order of priority.

```
221.0  SortTheThreads : () -o-> SCHDTP`ThreadId**
       SortTheThreads () ≜
         (dcl res : SCHDTP`ThreadId**;
          cases secondaryAlgorithm:
            mk-SCHDTP`RoundRobin () → res := [SortSelectedThreads (dom threads)],
            mk-SCHDTP`PriorityBased () →
                 (res := [[] | i ∈ {1, ..., maxPriority}];
                  for p = maxPriority to 1 by − 1
                  do let threadsWithThisPriority = {id | id ∈ dom threads ·
                                                   threads (id).priority = p} in
                       res(maxPriority + 1 − p) := SortSelectedThreads (threadsWithThisPriority))
         end ;
         return res );
```

The operation *SortSelectedThreads* sorts the given set of thread ids in a sequence ordered by their number in the following way: The first sub sequence will contain the thread id sorted by number starting from the thread ids that are greater than the current thread id. The next sub sequence will be the thread id ordered from the lowest thread id to the current thread id (if it still exists in the *threads* state).

```
222.0  SortSelectedThreads : SCHDTP`ThreadId-set -o-> SCHDTP`ThreadId*
       SortSelectedThreads (selected) ≜
         (dcl res : SCHDTP`ThreadId* := [],
```

.3 $\quad\quad threadseq2 : SCHDTP`ThreadId^* := [id \mid id \in selected \cdot id > CurThreadId\,()]$,
.4 $\quad\quad threadseq1 : SCHDTP`ThreadId^* := [id \mid id \in selected \cdot id \leq CurThreadId\,()]$;
.5 $\quad res := threadseq2 \curvearrowright threadseq1$;
.6 $\quad$ return res);

StartNewThread takes an object reference and adds a new thread for this instance (if any threads are defined for the given class (or any of its superclasses).

223.0 $StartNewThread : SEM`OBJ\text{-}Ref \xrightarrow{o} ()$
.1 $StartNewThread\,(objref) \triangleq$
.2 $\quad$ let $clnm = STATE`GetNameOfObjRef\,(objref)$,
.3 $\quad\quad thread = STATE`LookUpThread\,(clnm)$ in
.4 $\quad$ if $thread =$ nil
.5 $\quad$ then $RTERR`Error(RTERR`NO\text{-}THREAD,$ nil , nil , $[])$
.6 $\quad$ else (let mk- $(code, dur) = thread$ in
.7 $\quad\quad\quad$ if $dur \neq$ nil
.8 $\quad\quad\quad$ then $AddPerThread(objref, clnm)$
.9 $\quad\quad\quad$ else let $instr = code$ in
.10 $\quad\quad\quad\quad$ let $threadid = AddNewThreadId\,($false$, objref, instr)$ in
.11 $\quad\quad\quad\quad (STATE`SetObjectThreadId(objref, threadid)$;
.12 $\quad\quad\quad\quad\; SetThreadPriority(threadid, clnm)$;
.13 $\quad\quad\quad\quad\; SetThreadStatus(threadid,$ mk-$SCHDTP`Sleeping\,())\,))$;

When we add a periodic thread, we acquire the threads code and periodic duration, and a thread id for the thread. We set its priority and status. It is then added to the periodic threads in the system, as they are currently treated differently to procedural threads. The thread's state is initialized to be empty, and then the thread is inserted into the queue of periodic threads.

224.0 $AddPerThread : SEM`OBJ\text{-}Ref \times AS`Name \xrightarrow{o} ()$
.1 $AddPerThread\,(objref, clnm) \triangleq$
.2 $\quad$ let mk- $(instr, dur) = STATE`LookUpThread\,(clnm)$,
.3 $\quad\quad threadid = AddNewThreadId\,($false$, objref, instr)$ in
.4 $\quad (STATE`SetObjectThreadId(objref, threadid)$;
.5 $\quad\; SetThreadPriority(threadid, clnm)$;
.6 $\quad\; SetThreadStatus(threadid,$ mk-$SCHDTP`Sleeping\,())$;
.7 $\quad\; SetPerThread(threadid, dur, instr)$;
.8 $\quad\; InsertInQueue(threadid)$);

The operation *SetPerThread* sets those thread info fields which are relevant to periodic threads. Of course if the thread is procedural, these fields are set to be nil.

```
225.0 SetPerThread : SCHDTP`ThreadId × [ℕ] × STKM`SubProgram →° ()
SetPerThread (threadid, dur, instr) △
  if dur = nil
  then (threads(threadid).period := nil ;
        threads(threadid).next-run := nil ;
        threads(threadid).periodBody := nil )
  else def curtime = STKM`GetTime () in
       (threads(threadid).period := dur;
        threads(threadid).next-run := curtime + dur;
        threads(threadid).periodBody := instr);
```

All the periodic threads in the system are stored in a queue - *perthreads* - which is ordered on the next run time of each thread. The operation *InsertInQueue* inserts a thread into this queue in the correct position with respect to its next run time.

```
226.0 InsertInQueue : SCHDTP`ThreadId →° ()
InsertInQueue (threadid) △
  if perthreads = []
  then perthreads := [threadid]
  else let next-run = threads (threadid).next-run in
       if next-run < threads (hd perthreads).next-run
       then perthreads := [threadid] ⁀ perthreads
       else let num-perthreads = len perthreads in
            (for i = 1 to num-perthreads
             do if next-run < threads (perthreads (i)).next-run
                then (dcl lhs : SCHDTP`ThreadId* := perthreads (1, ..., i − 1),
                          rhs : SCHDTP`ThreadId* := perthreads (i, ..., num-perthreads);
                      perthreads := lhs ⁀ [threadid] ⁀ rhs;
                      return );
             perthreads := perthreads ⁀ [threadid]);
```

The operation *DequeueThread* removes a thread from the queue of periodic threads.

```
227.0 DequeueThread : SCHDTP`ThreadId →° ()
DequeueThread (threadid) △
  (perthreads := [perthreads (i) | i ∈ inds perthreads ·
                               perthreads (i) ≠ threadid])
pre threadid ∈ elems perthreads ;
```

A thread is said to be queueing if it is an element of *perthreads*.

```
228.0 AreQueueingThreads : () →° 𝔹
AreQueueingThreads () △
  return perthreads ≠ [];
```

The operation *NextQueueingThread* delivers the queued thread which is due to be run soonest.

229.0 $\mathit{NextQueueingThread} : () \xrightarrow{o} \mathit{SCHDTP}`\mathit{ThreadId}$
.1 $\mathit{NextQueueingThread}\,() \triangleq$
.2 $\quad \textsf{return hd } \mathit{perthreads};$

The operation *IsQueueingThread* tests whethe a particular thread is queueing or not.

230.0 $\mathit{IsQueueingThread} : \mathit{SCHDTP}`\mathit{ThreadId} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{IsQueueingThread}\,(\mathit{threadid}) \triangleq$
.2 $\quad \textsf{return } \mathit{threadid} \in \textsf{elems } \mathit{perthreads};$

The operation *IsPeriodicThread* tests whether a given thread is periodic or not.

231.0 $\mathit{IsPeriodicThread} : \mathit{SCHDTP}`\mathit{ThreadId} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{IsPeriodicThread}\,(\mathit{threadid}) \triangleq$
.2 $\quad \textsf{return } \mathit{threadid} \in \textsf{dom } \mathit{threads} \wedge$
.3 $\qquad\qquad \mathit{threads}\,(\mathit{threadid}).\mathit{period} \neq \textsf{nil}\ ;$

The operation *ResetNextRunTime* takes a thread id and increments its next_run component in the thread info map by the threads period. Thus this operation assumes that the thread is periodic.

232.0 $\mathit{ResetNextRunTime} : \mathit{SCHDTP}`\mathit{ThreadId} \xrightarrow{o} ()$
.1 $\mathit{ResetNextRunTime}\,(\mathit{threadid}) \triangleq$
.2 $\quad (\textsf{dcl } \mathit{cur\text{-}time} : \mathbb{N} := \mathit{STKM}`\mathit{GetTime}\,();$
.3 $\quad\ \textsf{while } \mathit{threads}\,(\mathit{threadid}).\mathit{next\text{-}run} \leq \mathit{cur\text{-}time}$
.4 $\quad\ \textsf{do } \mathit{threads}(\mathit{threadid}).\mathit{next\text{-}run} := \mathit{threads}\,(\mathit{threadid}).\mathit{next\text{-}run} +$
.5 $\qquad\qquad\qquad\qquad \mathit{threads}\,(\mathit{threadid}).\mathit{period});$

The operation *GetNextRunTime* takes a thread id for a periodic thread, and returns the next time at which the thread should be run.

233.0 $\mathit{GetNextRunTime} : \mathit{SCHDTP}`\mathit{ThreadId} \xrightarrow{o} \mathbb{N}$
.1 $\mathit{GetNextRunTime}\,(\mathit{threadid}) \triangleq$
.2 $\quad \textsf{return } \mathit{threads}\,(\mathit{threadid}).\mathit{next\text{-}run}$
.3 $\textsf{pre } \mathit{threads}\,(\mathit{threadid}).\mathit{period} \neq \textsf{nil}\ ;$

The operation *SetThreadPriority* takes a thread id and a class name, and uses the classPriorityMap to set the priority field for this thread in the thread info map.

234.0 $\mathit{SetThreadPriority} : \mathit{SCHDTP}`\mathit{ThreadId} \times \mathit{AS}`\mathit{Name} \xrightarrow{o} ()$

.1 $\mathit{SetThreadPriority}\,(\mathit{threadid}, \mathit{clnm}) \triangleq$

.2 $\quad \mathsf{let}\ \mathit{id} = \mathsf{hd}\ \mathit{clnm.ids}\ \mathsf{in}$

.3 $\quad \mathsf{let}\ \mathit{priority} = \mathsf{if}\ \mathit{id} \in \mathsf{dom}\ \mathit{classPriorityMap}$

.4 $\qquad\qquad\qquad \mathsf{then}\ \mathit{classPriorityMap}\,(\mathit{id})$

.5 $\qquad\qquad\qquad \mathsf{else}\ \mathit{Default\text{-}priority}\ \mathsf{in}$

.6 $\quad \mathit{threads}(\mathit{threadid}).\mathit{priority} := \mathit{priority};$

The operation *SetMaxPriority* simply sets a threads priority to be the maximum one. This is only used for the debug thread.

235.0 $\mathit{SetMaxPriority} : \mathit{SCHDTP}`\mathit{ThreadId} \xrightarrow{o} ()$

.1 $\mathit{SetMaxPriority}\,(\mathit{threadid}) \triangleq$

.2 $\quad \mathit{threads}(\mathit{threadid}).\mathit{priority} := \mathit{maxPriority};$

The *KillThread* operation is called when the reference counter for an object has reached zero and the object is destroyed. Its corresponding thread (if any) is then also killed. In case the object reference does not have a thread associated with it this operation will have no effect.

236.0 $\mathit{KillThread} : \mathit{SEM}`\mathit{OBJ\text{-}Ref} \xrightarrow{o} ()$

.1 $\mathit{KillThread}\,(\mathit{objref}) \triangleq$

.2 $\quad \mathsf{let}\ \mathit{threadid} = \mathit{STATE}`\mathit{Lookup\text{-}obj\text{-}tab}\,(\mathit{objref}).\mathit{threadid}\ \mathsf{in}$

.3 $\quad \mathit{threads} := \{\mathit{threadid}\} \lhd\!\!\!- \mathit{threads};$

The operation *CurThreadId* return the thread id of the current thread, that is, the one that is running on the stack machine currently.

237.0 $\mathit{CurThreadId} : () \xrightarrow{o} \mathit{SCHDTP}`\mathit{ThreadId}$

.1 $\mathit{CurThreadId}\,() \triangleq$

.2 $\quad \mathsf{return}\ \mathit{curthread};$

The operation *ExistsThreads* checks if there exists threads. The operation is used by *EvalDebug* in order to decide if a debug thread should be started. This should only happen if the debug command is the first debug command after the interpreter has been initialised.

238.0 $\mathit{ExistsThreads} : () \xrightarrow{o} \mathbb{B}$

.1 $\mathit{ExistsThreads}\,() \triangleq$

.2 $\quad \mathsf{return}\ \neg\,(\mathit{threads} = \{\mapsto\});$

GiveAllThreads returns the thread information for all threads in the scheduler.

239.0 $\mathit{GiveAllThreads} : () \xrightarrow{o} \mathit{SCHDTP}`\mathit{Threads}$

.1 $\mathit{GiveAllThreads}\,() \triangleq$
.2 $\quad(\mathsf{return}\ \{\mathit{id} \mapsto \mathit{threads}\,(\mathit{id}) \mid$
.3 $\qquad\qquad\quad \mathit{id} \in \mathsf{dom}\ \mathit{threads}\}\ );$

The operation *SelThread* is used to support the user command selthread. The operation makes the scheduler schedule a certain thread id. It does not check the status of the thread, that is, if the thread is blocked, this thread is scheduled and the guard is evaluated.

240.0 $\mathit{SelThread} : \mathit{SCHDTP}`\mathit{ThreadId} \xrightarrow{o} ()$

.1 $\mathit{SelThread}\,(\mathit{selid}) \triangleq$
.2 $\quad(\mathsf{if}\ \neg\ \mathit{selid} \in \mathsf{dom}\ \mathit{threads}$
.3 $\quad\ \mathsf{then}\ \mathit{RTERR}`\mathit{InitError}(\mathit{RTERR}`\mathit{SEL\text{-}NONVALID\text{-}THREADID}, \mathit{CI}`\mathit{NilContextId});$
.4 $\quad\ \mathit{SetThreadState}(\mathit{CurThreadId}\,())\,;$
.5 $\quad\ \mathit{SetThreadStatus}(\mathit{CurThreadId}\,(), \mathsf{mk\text{-}}\mathit{SCHDTP}`\mathit{Sleeping}\,())\,;$
.6 $\quad\ \mathit{SetCurThread}(\mathit{selid})\,;$
.7 $\quad\ \mathit{RestoreSTKM}(\mathit{CurThreadId}\,())\,;$
.8 $\quad\ \mathit{SetThreadStatus}(\mathit{CurThreadId}\,(), \mathsf{mk\text{-}}\mathit{SCHDTP}`\mathit{Running}\,())\ );$

1.6 Primary Scheduling Algorithm

In this section we provide auxiliary operations for the primary scheduling algorithm. Note that since the primary scheduling algorithm determines when a thread should be *descheduled*, this operations are used in the stack machine itself.

241.0 $\mathit{Deschedule} : \mathit{STKM}`\mathit{EvaluatorStatus} \xrightarrow{o} \mathbb{B}$

.1 $\mathit{Deschedule}\,(\mathit{threadstate}) \triangleq$
.2 $\quad\mathsf{if}\ \mathit{CheckingGuard}\,()$
.3 $\quad\mathsf{then\ return\ false}$
.4 $\quad\mathsf{else}\ \ \mathsf{cases}\ \ \mathit{SETTINGS}`\mathit{GetPrimaryAlgorithm}\,():$
.5 $\qquad\qquad \mathsf{mk\text{-}}\mathit{SCHDTP}`\mathit{PureCooperative}\,() \rightarrow \mathsf{return\ false},$
.6 $\qquad\qquad \mathsf{mk\text{-}}\mathit{SCHDTP}`\mathit{TimeSlice}\,() \rightarrow$
.7 $\qquad\qquad\quad \mathsf{return}\ \mathit{STKM}`\mathit{GetTime}\,() - \mathit{threadstate.release\text{-}time} >$
.8 $\qquad\qquad\qquad\quad \mathit{SETTINGS}`\mathit{GetTimeSlice}\,(),$
.9 $\qquad\qquad \mathsf{mk\text{-}}\mathit{SCHDTP}`\mathit{InstrnumSlice}\,() \rightarrow$
.10 $\qquad\qquad\quad \mathsf{return}\ \mathit{threadstate.instrno} \geq \mathit{SETTINGS}`\mathit{GetMaxInstr}\,()$
.11 $\qquad\quad \mathsf{end};$

242.0 $\mathit{IncInstrnum} : \mathit{STKM}`\mathit{EvaluatorStatus} \xrightarrow{o} \mathit{STKM}`\mathit{EvaluatorStatus}$

.1 $\mathit{IncInstrnum}\,(\mathit{threadstate}) \triangleq$
.2 $\quad(\mathsf{dcl}\ \mathit{localstate} : \mathit{STKM}`\mathit{EvaluatorStatus} := \mathit{threadstate};$

```
.3     if ¬ CheckingGuard () ∧
.4        SETTINGS`GetPrimaryAlgorithm () = mk-SCHDTP`InstrnumSlice ()
.5     then localstate.instrno := localstate.instrno + 1;
.6     return localstate );
```

```
243.0  EndOfSliceReached : STKM`EvaluatorStatus →(o) 𝔹
   .1  EndOfSliceReached (threadstate) ≜
   .2    cases SETTINGS`GetPrimaryAlgorithm ():
   .3      mk-SCHDTP`PureCooperative () → return false,
   .4      mk-SCHDTP`TimeSlice () →
   .5            return STKM`GetTime () − threadstate.release-time >
   .6                   SETTINGS`GetTimeSlice (),
   .7      mk-SCHDTP`InstrnumSlice () →
   .8            return threadstate.instrno ≥ SETTINGS`GetMaxInstr ()
   .9    end;
```

```
244.0  InitSlice : STKM`EvaluatorStatus →(o) STKM`EvaluatorStatus
   .1  InitSlice (threadstate) ≜
   .2    (dcl localstate : STKM`EvaluatorStatus := threadstate;
   .3      cases SETTINGS`GetPrimaryAlgorithm ():
   .4        mk-SCHDTP`TimeSlice () → localstate.release-time := STKM`GetTime (),
   .5        mk-SCHDTP`InstrnumSlice () → localstate.instrno := 0,
   .6        others → skip
   .7      end;
   .8      return localstate );
```

```
245.0  CheckingGuard : () →(o) 𝔹
   .1  CheckingGuard () ≜
   .2    return checkingGuard;
```

```
246.0  SetCheckingGuard : 𝔹 →(o) ()
   .1  SetCheckingGuard (b) ≜
   .2    checkingGuard := b;
```

1.7 Auxiliary operations on the State

The operations in this section are all operations that are auxiliary operations that sets information on the thread state or stack machine state, or that gets information on these states.

The operation *RestoreSTKM* instantiate the state of the stack machine to the the state stored in the thread information of *threadid*.

247.0 $RestoreSTKM : SCHDTP`ThreadId \xrightarrow{o} ()$
.1 $RestoreSTKM\ (threadid) \triangleq$
.2 $(STKM`Instantiate(threads\ (threadid).stackeval,$ nil $)\ );$

RestoreInstrAndSTKM is similar but also instantiates the instruction code for queueing threads which are being started. Note that at the implementation level this uses the function CalculateArgumentsForPushCS. However this function is not available at the specification level.

248.0 $RestoreInstrAndSTKM : SCHDTP`ThreadId \xrightarrow{o} ()$
.1 $RestoreInstrAndSTKM\ (threadid) \triangleq$
.2 let $instr = threads\ (threadid).periodBody$ in
.3 (if $threads\ (threadid).stackeval.call\text{-}stack = []$
.4 then let $threadstate = threads\ (threadid).stackeval$ in
.5 let $env\text{-}l\text{-}h =$ len $threadstate.env\text{-}l,$
.6 $typeinst\text{-}h =$ len $threadstate.typeinst,$
.7 $os\text{-}h =$ len $threadstate.os,$
.8 $cur\text{-}mod\text{-}obj\text{-}l\text{-}h =$ len $threadstate.obj\text{-}l,$
.9 $cid = threadstate.curCid$ in
.10 $threads(threadid).stackeval.call\text{-}stack :=$ [mk-$STKM`CallStackItem$ (INTERNAL, mk-$STKM`Debug$
.11 "$ThreadStart$", nil , nil , $cid,$
.12 $env\text{-}l\text{-}h, typeinst\text{-}h, os\text{-}h, cur\text{-}mo$
.13 $STKM`Instantiate(threads\ (threadid).stackeval, instr)\ );$

This operation returns if the thread id *threadid* is a artificial debug thread.

249.0 $IsDebugThread : SCHDTP`ThreadId \xrightarrow{o} \mathbb{B}$
.1 $IsDebugThread\ (threadid) \triangleq$
.2 (return $threads\ (threadid).debug\text{-}thread$);

AddNewThreadId creates a new thread in the *threads* state with some initial settings, the operation returns the thread id of the new thread.

250.0 $AddNewThreadId : \mathbb{B} \times [SEM`OBJ\text{-}Ref] \times STKM`SubProgram \xrightarrow{o} SCHDTP`ThreadId$
.1 $AddNewThreadId\,(debug\text{-}thread, objref, instr) \triangleq$
.2 (let $threadid = GetNextThreadId\,()$ in
.3 $(threads := threads\ \dagger$
.4 $\{threadid \mapsto$
.5 mk-$SCHDTP`ThreadInfo\,(debug\text{-}thread,$
.6 $objref,$
.7 mk-$SCHDTP`Sleeping\,(),$
.8 $STKM`InitEvaluatorStatus\,(instr, objref),$
.9 $Default\text{-}priority,$
.10 nil ,
.11 nil ,
.12 nil ,
.13 nil $)\};$
.14 return $threadid$));

SetThreadStatus sets the thread status of *threadid* to *tstatus*.

251.0 $SetThreadStatus : SCHDTP`ThreadId \times SCHDTP`ThreadStatus \xrightarrow{o} ()$
.1 $SetThreadStatus\,(threadid, tstatus) \triangleq$
.2 $(threads(threadid).status := tstatus);$

GetThreadStatus returns the thread status of *threadid*.

252.0 $GetThreadStatus : SCHDTP`ThreadId \xrightarrow{o} SCHDTP`ThreadStatus$
.1 $GetThreadStatus\,(threadid) \triangleq$
.2 (return $threads\,(threadid).status$);

SetThreadState sets the state of the thread to the current state of the stack machine and of the thread relevant state in module *STATE*.

253.0 $SetThreadState : SCHDTP`ThreadId \xrightarrow{o} ()$
.1 $SetThreadState\,(threadid) \triangleq$
.2 $(threads(threadid).stackeval := STKM`GetEvaluatorState\,());$

GetNextThreadId returns the next avaialable thread id.

254.0 $GetNextThreadId : () \xrightarrow{o} SCHDTP`ThreadId$
.1 $GetNextThreadId\,() \triangleq$
.2 (dcl $the\text{-}id : SCHDTP`ThreadId := next\text{-}thread\text{-}id;$
.3 $next\text{-}thread\text{-}id := next\text{-}thread\text{-}id + 1;$
.4 return $the\text{-}id$);

SetCurThread sets the *id* to the current thread id.

255.0 $SetCurThread : SCHDTP`ThreadId \xrightarrow{o} ()$
.1 $SetCurThread\,(id) \triangleq$
.2 $\quad (curthread := id)$

The value *Default_priority* is a constant describing the default priority of a thread.
values

256.0 $Default\text{-}priority = 1$

1.8 Priority-based Scheduling

In this section we provide auxiliary operations for priority based scheduling. Currently information about thread priority is stored in a separate file and read in to the toolbox.

1.8.1 Priority File

A syntactically correct priority file is well-formed if no class occurs more than once in it. A map from class names to priorities can then be generated. If a given class is not listed, then it is assumed to take the default (lowest) priority (that is, priority 1).
functions

257.0 $WellFormedPriorityFile : SCHDTP`PriorityFile \rightarrow \mathbb{B}$
.1 $WellFormedPriorityFile\,(pf) \triangleq$
.2 $\quad \mathsf{card}\ \{pe.clnm \mid pe \in \mathsf{elems}\ pf\} = \mathsf{len}\ pf;$

The function *MakePriorityMap* converts a well-formed priority file into a map from class names to priority values.

258.0 $MakePriorityMap : SCHDTP`PriorityFile \rightarrow AS`Id \xrightarrow{m} \mathbb{N}$
.1 $MakePriorityMap\,(pf) \triangleq$
.2 $\quad \{clnm \mapsto priority \mid$
.3 $\qquad \mathsf{mk}\text{-}SCHDTP`PriorityEntry\,(clnm, priority) \in$
.4 $\qquad \mathsf{elems}\ pf\}$

Operations are provided for getting and setting the priority map. Note that when setting the priority map, the maximum priority value is set to be 1 greater than any of the priorities in the map. This ensures that the debug thread always has strictly higher priority than any other thread in the system.

operations

259.0 $GetPriorityMap : () \xrightarrow{o} AS`Id \xrightarrow{m} \mathbb{N}$
.1 $GetPriorityMap\,() \triangleq$
.2 return $classPriorityMap$;

260.0 $SetPriorityMap : AS`Id \xrightarrow{m} \mathbb{N} \xrightarrow{o} ()$
.1 $SetPriorityMap\,(pm) \triangleq$
.2 $(classPriorityMap := pm;$
.3 $maxPriority := max\,(\mathsf{rng}\ pm \cup \{Default\text{-}priority\}) + 1)$

functions

261.0 $max : \mathbb{N}\text{-}\mathsf{set} \to \mathbb{N}$
.1 $max\,(s) \triangleq$
.2 let $x \in s$ in
.3 if card $s = 1$
.4 then x
.5 else let $m = max\,(s \setminus \{x\})$ in
.6 if $x \geq m$
.7 then x
.8 else m

end $SCHD$

1.9 Scheduler Types

This module contains all the types used for the scheduler. They are placed in seperate module in order to be able to code generate the types seperately.
module $SCHDTP$

imports

262.0 from AS
263.0 types $AS`Id$;
.1 $AS`Name$,
264.0 from SEM
265.0 types $OBJ\text{-}Ref$,
266.0 from $STKM$
267.0 types $SubProgram$;
.1 $EvaluatorStatus$

exports all

definitions

The type *Threads* describes the current threads in the interpreter. It is a map from a thread id to a record containing the relevant information of the thread. The *Threads* data type should only contain the threads that at some point is schedulable. That is, the intension is that the *Threads* should not contain terminated threads.

types

268.0 $Threads = SCHDTP`ThreadId \xrightarrow{m} SCHDTP`ThreadInfo;$

The information relevant for a thread is described by the record *ThreadInfo*. The thread information is:

debug_thread: a boolean indicating if the thread is an artificial debug thread.

obj_ref: this field contains the object reference for the given thread. In case the thread is a debug thread this value will be nil.

status: The status of the thread, it may e.g. be running, sleeping, etc. See also the data type *ThreadStatus*.

stackeval: Whenever a thread is not scheduled - that is not running in the stack machine - this field contains the state of the stack machine as it was when it was swapped out from the stack machine.

currstate: When a thread is not scheduled - that is not running - this field contains the thread specific state like for instance, block-environments, scope etc. as it was just before it was swapped out from the stack machine. When the trhead is running in the stack machine this field has the value nil.

priority: A field to be used for describing the priority of the thread. Currently priority-based scheduling is not supported.

269.0 $ThreadInfo :: debug\text{-}thread : \mathbb{B}$
.1 $obj : [SEM`OBJ\text{-}Ref]$
.2 $status : ThreadStatus$
.3 $stackeval : STKM`EvaluatorStatus$
.4 $priority : \mathbb{N}$
.5 $period : [\mathbb{N}]$
.6 $next\text{-}run : [\mathbb{N}]$
.7 $periodBody : [STKM`SubProgram]$
.8 $release\text{-}time : [\mathbb{N}]$
.9 inv mk-$ThreadInfo\ (debug\text{-}thread, obj, status, stackeval, priority,$
.10 $period, next\text{-}run, \text{-}, \text{-}) \triangleq$
.11 true $\wedge$
.12 $next\text{-}run = \mathsf{nil} \Leftrightarrow period = \mathsf{nil}$;

The type *ThreadStatus* describes the state that each thread can be in.

270.0 $ThreadStatus = Running \mid Blocked \mid MaxReached \mid Sleeping;$

If the thread is running in the stack machine the status of the thread is *Running*.

271.0 $Running ::\ ;$

The thread is blocked in case a permission predicate must be evaluated before it is allowed to run. The *opnm* describes the name of the operation that blocks the thread, the *objref* is the object scope that the operation should be run in, if it is nil, it means the current object scope.

272.0 $Blocked :: opnm : AS`Name$
.1 $objref : [SEM`OBJ\text{-}Ref];$

The thread is in a *MaxReached* state in the thread is being swapped out because maximum number of instructions have been reached.

273.0 $MaxReached ::\ ;$

The thread is in a *Sleeping* state in case the thread can be scheduled, but is actually not running on the stack machine. So the difference between a Sleeping and a Running thread is that the Running thread is actually running in the stack-machine where as the sleeping threads are schedulable, but is not currently running in the stack machine.

274.0 $Sleeping ::\ ;$

A thread id is a natural number.

275.0 $ThreadId = \mathbb{N};$

1.9.1 Priorities

In this section we define the abstract syntax for the file in which thread priorities are defined.

276.0 $PriorityFile = PriorityEntry^*;$

277.0 $PriorityEntry :: clnm : AS`Id$
.1 $priority : \mathbb{N};$

A priority file consists of a number of priority entries. Each priority entry relates a class name to a numeric priority, where 1 (the default) is the lowest priority.

1.9.2 Scheduling Algorithms

We define two types of scheduling algorithm: primary and secondary. The primary scheduling algorithm defines the manner in which a thread may be descheduled.

278.0 $PrimarySchedulerAlgorithm = PureCooperative \mid$
.1 $TimeSlice \mid$
.2 $InstrnumSlice;$

279.0 $PureCooperative ::\ \ ;$

280.0 $TimeSlice ::\ \ ;$

281.0 $InstrnumSlice ::\ \ ;$

Pure Cooperative The only way in which a thread may be descheduled is by termination of the thread, or reaching a permission predicate that is false. The onus is on the specifier to ensure (any) fairness.

Time Slice Each thread has a maximum amount of simulated time which it may executed for.

Instrnum Slice Each thread has a maximum number of stack instructions which it may be executed for.

We define a union type for secondary scheduling algorithms, to make addition of new algorithms easy. Currently only two algorithms are supported:

Round robin In which each thread is given equal opportunity to be selected by the scheduler.

Priority based In which the scheduler attempts to schedule threads of higher priority in preference to those with lower priority. Note that this does *not* imply that lower priority threads are automatically interrupted when a higher priority one is enabled.

282.0 $SecondarySchedulerAlgorithm = RoundRobin \mid PriorityBased;$

283.0 $RoundRobin ::\ \ ;$

284.0 $PriorityBased ::$

end $SCHDTP$

1.10 The Core Stack Evaluator

The module *STKM* defines the core parts of the stack evaluator. That is, the the necessary environment for evaluating the different stack instructions defined in module *INSTR*. The evaluator uses an evaluation stack by which programs are evaluated and a separate call stack to keep track of nested function calls.

module $STKM$

imports

285.0 from AS all ,

286.0 from CI all ,

287.0 from IO all ,

288.0 from AUX all ,

289.0 from PAT all ,

290.0 from REP all ,

291.0 from SEM all ,

292.0 from $CMPL$ all ,

293.0 from $EXPR$ all ,

294.0 from $SCHD$ all ,

295.0 from $DEBUG$ all ,

296.0 from $INSTR$ all ,

297.0 from $RTERR$ all ,

298.0 from $STATE$ all ,

299.0 from $GLOBAL$ all ,

300.0 from $SCHDTP$ all ,

301.0 from $DEBUGTP$ all ,

302.0 from $INSTRTP$ all ,

303.0 from $TIMEMAP$ all ,

304.0 from $SETTINGS$ all ,

305.0 from $TIMEPARSER$ all

exports all

definitions

1.10.1 The Stack Machine Environment

The following type declarations and the module state *sigma* together define the environment of the evaluator. A *SubProgram* is simply a sequence of instructions and values (values being either semantic values or names of identifiers). A *SubProgram* will be generated for each function and operation of the specification.

While a *SubProgram* is being evaluated the current position is recorded in the program counter, which is simply an index into the sub program instruction sequence.

types

306.0 $SubProgram = ProgItem^*;$

307.0 $ProgItem = SEM`VAL \mid AS`Name \mid INSTRTP`Instruction;$

308.0 $ProgramCounter = \mathbb{N};$

309.0 $SubProgramId = \mathbb{N};$

The evaluation stack can contain different kinds of values, i.e. semantic values, identifiers or sequence of semantic values. In addition the evaluation stack is used for patterns, state designators and for sets of block environments. Values are pushed onto the stack, while instructions may pop a number of values of the stack and push their result back onto the stack.

310.0 $EvalStack = EvalStackItem^*;$

311.0 $EvalStackItem = SEM`VAL \mid AS`Name \mid AS`OldName \mid SemArgList \mid Pattern \mid$
.1 $StateDesignator \mid SEM`BlkEnv\text{-set} \mid \mathbb{N}^* \mid$
.2 $AS`FctTypeInstExpr;$

312.0 $SemArgList = SEM`VAL^*;$

1.10.2 Patterns

The evaluation stack can also contain a pattern. A pattern has almost the same structure as the corresponding AS‘Pattern, the only difference is that a *MatchVal* contains a semantic value rather than an expression.

313.0 $Pattern = PatternName \mid MatchVal \mid SetPattern \mid SeqPattern \mid$
.1 $TuplePattern \mid RecordPattern;$

314.0 $PatternName :: nm : [AS`Name]$
.1 $cid : \text{-}CI`ContextId;$

315.0 $MatchVal :: val : SEM`VAL;$

316.0 $SetPattern = SetEnumPattern \mid SetUnionPattern;$

317.0 $SetEnumPattern :: els : Pattern^*$
.1 $cid : \text{-}CI`ContextId;$

318.0 $SetUnionPattern :: lp : Pattern$
.1 $rp : Pattern$
.2 $cid : \text{-}CI`ContextId;$

319.0 $SeqPattern = SeqEnumPattern \mid SeqConcPattern;$

320.0 $SeqEnumPattern :: els : Pattern^*$
.1 $cid : \text{-}CI`ContextId;$

321.0 $SeqConcPattern :: lp : Pattern$
.1 $rp : Pattern$
.2 $cid : \text{-} CI`ContextId;$

322.0 $TuplePattern :: fields : Pattern^*$
.1 $cid : \text{-} CI`ContextId;$

323.0 $RecordPattern :: nm : AS`Name$
.1 $fields : Pattern^*$
.2 $cid : \text{-} CI`ContextId$

functions

324.0 $IsPat : EvalStackItem \to \mathbb{B}$
.1 $IsPat\,(sctitem) \triangleq$
.2 is-$MatchVal\,(sctitem) \lor$
.3 is-$PatternName\,(sctitem) \lor$
.4 is-$SetEnumPattern\,(sctitem) \lor$
.5 is-$SetUnionPattern\,(sctitem) \lor$
.6 is-$SeqConcPattern\,(sctitem) \lor$
.7 is-$SeqEnumPattern\,(sctitem) \lor$
.8 is-$TuplePattern\,(sctitem) \lor$
.9 is-$RecordPattern\,(sctitem)$

1.10.3 State Designators

In the same way as for patterns it is necessary to treat state designators seperately.
types

325.0 $StateDesignator = AS`Name \mid FieldRef \mid MapOrSeqRef;$

326.0 $FieldRef :: var : StateDesignator$
.1 $sel : AS`Name$
.2 $cid : \text{-} CI`ContextId;$

327.0 $MapOrSeqRef :: var : StateDesignator$
.1 $arg : SEM`VAL$
.2 $cid : \text{-} CI`ContextId$

functions

328.0 $IsSD : EvalStackItem \to \mathbb{B}$
.1 $IsSD\,(sctitem) \triangleq$
.2 is-$AS`Name\,(sctitem) \lor$
.3 is-$FieldRef\,(sctitem) \lor$
.4 is-$MapOrSeqRef\,(sctitem)$

1.10.4 The Call Stack

******** The following has been updated on 2/5/2000 by Jesper K. Pedersen ********

The call stack is used for a number of things. These includes:

- Keeping information about functions at higher levels when calling sub-functions, so that it is possible to restore the state of the stack interpreter when the sub-function exits and the caller must continue on.
- Keeping information about the state of the stack interpreter prior to a recursive debug. This is necessary to be able to restore the state in case of a runtime error in the recursive debug.
- On the call stack information is also located, which makes it possible to go up and down through the function invocations.
- On the call stack information is kept which makes it possible to show a back trace stack.
- In certain situations the interpreter needs to evaluate something which resembles a function call, and it therefore uses the call stack for this too. (These situations includes evaluation of global values, invariants for type names and more - see the function DEBUG‘EvalAuxCommand)

An alternative to using only one stack for both recursive debugs and recursive function calls would have been to have a debug-stack, where each of the elements were call-stacks. Each element on the debug stack would then stem from one recursive debug, and each element on a specific call stack would be one function invocation. This solution was, however, discarded due to efficiency consideration in the implementation.

The CallStackItem record has the following fields:

type Information about which kind this record is.

code Code for the current function (This need more description!)

pc Program Counter for the function which has called the function the given record represent. That is when a sub-function is invoked a new record is created and the program counter in the caller is inserted into this record.

nmOrDesc This field is used when showing the back trace stack. It is either a textual representation of the function name or another text describing the given record. (E.g. “debug f(x)” or “Global initialization”)

arg_l This variable contains the arguments for the function call this record describes. (This field is only used for function invocation records)

evalstate This field is used to keep a copy of the entire state of the interpreter. This is used when doing a recursive debug.

oldCid This field contains the context id of the position in the *previous* function. That is the function calling the function which this record describes. In case this record describes a recursive debugging, then the context id is the location where this recursive debugging is started from. This information is required when going up/down in the back trace stack.

env_l_h, typeinst_h, os_h, cur_mod_obj_l_h These four variables contain the hight of the environment stack, the typeinst stack, the old state stack, and the current module or object list stack, respectively. These heights are required to be able to cut of the stacks to the level they had at the point a given function was invoked (needed when going up and down in the back trace stack). See the section on the Up/Down commands (sec. 1.10.13) and the section on the EvaluatorStatus record (sec. 1.10.7) for more information about this.

******** End of updated documentation ****************

types

329.0 $\mathit{CallStack} = \mathit{CallStackItem}^*;$

330.0 $\mathit{CSItemType} = \mathrm{DEBUGCMD} \mid \mathrm{FNOP} \mid \mathrm{INTERNAL};$

331.0 $\mathit{CallStackItem} :: \mathit{type} : \mathit{CSItemType}$
.1 $\mathit{code} : \mathit{Code}$
.2 $\mathit{pc} : \mathit{ProgramCounter}$
.3 $\mathit{nmOrDesc} : \mathit{AS}`\mathit{Name} \mid \mathsf{char}^*$
.4 $\mathit{arg\text{-}l} : [\mathit{SEM}`\mathit{VAL}^*]$
.5 $\mathit{evalstate} : [\mathit{EvaluatorStatus}]$
.6 $\mathit{oldCid} : \mathit{CI}`\mathit{ContextId}$
.7 $\mathit{env\text{-}l\text{-}h} : [\mathbb{N}]$
.8 $\mathit{typeinst\text{-}h} : [\mathbb{N}]$
.9 $\mathit{os\text{-}h} : [\mathbb{N}]$
.10 $\mathit{cur\text{-}mod\text{-}obj\text{-}l\text{-}h} : [\mathbb{N}]$

.11 $\mathsf{inv}\ \mathsf{mk\text{-}}\mathit{CallStackItem}\,(\mathit{type}, \text{-}, \text{-}, \mathit{nmOrDesc}, \mathit{arg\text{-}l}, \mathit{evalstate}, \mathit{oldCid}, \mathit{env\text{-}l\text{-}h}, \mathit{typeinst\text{-}h}, \mathit{os\text{-}h},$
.12 $\mathit{cur\text{-}mod\text{-}obj\text{-}l\text{-}h}) \triangleq$
.13 $(\mathit{type} = \mathrm{DEBUGCMD} \Rightarrow$
.14 $\mathit{arg\text{-}l} = \mathsf{nil} \wedge$
.15 $\mathit{evalstate} \neq \mathsf{nil} \wedge$
.16 $\neg\, \mathsf{is\text{-}}\mathit{AS}`\mathit{Name}\,(\mathit{nmOrDesc}) \wedge$
.17 $\mathit{env\text{-}l\text{-}h} = \mathsf{nil} \wedge$
.18 $\mathit{typeinst\text{-}h} = \mathsf{nil} \wedge$
.19 $\mathit{os\text{-}h} = \mathsf{nil} \wedge$
.20 $\mathit{cur\text{-}mod\text{-}obj\text{-}l\text{-}h} = \mathsf{nil}\ ) \wedge$
.21 $(\mathit{type} = \mathrm{FNOP} \Rightarrow$
.22 $\mathit{arg\text{-}l} \neq \mathsf{nil} \wedge$
.23 $\mathit{evalstate} = \mathsf{nil} \wedge$
.24 $\mathit{env\text{-}l\text{-}h} \neq \mathsf{nil} \wedge$
.25 $\mathit{typeinst\text{-}h} \neq \mathsf{nil} \wedge$
.26 $\mathit{os\text{-}h} \neq \mathsf{nil} \wedge$
.27 $\mathit{cur\text{-}mod\text{-}obj\text{-}l\text{-}h} \neq \mathsf{nil}\ );$

332.0 $Code = SEM`ExplFN \mid SEM`ImplFN \mid$
.1 $SEM`OP \mid DebugCmd;$

333.0 $DebugCmd :: instr : SubProgram;$

1.10.5 The Evaluation State

The state of the stack machine when the *EvalMainLoop* is completing can either be a breakpoint, an interrupt, a success (completed execution), (in VDM++) either a guard state (meaning that it must be checked whether the permission guard is satisfied) or the maximum number of instructions for the given thread has been reached.

334.0 $EvaluationState = Breakpoint \mid Interrupt \mid Success \mid$
.1 $Guard \mid EndOfSlice;$

335.0 $Breakpoint :: \;\; ;$

336.0 $Success :: \;\; ;$

337.0 $Interrupt :: \;\; ;$

338.0 $EndOfSlice :: \;\; ;$

339.0 $Guard :: opnm : AS`Name$
.1 $curobj : [SEM`OBJ\text{-}Ref];$

340.0 $DebugFlag = Step \mid StepIn \mid SingleStep \mid Finish \mid Continue;$

341.0 $Step :: level : \mathbb{N};$

342.0 $StepIn :: \;\; ;$

343.0 $SingleStep :: level : \mathbb{N};$

344.0 $Finish :: level : \mathbb{N};$

345.0 $Continue ::$

Additional environment operations.
operations

346.0 $GetEnvLLengths : () \xrightarrow{o} \mathbb{N} \times \mathbb{N}$
.1 $GetEnvLLengths\,() \triangleq$
.2 if len $threadstate.env\text{-}l = 0$
.3 then return mk- $(0, 0)$

.4 $\quad$ else return mk- $(\mathsf{len}\ threadstate.env\text{-}l, \mathsf{len}\ \mathsf{hd}\ threadstate.env\text{-}l)$;

347.0 $\quad UpgradeENVL : \mathbb{N} \times \mathbb{N} \xrightarrow{o} ()$

.1 $\quad UpgradeENVL\,(lenvl, ltopenvl) \triangleq$

.2 $\quad$ let $l = \mathsf{len}\ threadstate.env\text{-}l$ in

.3 $\quad$ if $l > 0$

.4 $\quad$ then $(threadstate.env\text{-}l := threadstate.env\text{-}l\,(l + 1 - lenvl, \ldots, l);$

.5 $\quad threadstate.env\text{-}l(1) := threadstate.env\text{-}l\,(1)$

.6 $\quad (\mathsf{len}\ threadstate.env\text{-}l\,(1) + 1 - ltopenvl,$

.7 $\quad \ldots, \mathsf{len}\ threadstate.env\text{-}l\,(1)))$

1.10.6 The Trap Stack

In order to handle different kinds of exception handling statements a stack with the different trap handlers currently in scope is maintained. When an exit is returned the top-most trap-handler must be considered first.

types

348.0 $\quad TrapStack = Trap^*;$

349.0 $\quad Trap :: handid : \mathbb{N}$

.1 $\quad lenes : \mathbb{N}$

.2 $\quad lencs : \mathbb{N}$

.3 $\quad lenobjl : \mathbb{N}$

.4 $\quad cid : CI`ContextId$

.5 $\quad lenenvl : \mathbb{N}$

.6 $\quad lentopenvl : \mathbb{N};$

1.10.7 The Stack Machine State

******** The following has been updated on 2/5/2000 by Jesper K. Pedersen ********

When interpreting concurrent specifications it will be necessary to "swap" processes in and out of the main evaluation loop to model multitasking of concurrent processes.

The state being swapped in and out is defined in the type *EvaluatorStatus*.

A number of fields in the EvaluatorStatus record contains the extension `_bak`. These fields are used to contain part of another stack (the one named the same but without this extension). For more information about this, please see the section on the Up/Down commands (sec. 1.10.13).

The description of each of these fields are:

eval_stack This is the evaluator stack. The stackinterpreter will use this stack to contain intermediated value when interpreting. See section 1.10.1 for more information.

call_stack This stack is used to control recursive functions, recursive debugs, up/down commands and displaying of the back trace stack. See section 1.10.4 for more information.

curCid This is the latest context id passed in program evaluation. This is used when showing the location in the program in case of either breakpoints or runtime errors.

trap_stack This stack is used to handle exceptions. See section 1.10.6 for more information.

PC This is the current program counter.

debug_flag This flag is used to signal the state of debugging to the stack interpreter.

upDnIndex This is an index into the call stack used when going up and down on the back trace stack. See section 1.10.13 for more information.

instrno A counter used to limited the number of instructions a thread can be executed for before it is descheduled.

dur A counter used to record the nesting level of duration statements. If this counter is nil, then time is incremented normally. If it is not nil, the value given by the outermost duration statement is used to increment the time.

release_time The time at which a thread was released. This is used to limit the time a thread is allowed to execute for, if time sliced scheduling is being used.

cf This is a stack used solely when debugging the code generated by the stack code compiler. Instructions which pushes the name of a compilation function on and off this stack is compiled into the program. The actual instructions used for this is `ISTART` and `IEND`.

env_l This is the environment which contains the variables of the program.

typeinst This is a stack of type instantiation mappings used for polymorhic functions which are applied after they have been instantiated.

os This is an environment which contains a copy of the environment at the time an operation is invoked. This is necessary to be able to refer to old values in the post conditions.

obj_l This is a stack of objects which are in scope of the current evaluation.

******** End of updated documentation ****************

350.0 $\mathit{EvaluatorStatus} :: \mathit{eval\text{-}stack} : \mathit{EvalStack}$
.1 $\mathit{call\text{-}stack} : \mathit{CallStack}$
.2 $\mathit{curCid} : \mathit{CI}`\mathit{ContextId}$
.3 $\mathit{trap\text{-}stack} : \mathit{TrapStack}$
.4 $\mathit{PC} : \mathit{ProgramCounter}$
.5 $\mathit{debug\text{-}flag} : \mathit{DebugFlag}$
.6 $\mathit{upDnIndex} : \mathbb{N}$
.7 $\mathit{instrno} : \mathbb{N}$
.8 $\mathit{dur} : \mathbb{N}$
.9 $\mathit{release\text{-}time} : [\mathbb{N}]$
.10 $\mathit{cf} : (\mathsf{char}^* \times \mathit{CI}`\mathit{ContextId})^*$
.11 $\mathit{env\text{-}l} : \mathit{SEM}`\mathit{ENVL}$
.12 $\mathit{env\text{-}l\text{-}bak} : \mathit{SEM}`\mathit{ENVL}$
.13 $\mathit{typeinst} : \mathit{AS}`\mathit{TypeVar} \xrightarrow{m} \mathit{AS}`\mathit{Type}^*$
.14 $\mathit{typeinst\text{-}bak} : \mathit{AS}`\mathit{TypeVar} \xrightarrow{m} \mathit{AS}`\mathit{Type}^*$
.15 $\mathit{os} : \mathit{GLOBAL}`\mathit{OBJ\text{-}tab}^*$
.16 $\mathit{os\text{-}bak} : \mathit{GLOBAL}`\mathit{OBJ\text{-}tab}^*$
.17 $\mathit{obj\text{-}l} : \mathit{GLOBAL}`\mathit{OBJscope}^*$
.18 $\mathit{obj\text{-}l\text{-}bak} : \mathit{GLOBAL}`\mathit{OBJscope}^*$

351.0 state sigma of
.1 $\mathit{threadstate} : \mathit{EvaluatorStatus}$
.2 $\mathit{curr\text{-}program} : \mathit{SubProgram}$
.3 $\mathit{BREAK} : \mathbb{B}$
.4 $\mathit{lastres} : \mathit{SEM}`\mathit{VAL}$
.5 $\mathit{guard} : [\mathit{Guard}]$
.6 $\mathit{dur} : \mathbb{N}$
.7 $\mathit{time} : \mathbb{N}$
.8 init $s \triangleq s = \mathit{GetInitSigma}\,()$
.9 end

functions

352.0 $\mathit{GetInitSigma} : () \to \mathit{sigma}$
.1 $\mathit{GetInitSigma}\,() \triangleq$
.2 mk-$\mathit{sigma}\,(\mathit{EvaluatorStatusInit}\,(),$
.3 $[\,]$, false, mk-$\mathit{SEM}`\mathit{UNDEF}\,()$,
.4 nil ,
.5 $0, 0);$

Even though redundant, the state also contains the instruction sequence currently being evaluated. This is simply for clarity and efficiency of the main evaluation loop.

The field *BREAK* is modified by the instructions *INSTRTP‘CONTEXT* and *INSTRTP‘POPCONTEXT* and read by the operation *EvalMainLoop*. It is used to signal whether the evaluation should be suspended due to a break point.

1.10.8 Initializartion of the Evaluator Status

353.0 $\mathit{EvaluatorStatusInit} : () \to \mathit{EvaluatorStatus}$

.1 $\mathit{EvaluatorStatusInit}\,() \triangleq$

.2 $\quad \textsf{mk-}\mathit{EvaluatorStatus}\,([\,],$

.3 $\qquad [\,],$

.4 $\qquad \mathit{CI}`\mathit{NilContextId},$

.5 $\qquad [\,],$

.6 $\qquad 0,$

.7 $\qquad \textsf{mk-}\mathit{Continue}\,(),$

.8 $\qquad 0,$

.9 $\qquad 0,$

.10 $\qquad 0,$

.11 $\qquad \textsf{nil}\ ,$

.12 $\qquad [\,],$

.13 $\qquad \mathit{envl\text{-}init}\,(),$

.14 $\qquad [\,],$

.15 $\qquad \mathit{typeinst\text{-}init}\,(),$

.16 $\qquad [\,],$

.17 $\qquad [\,],$

.18 $\qquad [\,],$

.19 $\qquad [\,],$

.20 $\qquad [\,])$

1.10.9 Instantiating and Storing the State of the Evaluator

The following operations *Instantiate* and *Persist* are used to set, respectively read the state of the evaluator. A scheduling mechanism to facilitate the evaluation of concurrent processes should use these operations to swap processes in and out of the main evaluation loop. The scheduler will naturally have to maintain a queue of "sleeping" processes and their state as reported by *Persist*.

Note that *Instantiate* can also replace the current program. This is used with queueing threads which are being instantiated.

operations

```
354.0 Instantiate : EvaluatorStatus × [SubProgram] →o ()
  .1  Instantiate (es2, instr) △
  .2    (threadstate := es2;
  .3     curr-program := if instr ≠ nil
  .4                     then instr
  .5                     elseif threadstate.call-stack ≠ []
  .6                     then ExtractInstr (HeadCS ().code)
  .7                     else [];
  .8     if instr ≠ nil
  .9     then threadstate.PC := 0;
  .10    dur := es2.dur);

355.0 GetEvaluatorState : () →o EvaluatorStatus
  .1  GetEvaluatorState () △
  .2    return threadstate;

356.0 InitEvaluatorStatus : [SubProgram] × [SEM`OBJ-Ref] →o EvaluatorStatus
  .1  InitEvaluatorStatus (instr, objref) △
  .2    (dcl e : EvaluatorStatus := EvaluatorStatusInit ();
  .3     if objref ≠ nil
  .4     then (e.obj-l := [mk-GLOBAL`OBJscope (objref,
  .5                                           [STATE`GetNameOfObjRef (objref)],
  .6                                           [STATE`GetNameOfObjRef (objref)])]);
  .7     if instr ≠ nil
  .8     then (let code : Code = mk-DebugCmd (instr ⁀ [mk-INSTRTP`EOCL ()]) in
  .9            (e.call-stack := [mk-CallStackItem (INTERNAL, code, 0, "ThreadStart",
  .10                                              nil , nil , CI`NilContextId, nil , nil , nil , nil )];
  .11            e.debug-flag := mk-Continue ()));
  .12    return e );

357.0 Init : () →o ()
  .1  Init () △
  .2    (threadstate := EvaluatorStatusInit ());

358.0 ExtractInstr : Code →o SubProgram
  .1  ExtractInstr (i) △
  .2    if is-DebugCmd (i)
  .3    then return i.instr
  .4    else return CMPL`GetProgram (i.modName, i.instr) ;
```

359.0 $stackeval\text{-}Init : () \xrightarrow{o} ()$

.1 $stackeval\text{-}Init\,() \triangleq$

.2 $\quad (sigma := GetInitSigma\,();$

.3 $\quad\; ResetEnvL()\,);$

360.0 $User\text{-}Init : AS`Document \times \mathbb{B} \xrightarrow{o} ()$

.1 $User\text{-}Init\,(ast, ast\text{-}is\text{-}new) \triangleq$

.2 $\quad (stackeval\text{-}Init()\,;$

.3 $\quad\; SCHD`InitScheduler()\,;$

.4 $\quad\; STATE`Init\text{-}Sigma(ast\text{-}is\text{-}new)\,;$

.5 $\quad\; STATE`TranslateAST\,(ast, ast\text{-}is\text{-}new)\,;$

.6 $\quad\; STATE`InitializeGSGV\,(ast\text{-}is\text{-}new)\,);$

User_Init

User_Init also pushes a module on the stack to have a current module. If a document is only a single definitions block, this ensures that we are always evaluating in the created module.

361.0 $envl\text{-}init : () \xrightarrow{o} SEM`ENV^*$

.1 $envl\text{-}init\,() \triangleq$

.2 $\quad$ return $[[]];$

This operation creates an evaluation stack, with an initially empty function application environment.

362.0 $ResetEnvL : () \xrightarrow{o} ()$

.1 $ResetEnvL\,() \triangleq$

.2 $\quad threadstate.env\text{-}l := envl\text{-}init\,();$

363.0 $ResetTypeInst : () \xrightarrow{o} ()$

.1 $ResetTypeInst\,() \triangleq$

.2 $\quad threadstate.typeinst := typeinst\text{-}init\,();$

364.0 $typeinst\text{-}init : () \xrightarrow{o} AS`TypeVar \xrightarrow{m} AS`Type^*$

.1 $typeinst\text{-}init\,() \triangleq$

.2 $\quad$ return $[\{\mapsto\}];$

This operation creates an empty type variable map in the type instantiation sequence.

1.10.10 Auxiliary operations on OS state

365.0 $PushOS : GLOBAL`OBJ\text{-}tab \xrightarrow{o} ()$

.1 $PushOS\,(o) \triangleq$

.2 $\quad threadstate.os := [o] \curvearrowright threadstate.os;$

366.0 $PopOS : () \xrightarrow{o} ()$

.1 $PopOS\,() \triangleq$

.2 $\quad threadstate.os := \mathsf{tl}\ threadstate.os;$

367.0 $PushCurObjTab2OS : () \xrightarrow{o} ()$

.1 $PushCurObjTab2OS\,() \triangleq$

.2 $\quad threadstate.os := [STATE`Get\text{-}obj\text{-}tab\,()] \curvearrowright threadstate.os;$

1.10.11 Auxiliary operations on typeinst state

368.0 $PushTypeInst : AS`TypeVar \xrightarrow{m} AS`Type \xrightarrow{o} ()$

.1 $PushTypeInst\,(tm) \triangleq$

.2 $\quad threadstate.typeinst := [tm] \curvearrowright threadstate.typeinst;$

369.0 $PopTypeInst : () \xrightarrow{o} ()$

.1 $PopTypeInst\,() \triangleq$

.2 $\quad threadstate.typeinst := \mathsf{tl}\ threadstate.typeinst;$

370.0 $HdTypeInst : () \xrightarrow{o} AS`TypeVar \xrightarrow{m} AS`Type$

.1 $HdTypeInst\,() \triangleq$

.2 $\quad \mathsf{return}\ \mathsf{hd}\ threadstate.typeinst;$

1.10.12 Environments

371.0 $PopEnvL : () \xrightarrow{o} ()$

.1 $PopEnvL\,() \triangleq$

.2 $\quad (threadstate.env\text{-}l := \mathsf{tl}\ threadstate.env\text{-}l);$

The operation *PopEnvL* removes the current application environment from the evaluation stack. The top element of the temporay object reference stack is also removed, and the object references bound to the environment is deleted.

372.0 $TopEnvL : () \xrightarrow{o} SEM`ENV$
.1 $TopEnvL\,() \triangleq$
.2 return hd $threadstate.env\text{-}l$;

This operation returns the current function application environment. The evaluation stack is not altered.

373.0 $PushEmptyBlkEnv : SEM`Permission \xrightarrow{o} ()$
.1 $PushEmptyBlkEnv\,(permis) \triangleq$
.2 $(threadstate.env\text{-}l(1) := [\mathsf{mk}\text{-}SEM`BlkEnv\,(\{\mapsto\}, permis)] \curvearrowright threadstate.env\text{-}l\,(1))$
.3 pre $threadstate.env\text{-}l \neq []$;

374.0 $PushEmptyEnv : () \xrightarrow{o} ()$
.1 $PushEmptyEnv\,() \triangleq$
.2 $(threadstate.env\text{-}l := [[]] \curvearrowright threadstate.env\text{-}l)$;

This operation creates a new and empty function application environment. This environment is made the current evaluation environment.

375.0 $IsEmptyEnvL : () \xrightarrow{o} \mathbb{B}$
.1 $IsEmptyEnvL\,() \triangleq$
.2 return (len $TopEnvL\,() = 0$);

This operation returns true if the current function application environment is empty, i.e. the current evaluation environment does not contain any block environments.

376.0 $AddToTopBlkEnv : SEM`BlkEnv \xrightarrow{o} ()$
.1 $AddToTopBlkEnv\,(blkenv) \triangleq$
.2 $(threadstate.env\text{-}l(1)(1) := AUX`CombineBlkEnv\,(threadstate.env\text{-}l\,(1)\,(1), blkenv))$;

The operation *GetObjLLen* is used for the test environment to check for memory leaks.

377.0 $GetObjLLen : () \xrightarrow{o} \mathbb{N}$
.1 $GetObjLLen\,() \triangleq$
.2 (return len $threadstate.obj\text{-}l$);

The operation *GetCurObjRef* returns the current object refence of the top of the object list *obj-l*.

378.0 *GetCurObjRef* : () $\xrightarrow{o}$ *SEM‘OBJ-Ref*
.1 *GetCurObjRef* () $\triangleq$
.2 let mk-*GLOBAL‘OBJscope* (*ref*, -, -) = hd *threadstate.obj-l* in
.3 return *ref*
.4 pre *threadstate.obj-l* $\neq$ [] ;

The operation *GetCurObjName* returns the name of the current object on the top of the object list *obj-l*.

379.0 *GetCurObjName* : () $\xrightarrow{o}$ *AS‘Name*
.1 *GetCurObjName* () $\triangleq$
.2 let mk-*GLOBAL‘OBJscope* (*ref*, -, -) = hd *threadstate.obj-l* in
.3 let mk-*GLOBAL‘OBJ-Desc* (-, mk-*SEM‘OBJ* (*nm*, -, -), -, -) = *STATE‘Lookup-obj-tab* (*ref*) in
.4 return *nm*
.5 pre *threadstate.obj-l* $\neq$ [] ;

The operation *GetCurCl* returns the current class of the scope, that is, the class being on top of the class stack of the object list *obj-l*

380.0 *GetCurCl* : () $\xrightarrow{o}$ *GLOBAL‘OrigCl*
.1 *GetCurCl* () $\triangleq$
.2 if *threadstate.obj-l* = []
.3 then return mk-*GLOBAL‘Start* ()
.4 else let mk-*GLOBAL‘OBJscope* (-, *cl*, -) = hd *threadstate.obj-l* in
.5 return hd *cl* ;

381.0 *GetOrigCl* : () $\xrightarrow{o}$ *GLOBAL‘OrigCl*
.1 *GetOrigCl* () $\triangleq$
.2 if *threadstate.obj-l* = []
.3 then return mk-*GLOBAL‘Start* ()
.4 else let mk-*GLOBAL‘OBJscope* (-, -, *orig-cll*) = hd *threadstate.obj-l* in
.5 return hd *orig-cll*
.6 pre *threadstate.obj-l* $\neq$ [] ;

The `GetOrigOldCl` operation is needed when new expressions are used to extract the class for the object before the current origin for object was pushed.

382.0 $GetOrigOldCl : () \xrightarrow{o} GLOBAL`OrigCl$

.1 $GetOrigOldCl\,() \triangleq$

.2 let mk-$GLOBAL`OBJscope\,(\text{-}, \text{-}, orig\text{-}cll) =$ hd $threadstate.obj\text{-}l$ in

.3 if len $orig\text{-}cll > 1$

.4 then return hd tl $orig\text{-}cll$

.5 else return hd $orig\text{-}cll$

.6 pre $threadstate.obj\text{-}l \neq []$;

383.0 $HasCurCl : () \xrightarrow{o} \mathbb{B}$

.1 $HasCurCl\,() \triangleq$

.2 return $threadstate.obj\text{-}l \neq []$;

The operation *GetCurObj* returns the semantic value *SEM‘OBJ* of the current of object, that is, the object being on top of the object stack *obj-l*.

384.0 $GetCurObj : () \xrightarrow{o} SEM`OBJ$

.1 $GetCurObj\,() \triangleq$

.2 let mk-$GLOBAL`OBJscope\,(ref, \text{-}, \text{-}) =$ hd $threadstate.obj\text{-}l$ in

.3 return $STATE`GetSemObjInTab\,(ref)$

.4 pre $threadstate.obj\text{-}l \neq []$;

The operation *IsEmptyObjL* checks if the *obj_l* is empty. This could be the case when evaluating from the debugger.

385.0 $IsEmptyObjL : () \xrightarrow{o} \mathbb{B}$

.1 $IsEmptyObjL\,() \triangleq$

.2 return $threadstate.obj\text{-}l = []$;

The operation *PushCurObj* pushes an object reference on the current object stack. The operation takes three parameters:

- the object reference.
- the name of the class in the object reference that we are to look from.
- the original class from which we call. Only in the case where a client application is done, the clnm and the origcl is different. Consider the example below:

```
class A

instance variables

private
```

```
    myvar : int := 3;

operations
public
  op1: () ==> ()
  op1() ==
    myvar := 4;



end A

class B

operations

  Test: () ==> ()
  Test() ==
   (dcl a: A := new A()
    a.op1();                        - the op1 is now a client to object a.
   )

end B
```

When calling the operation *a.op1()* operation *op1()* it should be checked that the modifier on operation *op1* is public. When evaluating the *FieldSelectExpr* "a.op1". The scope that is setting up is to push the object reference of *a*. The current class will be *A*, however, the orignal class will be *B*. Because the access check on op1, should be checked with the assumption that a.op1 is a client call from class B. When the semantic value of *op1* has been found, and when we are to set up the environment for evaluating the body of *op1*, the object scope would be pushing object reference of *a*, the current class will be *A* and the orignal class will also be *A*, because now the code that is to evaluated inside op1 is **not** client to class *A*.

NOTE: HC: I think that we should not allow this use of nil. I suggest that the AS'Name are not optional.

386.0 $PushCurObj : SEM`OBJ\text{-}Ref \times [AS`Name] \times [GLOBAL`OrigCl] \xrightarrow{o} ()$

.1 $PushCurObj\,(objref, clnm, origcl) \triangleq$

.2 $\quad$ let $nmobjref = STATE`GetNameOfObjRef\,(objref),$

.3 $\qquad nm =$ if $clnm =$ nil

.4 $\qquad\qquad$ then $nmobjref$

.5 $\qquad\qquad$ else $clnm,$

.6 $\qquad orignm =$ if $origcl =$ nil

.7 $\qquad\qquad\quad$ then $nmobjref$

.8 $\qquad\qquad\quad$ else $origcl$ in

.9 $\quad (threadstate.obj\text{-}l := [\text{mk-}GLOBAL`OBJscope\,(objref, [nm], [orignm])] \curvearrowright threadstate.obj\text{-}l);$

The operation *PopCurObj* pops an object from the object stack *obj-l*.

387.0 $PopCurObj : () \xrightarrow{o} ()$

.1 $PopCurObj\ () \triangleq$

.2 $(threadstate.obj\text{-}l := \mathsf{tl}\ threadstate.obj\text{-}l);$

The operation *PushClNmCurObj* pushes a class name *clnm* on the object stack *obj-l*

388.0 $PushClNmCurObj : AS`Name \times AS`Name \xrightarrow{o} ()$

.1 $PushClNmCurObj\ (clnm, origcl) \triangleq$

.2 $\mathsf{let\ mk\text{-}}GLOBAL`OBJscope\ (obj, cl\text{-}l, orig\text{-}cll) = \mathsf{hd}\ threadstate.obj\text{-}l\ \mathsf{in}$

.3 $threadstate.obj\text{-}l := [\mathsf{mk\text{-}}GLOBAL`OBJscope\ (obj,$

.4 $[clnm] \curvearrowright cl\text{-}l,$

.5 $[origcl] \curvearrowright orig\text{-}cll)] \curvearrowright$

.6 $\mathsf{tl}\ threadstate.obj\text{-}l$

.7 $\mathsf{pre}\ \ threadstate.obj\text{-}l \neq []\ ;$

The operation *PopClNmCurObj* pops a class from the object stack.

389.0 $PopClNmCurObj : () \xrightarrow{o} ()$

.1 $PopClNmCurObj\ () \triangleq$

.2 $\mathsf{let\ mk\text{-}}GLOBAL`OBJscope\ (obj, cl\text{-}l, orig\text{-}cll) = \mathsf{hd}\ threadstate.obj\text{-}l\ \mathsf{in}$

.3 $threadstate.obj\text{-}l := [\mathsf{mk\text{-}}GLOBAL`OBJscope\ (obj, \mathsf{tl}\ cl\text{-}l, \mathsf{tl}\ orig\text{-}cll)] \curvearrowright \mathsf{tl}\ threadstate.obj\text{-}l;$

390.0 $PopBlkEnv : () \xrightarrow{o} ()$

.1 $PopBlkEnv\ () \triangleq$

.2 $(threadstate.env\text{-}l := [\mathsf{tl}\ \mathsf{hd}\ threadstate.env\text{-}l] \curvearrowright \mathsf{tl}\ threadstate.env\text{-}l);$

This operation removes the current block environment from the current function application environment.

The corresponding block environment in the temporary object reference stack is removed, and the object references bound in the block environment is deleted.

391.0 $PushBlkEnv : SEM`BlkEnv \xrightarrow{o} ()$

.1 $PushBlkEnv\ (env) \triangleq$

.2 $(threadstate.env\text{-}l := [[env] \curvearrowright \mathsf{hd}\ threadstate.env\text{-}l] \curvearrowright \mathsf{tl}\ threadstate.env\text{-}l);$

This operation makes the argument block environment the current block environment in the current function application environment.

A corresponding block environment is pushed on the temporary object reference stack, and the object references in the environment *env* are incremented.

392.0 $\mathit{AppendToTopBlkEnv} : \mathit{AS}`\mathit{Name} \times \mathit{SEM}`\mathit{VAL} \times [\mathit{AS}`\mathit{Type}] \xrightarrow{o} ()$
.1 $\mathit{AppendToTopBlkEnv}\,(\mathit{id}, \mathit{val}, \mathit{tp}) \triangleq$
.2 $\quad(\mathsf{let}\ \mathit{blkenv} = (\mathsf{hd}\ \mathsf{hd}\ \mathit{threadstate.env\text{-}l}),$
.3 $\qquad \mathit{id\text{-}m} = \{\mathit{id} \mapsto \mathsf{mk\text{-}}\mathit{SEM}`\mathit{ValTp}\,(\mathit{val}, \mathit{tp})\}\ \mathsf{in}$
.4 $\quad(\mathsf{let}\ \mathit{new\text{-}blkenv} = \mathsf{mk\text{-}}\mathit{SEM}`\mathit{BlkEnv}\,(\mathit{blkenv.id\text{-}m} \dagger \mathit{id\text{-}m}, \mathit{blkenv.perm})\ \mathsf{in}$
.5 $\quad\ \mathit{threadstate.env\text{-}l} := [[\mathit{new\text{-}blkenv}] \curvearrowright \mathsf{tl}\ \mathsf{hd}\ \mathit{threadstate.env\text{-}l}] \curvearrowright$
.6 $\qquad\qquad\qquad \mathsf{tl}\ \mathit{threadstate.env\text{-}l}));$

This operation will append an identifier, a semantic value and possible a type to the top BlkEnv.

393.0 $\mathit{TopBlkEnv} : () \xrightarrow{o} \mathit{SEM}`\mathit{BlkEnv}$
.1 $\mathit{TopBlkEnv}\,() \triangleq$
.2 $\quad\mathsf{return}\ \mathsf{hd}\ \mathsf{hd}\ \mathit{threadstate.env\text{-}l};$

This operation returns the current block environment from the current function application environment.

394.0 $\mathit{MakeNewObj} : \mathit{SEM}`\mathit{OBJ} \xrightarrow{o} \mathit{SEM}`\mathit{OBJ\text{-}Ref}$
.1 $\mathit{MakeNewObj}\,(\mathit{semobj}) \triangleq$
.2 $\quad(\mathsf{let}\ \mathit{ref} = \mathit{STATE}`\mathit{global\text{-}obj\text{-}tab\text{-}insert}\,(\mathit{semobj})\ \mathsf{in}$
.3 $\quad\ \mathsf{return}\ \mathit{ref}\ );$

The operation *MakeNewObj* creates an object in the *obj-tab*, and puts the temporary object reference in the object reference environment.

1.10.13 The Up/Down functions

******** The following has been updated on 2/5/2000 by Jesper K. Pedersen ********

Whenever the user issues an up-command a number of the stacks located in the EvaluatorStatus (sec. 1.10.7) is cut in two. This is done so the stacks will contain the content as was the case when the function which scope we go into was the active one. In other words, the stacks will look as they did when this function called its sub-function. The rest of the content of this stack is kept on a backup stack.

This way the rest of the functions in the toolbox do not need to know anything about the up and down commands. As an example, the lookup functions work on the top element of the environment list (`env_l`) without knowing that this element might stem from a function, which is currently stalled to let a sub-function evaluate.

The backup-stacks are copied into the `CallStackItem` along with the environment list and copied back upon completion of the recursive debug command or print command. Furthermore the backup-stacks are emptied when a recursive debug or print command is issued.

If the backup stack was not emptied when the new recursive debug command was issued, then the following scenario may have occured:

1. The user executes the `up` command a number of times (say through the functions f,g and h).
2. Then he invokes a recursive debug command.
3. This command breaks at a breakpoint.
4. the user executes the command `down`. This results in that first the function `f` is brought to scope. On further invocation of the `down` command `g` and finally `h` will brought to scope. None of these functions is an extension to the recursive debug command, but rather to the previous debug command.

When one of continue, finish, step, stepin, or single-step commands is issued the elements from the backup stacks are moved back to the original stacks.

******** End of updated documentation ****************

395.0 $GoUp : () \xrightarrow{o} ()$
.1 $GoUp\,() \triangleq$
.2 let $levels = GetNextStackLevelsUp\,()$ in
.3 if $levels =$ nil
.4 then error
.5 else $UpdateStacks(levels)$;

396.0 $GoDown : () \xrightarrow{o} ()$
.1 $GoDown\,() \triangleq$
.2 if $threadstate.upDnIndex = 0$
.3 then error
.4 elseif $threadstate.upDnIndex = 1$
.5 then $ResetUpDn()$
.6 else let $levels = GetNextStackLevelsDown\,()$ in
.7 if $levels =$ nil
.8 then error
.9 else $UpdateStacks(levels)$;

397.0 $UpdateStacks : (\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}) \xrightarrow{o} ()$
.1 $UpdateStacks\,(\textsf{mk-}\,(env\text{-}l\text{-}h, typeinst\text{-}h, os\text{-}h, cur\text{-}mod\text{-}obj\text{-}l\text{-}h)) \triangleq$
.2 $(\textsf{let }\ env\text{-}l = threadstate.env\text{-}l\text{-}bak \curvearrowright threadstate.env\text{-}l \textsf{ in}$
.3 $(threadstate.env\text{-}l := env\text{-}l\,(\textsf{len } env\text{-}l - env\text{-}l\text{-}h + 1, \ldots, \textsf{len } env\text{-}l);$
.4 $threadstate.env\text{-}l\text{-}bak := env\text{-}l\,(1, \ldots, \textsf{len } env\text{-}l - env\text{-}l\text{-}h));$
.5 $\textsf{let }\ typeinst = threadstate.typeinst\text{-}bak \curvearrowright threadstate.typeinst \textsf{ in}$
.6 $(threadstate.typeinst := typeinst\,(\textsf{len } typeinst - typeinst\text{-}h + 1, \ldots, \textsf{len } typeinst);$
.7 $threadstate.typeinst\text{-}bak := typeinst\,(1, \ldots, \textsf{len } typeinst - typeinst\text{-}h));$
.8 $\textsf{let }\ os = threadstate.os\text{-}bak \curvearrowright threadstate.os \textsf{ in}$
.9 $(threadstate.os := os\,(\textsf{len } os - os\text{-}h + 1, \ldots, \textsf{len } os);$
.10 $threadstate.os\text{-}bak := os\,(1, \ldots, \textsf{len } os - os\text{-}h));$
.11 $\textsf{let }\ obj\text{-}l = threadstate.obj\text{-}l\text{-}bak \curvearrowright threadstate.obj\text{-}l \textsf{ in}$
.12 $(threadstate.obj\text{-}l := obj\text{-}l\,(\textsf{len } obj\text{-}l - cur\text{-}mod\text{-}obj\text{-}l\text{-}h + 1, \ldots, \textsf{len } obj\text{-}l);$
.13 $threadstate.obj\text{-}l\text{-}bak := obj\text{-}l\,(1, \ldots, \textsf{len } obj\text{-}l - cur\text{-}mod\text{-}obj\text{-}l\text{-}h)));$

398.0 $ResetUpDn : () \xrightarrow{o} ()$
.1 $ResetUpDn\,() \triangleq$
.2 $(UpdateStacks(\textsf{mk-}\,(\textsf{len } threadstate.env\text{-}l + \textsf{len } threadstate.env\text{-}l\text{-}bak,$
.3 $\textsf{len } threadstate.typeinst + \textsf{len } threadstate.typeinst\text{-}bak,$
.4 $\textsf{len } threadstate.os + \textsf{len } threadstate.os\text{-}bak,$
.5 $\textsf{len } threadstate.obj\text{-}l + \textsf{len } threadstate.obj\text{-}l\text{-}bak))\ ;$
.6 $threadstate.upDnIndex := 0);$

399.0 $GetNextStackLevelsUp : () \xrightarrow{o} [\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}]$
.1 $GetNextStackLevelsUp\,() \triangleq$
.2 $(\textsf{dcl }\ index : \mathbb{N} := threadstate.upDnIndex,$
.3 $indexFound : \mathbb{N} := -1;$
.4 $\textsf{while }\ index < \textsf{len } threadstate.call\text{-}stack$
.5 $\textsf{do }\ (index := index + 1;$
.6 $\textsf{if }\ threadstate.call\text{-}stack\,(index).type = \mathrm{DEBUGCMD}$
.7 $\textsf{then return nil}$
.8 $\textsf{elseif }\ threadstate.call\text{-}stack\,(index).type = \mathrm{FNOP}$
.9 $\textsf{then if }\ indexFound = -1$
.10 $\textsf{then }\ indexFound := index$
.11 $\textsf{else }\ (threadstate.upDnIndex := indexFound;$
.12 $\textsf{return mk-}\,(threadstate.call\text{-}stack\,(indexFound).env\text{-}l\text{-}h,$
.13 $threadstate.call\text{-}stack\,(indexFound).typeinst\text{-}h,$
.14 $threadstate.call\text{-}stack\,(indexFound).os\text{-}h,$
.15 $threadstate.call\text{-}stack\,(indexFound).cur\text{-}mod\text{-}obj\text{-}l\text{-}h)\ ));$
.16 $\textsf{return nil}\ \ );$

400.0 $GetNextStackLevelsDown : () \xrightarrow{o} [\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}]$
.1 $GetNextStackLevelsDown\,() \triangleq$
.2 $(\textsf{dcl }\ index : \mathbb{N} := threadstate.upDnIndex;$

.3 while $index > 1$
.4 do ($index := index - 1$;
.5 if $threadstate.call\text{-}stack\,(index).type = \mathrm{FNOP}$
.6 then ($threadstate.upDnIndex := index$;
.7 return mk- ($threadstate.call\text{-}stack\,(index).env\text{-}l\text{-}h$,
.8 $threadstate.call\text{-}stack\,(index).typeinst\text{-}h$,
.9 $threadstate.call\text{-}stack\,(index).os\text{-}h$,
.10 $threadstate.call\text{-}stack\,(index).cur\text{-}mod\text{-}obj\text{-}l\text{-}h$)));
.11 return nil);

401.0 $CurrentBacktraceLevel : () \xrightarrow{o} \mathbb{N}$
.1 $CurrentBacktraceLevel\,() \triangleq$
.2 return len $threadstate.env\text{-}l\text{-}bak + 1$;

402.0 $GetCidForCurBacktraceLevel : () \xrightarrow{o} CI`ContextId$
.1 $GetCidForCurBacktraceLevel\,() \triangleq$
.2 let $level =$ len $threadstate.env\text{-}l\text{-}bak$ in
.3 if $level = 0$
.4 then return $GetCurCid\,()$
.5 else let $cids = [threadstate.call\text{-}stack\,(callStkIndex).oldCid \mid$
.6 $callStkIndex \in$ inds $threadstate.call\text{-}stack \cdot$
.7 $threadstate.call\text{-}stack\,(callStkIndex).type = \mathrm{FNOP}]$ in
.8 return $cids\,(level)$;

1.10.14 Module Operations

The operation *PushModule* makes the module with name *mod-name* the current module.

1.10.15 Local States

1.10.16 Different State Manipulations

403.0 $IsLocalState : AS`Name \xrightarrow{o} \mathbb{B} \times [GLOBAL`State]$
.1 $IsLocalState\,(id) \triangleq$
.2 (let $env = TopEnvL\,()$ in
.3 for $blkenv$ in env
.4 do (if $blkenv.perm = \mathrm{READ\_WRITE} \land$
.5 $id \in$ dom $blkenv.id\text{-}m$
.6 then let mk-$SEM`ValTp\,(val, tp) = blkenv.id\text{-}m\,(id)$ in
.7 return mk- (true, mk-$GLOBAL`State\,(val, tp)$)

```
        else skip);
    return mk- (false, nil ) );
```

This operation returns true if the input identifier is defined in the domain of the current local state.

```
404.0 SetLocalState : AS`Name × GLOBAL`State o→ ()
SetLocalState (id, s-val) △
  (let env = hd threadstate.env-l in
   (dcl index1 : ℕ := 0,
        returnflag : 𝔹 := false;
    while ¬ returnflag
    do (index1 := index1 + 1;
        let blkenv = env (index1) in
        if blkenv.perm = READ_WRITE
        then if id ∈ dom blkenv.id-m
             then (threadstate.env-l(1)(index1).id-m(id) := mk-SEM`ValTp (s-val.val, s-val.tp);
                   returnflag := true))))
pre let mk- (isit, -) = IsLocalState (id) in
    isit ;
```

This operation updates the identifier in the local state (a block environment with read/write permission) The reason that I use returnflag instead of just making a return in the line `returnflag = true` is because of a bug, where an empty return statement inside of a loop doesn't return.

```
405.0 IsLocalVal : AS`Name o→ 𝔹 × [SEM`VAL]
IsLocalVal (name) △
  (let topenv = TopEnvL () in
   for blkenv in topenv
   do (if name ∈ dom blkenv.id-m
       then let mk-SEM`ValTp (val′, -) = blkenv.id-m (name) in
            return mk- (true, val′)
       else skip);
   return mk- (false, nil ) );
```

This operation returns true if the input identifier is defined in the current function application environment.

```
406.0 EvalOldName : AS`OldName o→ SEM`VAL
EvalOldName (mk-AS`OldName (name, cid)) △
  let orig-name = mk-AS`Name (name, cid),
      old-obj-tab = hd threadstate.os,
```

.4 $\qquad \textsf{mk-}\mathit{GLOBAL`OBJscope}\,(\mathit{ref}, \text{-}, \text{-}) = \textsf{hd}\ \mathit{threadstate.obj\text{-}l}\ \textsf{in}$

.5 $\quad \textsf{if}\ \mathit{ref} \in \textsf{dom}\ \mathit{old\text{-}obj\text{-}tab}$

.6 $\quad \textsf{then let}\ \mathit{old\text{-}obj\text{-}sem} = \mathit{old\text{-}obj\text{-}tab}\,(\mathit{ref}).\mathit{sem}\ \textsf{in}$

.7 $\qquad \textsf{let mk-}\,(\mathit{isit}, \text{-}, \mathit{val}, \text{-}, \text{-}, \mathit{access}) = \mathit{STATE`IsInObjScope}\,(\mathit{orig\text{-}name}, \mathit{old\text{-}obj\text{-}sem})\ \textsf{in}$

.8 $\qquad \textsf{if}\ \mathit{isit}$

.9 $\qquad \textsf{then return}\ \mathit{val}$

.10 $\qquad \textsf{else}\ \mathit{AUX`ErrorOp}(\text{"}\mathit{Unknownoldname} * \mathit{name} * \text{"})$

.11 $\quad \textsf{else}\ \mathit{AUX`ErrorOp}(\text{"}\mathit{InternalError}\text{"})\,;$

1.10.17 Stack Operations

The following operations all modify or read the state of the different stacks of the evaluator. That is the evaluation stack, the call stack, the context stack and the trap stack.

The CF stack

407.0 $\mathit{PrintCf} : () \xrightarrow{o} (\textsf{char}^* \times \textsf{char}^* \times \mathbb{N} \times \mathbb{N})^*$

.1 $\mathit{PrintCf}\,() \triangleq$

.2 $\quad \textsf{return}\ [\textsf{let mk-}\,(\mathit{name}, \mathit{cid}) = \mathit{threadstate.cf}\,(i),$

.3 $\qquad\qquad \textsf{mk-}\,(\mathit{file}, \mathit{line}, \mathit{column}) = \mathit{CI`GetFileLineColPos}\,(\mathit{cid})\ \textsf{in}$

.4 $\qquad\quad \textsf{mk-}\,(\mathit{name}, \mathit{file}, \mathit{line}, \mathit{column}) \mid$

.5 $\qquad\qquad\quad i \in \textsf{inds}\ \mathit{threadstate.cf}];$

The Evaluation Stack

The evaluation stack is modified during evaluation. The operation *Pop* pops a variable number of items of the evaluation stack and returns a sequence of the popped items. Items are popped one at a time, top-down, and *prepended* to the resulting sequence, therefore the sequence returned by *Pop* will be ordered with the topmost item at the rightmost position.

408.0 $\mathit{Pop} : \mathbb{N} \xrightarrow{o} \mathit{EvalStackItem}^*$

.1 $\mathit{Pop}\,(n) \triangleq$

.2 $\quad (\textsf{dcl}\ \mathit{arg\text{-}l} : \mathit{EvalStackItem}^* := [];$

.3 $\quad\ \textsf{while len}\ \mathit{arg\text{-}l} < n$

.4 $\quad\ \textsf{do}\ (\mathit{arg\text{-}l} := [\textsf{hd}\ \mathit{threadstate.eval\text{-}stack}] \curvearrowright \mathit{arg\text{-}l};$

.5 $\qquad\quad \mathit{threadstate.eval\text{-}stack} := \textsf{tl}\ \mathit{threadstate.eval\text{-}stack});$

.6 $\quad\ \textsf{return}\ \mathit{arg\text{-}l}\ )$

.7 $\textsf{pre}\ \ \textsf{len}\ \mathit{threadstate.eval\text{-}stack} \geq n\ ;$

409.0 $\mathit{Push} : \mathit{EvalStackItem} \xrightarrow{o} ()$
.1 $\mathit{Push}\,(e) \triangleq$
.2 $\quad \mathit{threadstate.eval\text{-}stack} := [e] \curvearrowright \mathit{threadstate.eval\text{-}stack};$

The operation *Head* returns the topmost item of the evaluation stack, but leaves the stack un-modified.

410.0 $\mathit{Head} : () \xrightarrow{o} \mathit{EvalStackItem}$
.1 $\mathit{Head}\,() \triangleq$
.2 $\quad \textsf{return hd } \mathit{threadstate.eval\text{-}stack};$

The operation *GetES* returns the n first elements of the evaluation stack. Note that the order used here is reversed compared to the order from the *Pop* operation.

411.0 $\mathit{GetES} : \mathbb{N} \xrightarrow{o} \mathit{EvalStackItem}^*$
.1 $\mathit{GetES}\,(n) \triangleq$
.2 $\quad \textsf{return } [\mathit{threadstate.eval\text{-}stack}\,(i) \mid i \in \{1, \ldots, n\}]$
.3 $\textsf{pre } \ \mathit{LenES}\,(n) \ ;$

412.0 $\mathit{LenES} : \mathbb{N} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{LenES}\,(n) \triangleq$
.2 $\quad \textsf{return len } \mathit{threadstate.eval\text{-}stack} \geq n;$

The Call Stack

413.0 $\mathit{PushCS} : \mathit{Code} \times (\mathit{AS}`\mathit{Name} \mid \textsf{char}^*) \times [\mathit{SEM}`\mathit{VAL}^*] \times \mathit{CSItemType} \xrightarrow{o} ()$
.1 $\mathit{PushCS}\,(\mathit{item}, \mathit{nm}, \mathit{val\text{-}l}, \mathit{type}) \triangleq$
.2 $\quad \textsf{let } \mathit{cid} = \mathit{GetCurCid}\,(),$
.3 $\qquad \mathit{env\text{-}l\text{-}h} = \textsf{len } \mathit{threadstate.env\text{-}l},$
.4 $\qquad \mathit{typeinst\text{-}h} = \textsf{len } \mathit{threadstate.typeinst},$
.5 $\qquad \mathit{os\text{-}h} = \textsf{len } \mathit{threadstate.os},$
.6 $\qquad \mathit{cur\text{-}mod\text{-}obj\text{-}l\text{-}h} = \textsf{len } \mathit{threadstate.obj\text{-}l} \textsf{ in}$
.7 $\quad (\mathit{threadstate.call\text{-}stack} := [\textsf{mk-}\mathit{CallStackItem}\,(\mathit{type}, \mathit{item}, \mathit{threadstate.PC}, \mathit{nm}, \mathit{val\text{-}l}, \textsf{nil}\ , \mathit{cid},$
.8 $\qquad \mathit{env\text{-}l\text{-}h}, \mathit{typeinst\text{-}h}, \mathit{os\text{-}h}, \mathit{cur\text{-}mod\text{-}obj\text{-}l\text{-}h})] \curvearrowright$
.9 $\qquad \mathit{threadstate.call\text{-}stack};$
.10 $\quad \mathit{threadstate.PC} := 0;$
.11 $\quad \mathit{curr\text{-}program} := \mathit{ExtractInstr}\,(\mathit{item}));$

414.0 $PopCS : () \xrightarrow{o} ()$

.1 $PopCS\,() \triangleq$

.2 let mk-$CallStackItem\,(type, \text{-}, PC', \text{-}, \text{-}, evalState, cid, \text{-}, \text{-}, \text{-}, \text{-}) =$ hd $threadstate.call\text{-}stack$ in

.3 $(threadstate.PC := PC';$

.4 if $(type = \mathrm{DEBUGCMD})$

.5 then $(threadstate.env\text{-}l\text{-}bak := evalState.env\text{-}l\text{-}bak;$

.6 $threadstate.typeinst\text{-}bak := evalState.typeinst\text{-}bak;$

.7 $threadstate.os\text{-}bak := evalState.os\text{-}bak;$

.8 $threadstate.curCid := cid;$

.9 $threadstate.obj\text{-}l\text{-}bak := evalState.obj\text{-}l\text{-}bak);$

.10 $threadstate.call\text{-}stack :=$ tl $threadstate.call\text{-}stack;$

.11 if len $threadstate.call\text{-}stack = 0$

.12 then $curr\text{-}program := []$

.13 else let $code = ($hd $threadstate.call\text{-}stack).code$ in

.14 $curr\text{-}program := ExtractInstr\,(code))$

.15 pre $threadstate.call\text{-}stack \neq []$;

415.0 $HeadCS : () \xrightarrow{o} CallStackItem$

.1 $HeadCS\,() \triangleq$

.2 return hd $threadstate.call\text{-}stack$

.3 pre $threadstate.call\text{-}stack \neq []$;

416.0 $CallStackLevel : () \xrightarrow{o} \mathbb{N}$

.1 $CallStackLevel\,() \triangleq$

.2 return len $threadstate.call\text{-}stack;$

417.0 $PushDS : STKM`EvaluatorStatus \times$ char$^* \times Code \xrightarrow{o} ()$

.1 $PushDS\,(evalst, debugString, code) \triangleq$

.2 $(threadstate.call\text{-}stack := [$mk-$CallStackItem\,(\mathrm{DEBUGCMD}, code, threadstate.PC, debugString,$ nil ,

.3 $evalst, GetCurCid\,(),$ nil , nil , nil , nil $)] \curvearrowright$

.4 $threadstate.call\text{-}stack;$

.5 $threadstate.PC := 0;$

.6 $curr\text{-}program := ExtractInstr\,(code);$

.7 $threadstate.env\text{-}l\text{-}bak := [];$

.8 $threadstate.typeinst\text{-}bak := [];$

.9 $threadstate.os\text{-}bak := [];$

.10 $threadstate.obj\text{-}l\text{-}bak := []);$

```
418.0 get-objref-from-fnop : SEM`ExplOP | SEM`ExplFN -o-> [SEM`OBJ-Ref]
   .1 get-objref-from-fnop (fnop) ≜
   .2   if is-SEM`ExplOP (fnop)
   .3   then return fnop.objref
   .4   elseif is-SEM`ExplFN (fnop)
   .5   then return fnop.objref
   .6   else error;
```

```
419.0 ReplaceEvaluatorStatus : EvaluatorStatus -o-> ()
   .1 ReplaceEvaluatorStatus (e) ≜
   .2   (Instantiate(e, nil ) );
```

```
420.0 UserPopDS : () -o-> ()
   .1 UserPopDS () ≜
   .2   if ¬ ∃ index ∈ inds threadstate.call-stack ·
   .3          threadstate.call-stack (index).type = DEBUGCMD
   .4   then error
   .5   else (dcl index : ℕ := 1,
   .6             more : 𝔹 := true;
   .7         DEBUG`ResetInActivity() ;
   .8         while more
   .9         do (let mk-CallStackItem (type, code, -, nmOrDesc, -, evalst, cid, -, -, -, -) = hd threadstate.call-stack in
  .10             (if type = DEBUGCMD
  .11              then (ReplaceEvaluatorStatus(evalst) ;
  .12                    more := false);
  .13              if type = FNOP ∧ is-SEM`ExplOP (code)
  .14              then (STATE`UpdateHistCount(nmOrDesc, mk-INSTRTP`fin (), get-objref-from-fnop (code));
  .15                    threadstate.curCid := cid));
  .16             index := index + 1));
```

```
421.0 IsProgramAtEnd : () -o-> 𝔹
   .1 IsProgramAtEnd () ≜
   .2   return threadstate.PC ≥ len curr-program;
```

The Context Stack

```
422.0 SetCid : CI`ContextId -o-> ()
   .1 SetCid (cid) ≜
   .2   threadstate.curCid := cid;
```

423.0 $GetCurCid : () \xrightarrow{o} CI`ContextId$
.1 $GetCurCid\,() \triangleq$
.2 return $threadstate.curCid$;

1.10.18 The Trap Stack

The trap state must store information about the sizes of the different stacks and the environment lists. When one wish to “goto” a trap handler these states must be restored to the sizes at the time of pushing the trap handler information.

424.0 $PushTS : \mathbb{N} \xrightarrow{o} ()$
.1 $PushTS\,(handid) \triangleq$
.2 let $les =$ len $threadstate.eval\text{-}stack$,
.3 $lcs =$ len $threadstate.call\text{-}stack$,
.4 $lobjl =$ len $threadstate.obj\text{-}l$,
.5 $cid = threadstate.curCid$,
.6 mk- $(lenvl, ltopenvl) = GetEnvLLengths\,()$ in
.7 $threadstate.trap\text{-}stack :=$ [mk-$Trap\,(handid, les, lcs,$
.8 $lobjl,$
.9 $cid, lenvl, ltopenvl)] \curvearrowright$
.10 $threadstate.trap\text{-}stack$;

425.0 $PopTS : () \xrightarrow{o} ()$
.1 $PopTS\,() \triangleq$
.2 $threadstate.trap\text{-}stack :=$ tl $threadstate.trap\text{-}stack$;

426.0 $EmptyTS : () \xrightarrow{o} \mathbb{B}$
.1 $EmptyTS\,() \triangleq$
.2 return $threadstate.trap\text{-}stack = []$;

If there is no trap handler the current operation should be finished (and cleaned up. This is done using the *ExeRETURN* operation. Otherwise we must find the program counter for the exception handler code and restore the different stacks to the state they were in when the handler was pushed.

427.0 $GotoTrapHandler : () \xrightarrow{o} ()$
.1 $GotoTrapHandler\,() \triangleq$
.2 if $EmptyTS\,()$
.3 then $INSTR`ExeRETURN()$

```
else let mk-Trap (handid, les, lcs,
                  lobjl,
                  cid, lenvl, ltopenvl) =
             hd threadstate.trap-stack in
     (threadstate.PC := FindTrapHandler (handid, lcs);
      threadstate.eval-stack := [hd threadstate.eval-stack] ⁀
                                        threadstate.eval-stack (len threadstate.eval-stack+1−
les,
                                                       ..., len threadstate.eval-stack);
      UpdateHistoryCounters(len threadstate.call-stack − lcs) ;
      threadstate.obj-l := threadstate.obj-l (len threadstate.obj-l + 1 − lobjl,
                                               ..., len threadstate.obj-l);
      threadstate.call-stack := threadstate.call-stack (len threadstate.call-stack+1−
lcs,
                                                      ..., len threadstate.call-stack);
      threadstate.curCid := cid;
      UpgradeENVL(lenvl, ltopenvl) );
```

```
428.0 FindTrapHandler : ℕ × ℕ -o-> ProgramCounter
      FindTrapHandler (handid, lengthcallstack) △
        (if len threadstate.call-stack = lengthcallstack
         then for i = 1 to len curr-program
              do if is-INSTRTP`HANDID (curr-program (i)) ∧
                    handid = curr-program (i).handid
                 then return i
                 else skip
         else let instr = ExtractInstr (threadstate.call-stack (len threadstate.call-stack+1−
                                                     lengthcallstack).code) in
              (curr-program := instr;
               for i = 1 to len instr
               do if is-INSTRTP`HANDID (instr (i)) ∧
                     handid = instr (i).handid
                  then return i
                  else skip);
         error);
```

```
429.0 UpdateHistoryCounters : ℕ -o-> ()
      UpdateHistoryCounters (count) △
        for index = 1 to count
        do let mk-CallStackItem (-, code, -, nm, -, -, -, -, -, -, -) = threadstate.call-stack (index) in
           if is-AS`Name (nm) ∧ is-SEM`ExplOP (code)
           then STATE`UpdateHistCount(nm, mk-INSTRTP`fin (), get-objref-from-fnop (code));
```

1.10.19 Various Operations to Modify the State

The operation *IncrPC* increments/decrements the PC with n.

430.0 $IncrPC : \mathbb{Z} \xrightarrow{o} ()$
.1 $IncrPC\,(n) \triangleq$
.2 $\quad threadstate.PC := threadstate.PC + n;$

The following operations are used to set the state of the debugging environment.

The operations *SetStep*, *SetStepIn*, *SetSingleStep*, *SetFinish*, and *SetContinue* should be called by the debugger to set the state of the evaluator.

431.0 $SetStep : () \xrightarrow{o} ()$
.1 $SetStep\,() \triangleq$
.2 $\quad threadstate.debug\text{-}flag := \mathsf{mk}\text{-}Step\,(CallStackLevel\,());$

432.0 $SetStepIn : () \xrightarrow{o} ()$
.1 $SetStepIn\,() \triangleq$
.2 $\quad threadstate.debug\text{-}flag := \mathsf{mk}\text{-}StepIn\,();$

433.0 $SetSingleStep : () \xrightarrow{o} ()$
.1 $SetSingleStep\,() \triangleq$
.2 $\quad threadstate.debug\text{-}flag := \mathsf{mk}\text{-}SingleStep\,(CallStackLevel\,());$

434.0 $SetFinish : () \xrightarrow{o} ()$
.1 $SetFinish\,() \triangleq$
.2 $\quad threadstate.debug\text{-}flag := \mathsf{mk}\text{-}Finish\,(CallStackLevel\,());$

435.0 $SetContinue : () \xrightarrow{o} ()$
.1 $SetContinue\,() \triangleq$
.2 $\quad threadstate.debug\text{-}flag := \mathsf{mk}\text{-}Continue\,();$

436.0 $GetDebugFlag : () \xrightarrow{o} DebugFlag$
.1 $GetDebugFlag\,() \triangleq$
.2 $\quad \mathsf{return}\ threadstate.debug\text{-}flag;$

437.0 $\mathit{SetDebugFlag} : \mathit{DebugFlag} \xrightarrow{o} ()$

.1 $\mathit{SetDebugFlag}\,(\mathit{new\text{-}df}) \triangleq$

.2 $\quad \mathit{threadstate.debug\text{-}flag} := \mathit{new\text{-}df};$

The *SetBREAK* operation is used by the *INSTR‘CONTEXT* and *INSTR‘POPCONTEXT* to signal if the main evaluation loop in *EvalMainLoop* should be suspended.

438.0 $\mathit{SetBREAK} : \mathbb{B} \xrightarrow{o} ()$

.1 $\mathit{SetBREAK}\,(b) \triangleq$

.2 $\quad \mathit{BREAK} := b;$

439.0 $\mathit{GetBREAK} : () \xrightarrow{o} \mathbb{B}$

.1 $\mathit{GetBREAK}\,() \triangleq$

.2 $\quad \mathsf{return}\ \mathit{BREAK};$

1.10.20 Setting the guard evaluator state

440.0 $\mathit{SetGuard} : [\mathit{AS‘Name}] \times [\mathit{SEM‘OBJ\text{-}Ref}] \xrightarrow{o} ()$

.1 $\mathit{SetGuard}\,(\mathit{fullopnm}, \mathit{obj}) \triangleq$

.2 $\quad \mathit{guard} := \mathsf{mk\text{-}}\mathit{Guard}\,(\mathit{fullopnm}, \mathit{obj});$

441.0 $\mathit{ResetGuard} : () \xrightarrow{o} ()$

.1 $\mathit{ResetGuard}\,() \triangleq$

.2 $\quad \mathit{guard} := \mathsf{nil}\ ;$

1.10.21 Scheduling Related Issues

442.0 $\mathit{ResetSlice} : () \xrightarrow{o} ()$

.1 $\mathit{ResetSlice}\,() \triangleq$

.2 $\quad \mathit{threadstate} := \mathit{SCHD‘InitSlice}\,(\mathit{threadstate});$

1.10.22 The Main loop

EvalMainLoop is the main operation used to evaluate an instruction sequence. The operation returns when the evaluation of the instruction sequence (the sub-program) is terminated or suspended.

Termination: a program is terminated if the evaluation reaches the end of the program or if a run time error occurs during evaluation.

Suspended: a program is suspended if the evaluation meets a break point or if the evaluation is suspended because a *StepIn*, *Step* or *Single Step* was completed.

The status of the evaluator after *EvalMainLoop* has returned can be read by calling the *MainLoopState* operation.

The *EvalMainLoop* operation assumes that the initialisation of the specification which has been read in has been made. In addition the object for which the execution should take place must have been pushed in the *STATE* module and the instruction code for the execution to be carried out must be installed inside the stack machine.

```
443.0  EvalMainLoop : () -o-> STKM`EvaluationState × [SEM`VAL]
       EvalMainLoop () △
         (SetBREAK(false) ;
          if (guard ≠ nil )
          then RTERR`Error(RTERR`INTERNAL-ERROR, nil , nil , []) ;
          while threadstate.PC < len curr-program ∧ ¬ BREAK ∧
                guard = nil ∧ ¬ SCHD`Deschedule (threadstate)
          do (threadstate.PC := threadstate.PC + 1;
              threadstate := SCHD`IncInstrnum (threadstate);
              EvalInstr(curr-program (threadstate.PC)) );
          let eval-state = MainLoopState () in
          if is-Success (eval-state)
          then return mk- (eval-state, hd Pop (1))
          else return mk- (eval-state, nil ) );
```

```
444.0  EvalUninterruptedLoop : () -o-> STKM`EvaluationState × SEM`VAL
       EvalUninterruptedLoop () △
         (while threadstate.PC < len curr-program
          do (threadstate.PC := threadstate.PC + 1;
              EvalInstr(curr-program (threadstate.PC)) );
          return mk- (mk-Success (), hd Pop (1)) );
```

445.0 $MainLoopState : () \xrightarrow{o} EvaluationState$

.1 $MainLoopState\,() \triangleq$
.2 (if $guard \neq$ nil
.3 then return $guard$;
.4 if $GetBREAK\,()$
.5 then return mk-$Breakpoint\,()$;
.6 if $threadstate.PC \geq$ len $curr\text{-}program \wedge$ len $curr\text{-}program \neq 0$
.7 then $(lastres := Head\,()$;
.8 return mk-$Success\,()$);
.9 if $SCHD`EndOfSliceReached\,(threadstate)$
.10 then return mk-$EndOfSlice\,()$;
.11 return mk-$Interrupt\,()$);

446.0 $IncrRelTime : \mathbb{R} \xrightarrow{o} ()$

.1 $IncrRelTime\,(t) \triangleq$
.2 $IncrAbsTime(SETTINGS`GetTimeFactor\,() \times t)$;

447.0 $IncrAbsTime : \mathbb{R} \xrightarrow{o} ()$

.1 $IncrAbsTime\,(t) \triangleq$
.2 $(time := time + t$;
.3 $RecordTime()$);

448.0 $InDuration : () \xrightarrow{o} \mathbb{B}$

.1 $InDuration\,() \triangleq$
.2 return $dur > 0$;

449.0 $RecordTime : () \xrightarrow{o} ()$

.1 $RecordTime\,() \triangleq$
.2 let - $= IO`fwriteval[\mathbb{N}]\,($"$timelog.msg$"$, time,$ APPEND$)$ in
.3 skip;

450.0 $GetTime : () \xrightarrow{o} \mathbb{R}$

.1 $GetTime\,() \triangleq$
.2 return $time$;

451.0 $SetTime : \mathbb{R} \xrightarrow{o} ()$

.1 $SetTime\,(new\text{-}time) \triangleq$
.2 $(time := new\text{-}time$;
.3 $RecordTime()$);

```
452.0 EvalInstr : INSTRTP‘Instruction –o→ ()
   .1 EvalInstr (i) ≜
   .2    cases i:
   .3       mk-INSTRTP‘PUSH (i) → Push(i) ,
   .4       mk-INSTRTP‘POP (n) → let - = Pop (n) in
   .5                                  skip,
   .6       mk-INSTRTP‘COPYVAL () → INSTR‘ExeCOPYVAL() ,
   .7       mk-INSTRTP‘UNOP (op) → INSTR‘EvalUNOP(op) ,
   .8       mk-INSTRTP‘BINOP (op) → INSTR‘EvalBINOP(op) ,
   .9       mk-INSTRTP‘PRE () → INSTR‘ExePRE() ,
  .10       mk-INSTRTP‘POST () → INSTR‘ExePOST() ,
  .11       mk-INSTRTP‘POSTENV (resnmtps, ci) → INSTR‘ExePOSTENV(resnmtps, ci),
  .12       mk-INSTRTP‘DTC (tp) → INSTR‘ExeDTC(tp) ,
  .13       mk-INSTRTP‘DTCSET () → INSTR‘ExeDTCSET() ,
  .14       mk-INSTRTP‘SIZE (n) → INSTR‘ExeSIZE(n) ,
  .15       mk-INSTRTP‘COMMENT (-) → skip,
  .16       mk-INSTRTP‘ISTART (str, cid) →
  .17             (threadstate.cf := [mk- (str, cid)] ⁀ threadstate.cf),
  .18       mk-INSTRTP‘IEND (-) → (threadstate.cf := tl threadstate.cf),
  .19       mk-INSTRTP‘APPENDSEQ () → INSTR‘ExeAPPENDSEQ() ,
  .20       mk-INSTRTP‘APPENDMAP () → INSTR‘ExeAPPENDMAP() ,
  .21       mk-INSTRTP‘ADDSET () → INSTR‘ExeADDSET() ,
  .22       mk-INSTRTP‘SETRNG () → INSTR‘ExeSETRNG() ,
  .23       mk-INSTRTP‘SUBSEQ () → INSTR‘ExeSUBSEQ() ,
  .24       mk-INSTRTP‘SELELEM () → INSTR‘ExeSELELEM() ,
  .25       mk-INSTRTP‘SELSEQELEM (direction) → INSTR‘ExeSELSEQELEM(direction),
  .26       mk-INSTRTP‘APPENDTUP () → INSTR‘ExeAPPENDTUP() ,
  .27       mk-INSTRTP‘NOP () → skip,
  .28       mk-INSTRTP‘APPLY () → INSTR‘ExeAPPLY() ,
  .29       mk-INSTRTP‘RETURN () → INSTR‘ExeRETURN() ,
  .30       mk-INSTRTP‘ISCHECK (ty) → INSTR‘ExeISCHECK(ty) ,
  .31       mk-INSTRTP‘EMPTYLIST () → INSTR‘ExeEMPTYLIST() ,
  .32       mk-INSTRTP‘ISEMPTYSET (n, tp) → INSTR‘ExeISEMPTYSET(n, tp) ,
  .33       mk-INSTRTP‘ISEMPTYSEQ (n, tp) → INSTR‘ExeISEMPTYSEQ(n, tp) ,
  .34       mk-INSTRTP‘APPENDESTCK () → INSTR‘ExeAPPENDESTCK() ,
  .35       mk-INSTRTP‘LOOKUP (nm) → INSTR‘ExeLOOKUP(nm) ,
  .36       mk-INSTRTP‘LOOKUPSTATIC (nm) → INSTR‘ExeLOOKUPSTATIC(nm) ,
  .37       mk-INSTRTP‘CBR (n) → INSTR‘ExeCBR(n) ,
  .38       mk-INSTRTP‘CNBR (n) → INSTR‘ExeCNBR(n) ,
  .39       mk-INSTRTP‘BR (n) → INSTR‘ExeBR(n) ,
  .40       mk-INSTRTP‘CONTEXT (cid, isStmt) → INSTR‘ExeCONTEXT(cid, isStmt),
  .41       mk-INSTRTP‘POPBLKENV () → INSTR‘ExePOPBLKENV() ,
  .42       mk-INSTRTP‘EOCL () → INSTR‘ExeEOCL() ,
  .43       mk-INSTRTP‘MULTBINDL (n, part) → INSTR‘ExeMULTBINDL(n, part) ,
  .44       mk-INSTRTP‘SELBLKENV (n) → INSTR‘ExeSELBLKENV(n) ,
  .45       mk-INSTRTP‘APPENDBLKENV (id, tp) → INSTR‘ExeAPPENDBLKENV(id, tp),
  .46       mk-INSTRTP‘REMSTACKELEM (n) → INSTR‘ExeREMSTACKELEM(n) ,
  .47       mk-INSTRTP‘SWAP () → INSTR‘ExeSWAP() ,
```

```
mk-INSTRTP`EMPTYBLKENV (permis) → INSTR`ExeEmptyBlkEnv(permis),
mk-INSTRTP`MATCHANDBIND () → INSTR`ExeMatchAndBind() ,
mk-INSTRTP`MATCHVAL () → INSTR`ExeMATCHVAL() ,
mk-INSTRTP`TRYMATCH () → INSTR`ExeTRYMATCH() ,
mk-INSTRTP`TRYANYMATCH () → INSTR`ExeTRYANYMATCH() ,
mk-INSTRTP`CLOSENV (blkenv, b-m) → INSTR`ExeCLOSENV(blkenv, b-m),
mk-INSTRTP`ADDTOBLKENV () → INSTR`ExeADDTOBLKENV() ,
mk-INSTRTP`UPDATECLOSENV (ex) → INSTR`ExeUPDATECLOSENV(ex),
mk-INSTRTP`ERRINST (err) → INSTR`ExeERRINST(err) ,
mk-INSTRTP`ENUMAPPEND () → INSTR`ExeEnumAppend() ,
mk-INSTRTP`SETUNION (cid) → INSTR`ExeSetUnion(cid) ,
mk-INSTRTP`SEQCONC (cid) → INSTR`ExeSeqConc(cid) ,
mk-INSTRTP`FREF (cid) → INSTR`ExeFREF(cid) ,
mk-INSTRTP`MOSREF (cid) → INSTR`ExeMOSREF(cid) ,
mk-INSTRTP`ASSIGNSD () → INSTR`ExeASSIGNSD() ,
mk-INSTRTP`FIELDSAPPEND () → INSTR`ExeFieldsAppend() ,
mk-INSTRTP`RECCONS (tag, length) → INSTR`ExeRECCONS(tag, length) ,
mk-INSTRTP`RECMOD (length) → INSTR`ExeRECMOD(length) ,
mk-INSTRTP`FIELDSEL () → INSTR`ExeFIELDSEL() ,
mk-INSTRTP`TOKENVAL () → INSTR`ExeTOKENVAL() ,
mk-INSTRTP`SEQCOMPBIND () → INSTR`ExeSEQCOMPBIND() ,
mk-INSTRTP`SEQELEMMATCH (index) → INSTR`ExeSEQELEMMATCH(index),
mk-INSTRTP`SEQMAPOVER () → INSTR`ExeSEQMAPOVER() ,
mk-INSTRTP`TUPSEL (index) → INSTR`ExeTUPSEL(index) ,
mk-INSTRTP`TYPEJUDGE (tp) → INSTR`ExeTYPEJUDGE(tp) ,
mk-INSTRTP`POLYTYPEINST (nm) → INSTR`ExePOLYTYPEINST(nm) ,
mk-INSTRTP`POLYINST (inst) → INSTR`ExePOLYINST(inst) ,
mk-INSTRTP`MKEXIT (isnil) → INSTR`ExeMKEXIT(isnil) ,
mk-INSTRTP`EXITVAL () → INSTR`ExeEXITVAL() ,
mk-INSTRTP`ISCONT () → INSTR`ExeISCONT() ,
mk-INSTRTP`ISEXIT () → INSTR`ExeISEXIT() ,
mk-INSTRTP`ISNEEXIT () → INSTR`ExeISNEEXIT() ,
mk-INSTRTP`REMEXITVAL () → INSTR`ExeREMEXITVAL() ,
mk-INSTRTP`PUSHTH (n) → INSTR`ExePUSHTH(n) ,
mk-INSTRTP`POPTH () → INSTR`ExePOPTH() ,
mk-INSTRTP`HANDID (-) → skip,
mk-INSTRTP`VERIFYINDEXARGS () → INSTR`ExeVERIFYINDEXARGS(),
mk-INSTRTP`TESTCOUNTER () → INSTR`ExeTESTCOUNTER() ,
mk-INSTRTP`INCRCOUNTER () → INSTR`ExeINCRCOUNTER() ,
mk-INSTRTP`NONDETSTMT () → INSTR`ExeNONDETSTMT() ,
mk-INSTRTP`NOBODY (err, modname, name, parms) →
    INSTR`ExeNOBODY(err, modname, name, parms) ,
mk-INSTRTP`RANDOM () → INSTR`ExeRANDOM() ,
mk-INSTRTP`ATOMIC (no) → INSTR`ExeATOMIC(no) ,
mk-INSTRTP`LASTRES () → Push(lastres) ,
mk-INSTRTP`NEWOBJ (nm, dlobject) →
    (if ¬ STATE`IsClassInit (nm)
     then STATE`InitClassName(nm) ;
     INSTR`ExeNEWOBJ(nm, dlobject) ),
mk-INSTRTP`NEWPOSABSOBJ (nm) → INSTR`ExeNEWPOSABSOBJ(nm, nil ),
```

```
    mk-INSTRTP`INITCLASS (nm, initno) → INSTR`ExeINITCLASS(nm, initno),
    mk-INSTRTP`BINDINSTVAR (nm) → INSTR`ExeBINDINSTVAR(nm) ,
    mk-INSTRTP`NEWCOMPL (checkinv) → INSTR`ExeNEWCOMPL(checkinv),
    mk-INSTRTP`PUSHCLNMCUROBJ (cl, origcl) → INSTR`ExePUSHCLNMCUROBJ(cl, origcl),
    mk-INSTRTP`POPCLNMCUROBJ () → INSTR`ExePOPCLNMCUROBJ() ,
    mk-INSTRTP`SELFEXPR () → INSTR`ExeSELFEXPR() ,
    mk-INSTRTP`ISOFCLASS (clnm) → INSTR`ExeISOFCLASS(clnm) ,
    mk-INSTRTP`ISOFBASECLASS (clnm) → INSTR`ExeISOFBASECLASS(clnm),
    mk-INSTRTP`SAMEBASECLASS () → INSTR`ExeSAMEBASECLASS() ,
    mk-INSTRTP`SAMECLASS () → INSTR`ExeSAMECLASS() ,
    mk-INSTRTP`CALLGUARD (hasobj, oprt) → INSTR`ExeCALLGUARD(hasobj, oprt),
    mk-INSTRTP`PPCALL () → INSTR`ExePPCALL() ,
    mk-INSTRTP`DLCALL (clName, opNm) → INSTR`ExeDLCALL(clName, opNm),
    mk-INSTRTP`HISTORY (kind, opnm) → INSTR`ExeHISTORY(kind, opnm) ,
    mk-INSTRTP`STARTLIST (no) → INSTR`ExeSTARTLIST(no) ,
    mk-INSTRTP`GUARD () → INSTR`ExeGUARD() ,
    mk-INSTRTP`THREADID () → INSTR`ExeTHREADID() ,
    mk-INSTRTP`INCRTIME (i) → INSTR`ExeINCRTIME(i) ,
    mk-INSTRTP`RUNTIME-INCRTIME-PREF (opr, oh) →
         INSTR`ExeINCRTIME-PREF(opr, oh) ,
    mk-INSTRTP`RUNTIME-INCRTIME-BIN (opr, oh1, oh2) →
         INSTR`ExeINCRTIME-BIN(opr, oh1, oh2) ,
    mk-INSTRTP`RUNTIME-INCRTIME-SETSEQMAP (oh) →
         INSTR`ExeINCRTIME-SETSEQMAP(oh) ,
    mk-INSTRTP`RUNTIME-INCRTIME-NEW (oh, nm) →
         INSTR`ExeINCRTIME-NEW(oh, nm) ,
    mk-INSTRTP`RUNTIME-INCRTIME-STARTLIST (oh) →
         INSTR`ExeINCRTIME-STARTLIST(oh) ,
    mk-INSTRTP`PUSHDURATION () → dur := dur + 1,
    mk-INSTRTP`POPDURATION (t) → (dur := dur − 1;
                                  if ¬ InDuration ()
                                  then IncrAbsTime(t) ),
    others → RTERR`Error(RTERR`INTERNAL-ERROR, nil , nil , [])
  end

end STKM
```

Test Suite : rtinfo.ast
Module : STKM

Name	#Calls	Coverage
STKM`Pop	**undefined**	**undefined**
STKM`GoUp	**undefined**	**undefined**
STKM`Head	**undefined**	**undefined**
STKM`Init	**undefined**	**undefined**
STKM`IsSD	**undefined**	**undefined**
STKM`Push	**undefined**	**undefined**
STKM`GetES	**undefined**	**undefined**
STKM`IsPat	**undefined**	**undefined**

Name	#Calls	Coverage
STKM‘LenES	**undefined**	**undefined**
STKM‘PopCS	**undefined**	**undefined**
STKM‘PopOS	**undefined**	**undefined**
STKM‘PopTS	**undefined**	**undefined**
STKM‘GoDown	**undefined**	**undefined**
STKM‘HeadCS	**undefined**	**undefined**
STKM‘IncrPC	**undefined**	**undefined**
STKM‘PushCS	**undefined**	**undefined**
STKM‘PushDS	**undefined**	**undefined**
STKM‘PushOS	**undefined**	**undefined**
STKM‘PushTS	**undefined**	**undefined**
STKM‘SetCid	**undefined**	**undefined**
STKM‘EmptyTS	**undefined**	**undefined**
STKM‘GetTime	**undefined**	**undefined**
STKM‘PopEnvL	**undefined**	**undefined**
STKM‘PrintCf	**undefined**	**undefined**
STKM‘SetStep	**undefined**	**undefined**
STKM‘SetTime	**undefined**	**undefined**
STKM‘TopEnvL	**undefined**	**undefined**
STKM‘GetBREAK	**undefined**	**undefined**
STKM‘GetCurCl	**undefined**	**undefined**
STKM‘HasCurCl	**undefined**	**undefined**
STKM‘SetBREAK	**undefined**	**undefined**
STKM‘SetGuard	**undefined**	**undefined**
STKM‘EvalInstr	**undefined**	**undefined**
STKM‘GetCurCid	**undefined**	**undefined**
STKM‘GetCurObj	**undefined**	**undefined**
STKM‘GetOrigCl	**undefined**	**undefined**
STKM‘PopBlkEnv	**undefined**	**undefined**
STKM‘PopCurObj	**undefined**	**undefined**
STKM‘ResetEnvL	**undefined**	**undefined**
STKM‘ResetUpDn	**undefined**	**undefined**
STKM‘SetFinish	**undefined**	**undefined**
STKM‘SetStepIn	**undefined**	**undefined**
STKM‘TopBlkEnv	**undefined**	**undefined**
STKM‘UserPopDS	**undefined**	**undefined**
STKM‘User-Init	**undefined**	**undefined**
STKM‘envl-init	**undefined**	**undefined**
STKM‘GetObjLLen	**undefined**	**undefined**
STKM‘HdTypeInst	**undefined**	**undefined**
STKM‘InDuration	**undefined**	**undefined**
STKM‘IsLocalVal	**undefined**	**undefined**
STKM‘MakeNewObj	**undefined**	**undefined**
STKM‘PushBlkEnv	**undefined**	**undefined**
STKM‘PushCurObj	**undefined**	**undefined**
STKM‘RecordTime	**undefined**	**undefined**
STKM‘ResetGuard	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
STKM‘ResetSlice	**undefined**	**undefined**
STKM‘EvalOldName	**undefined**	**undefined**
STKM‘IncrAbsTime	**undefined**	**undefined**
STKM‘IncrRelTime	**undefined**	**undefined**
STKM‘Instantiate	**undefined**	**undefined**
STKM‘IsEmptyEnvL	**undefined**	**undefined**
STKM‘IsEmptyObjL	**undefined**	**undefined**
STKM‘PopTypeInst	**undefined**	**undefined**
STKM‘SetContinue	**undefined**	**undefined**
STKM‘UpgradeENVL	**undefined**	**undefined**
STKM‘EvalMainLoop	**undefined**	**undefined**
STKM‘ExtractInstr	**undefined**	**undefined**
STKM‘GetCurObjRef	**undefined**	**undefined**
STKM‘GetDebugFlag	**undefined**	**undefined**
STKM‘GetInitSigma	**undefined**	**undefined**
STKM‘GetOrigOldCl	**undefined**	**undefined**
STKM‘IsLocalState	**undefined**	**undefined**
STKM‘PushEmptyEnv	**undefined**	**undefined**
STKM‘PushTypeInst	**undefined**	**undefined**
STKM‘SetDebugFlag	**undefined**	**undefined**
STKM‘UpdateStacks	**undefined**	**undefined**
STKM‘GetCurObjName	**undefined**	**undefined**
STKM‘MainLoopState	**undefined**	**undefined**
STKM‘PopClNmCurObj	**undefined**	**undefined**
STKM‘ResetTypeInst	**undefined**	**undefined**
STKM‘SetLocalState	**undefined**	**undefined**
STKM‘SetSingleStep	**undefined**	**undefined**
STKM‘typeinst-init	**undefined**	**undefined**
STKM‘AddToTopBlkEnv	**undefined**	**undefined**
STKM‘CallStackLevel	**undefined**	**undefined**
STKM‘GetEnvLLengths	**undefined**	**undefined**
STKM‘IsProgramAtEnd	**undefined**	**undefined**
STKM‘PushClNmCurObj	**undefined**	**undefined**
STKM‘stackeval-Init	**undefined**	**undefined**
STKM‘FindTrapHandler	**undefined**	**undefined**
STKM‘GotoTrapHandler	**undefined**	**undefined**
STKM‘PushEmptyBlkEnv	**undefined**	**undefined**
STKM‘PushCurObjTab2OS	**undefined**	**undefined**
STKM‘AppendToTopBlkEnv	**undefined**	**undefined**
STKM‘GetEvaluatorState	**undefined**	**undefined**
STKM‘EvaluatorStatusInit	**undefined**	**undefined**
STKM‘InitEvaluatorStatus	**undefined**	**undefined**
STKM‘GetNextStackLevelsUp	**undefined**	**undefined**
STKM‘get-objref-from-fnop	**undefined**	**undefined**
STKM‘CurrentBacktraceLevel	**undefined**	**undefined**
STKM‘EvalUninterruptedLoop	**undefined**	**undefined**
STKM‘UpdateHistoryCounters	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
STKM‘GetNextStackLevelsDown	**undefined**	**undefined**
STKM‘ReplaceEvaluatorStatus	**undefined**	**undefined**
STKM‘GetCidForCurBacktraceLevel	**undefined**	**undefined**
Total Coverage		**0%**

1.11 Instruction Set Types

This module contain the types for the instructions. It is located in a separate module for the purpose of code generation.

module *INSTRTP*

imports

from *AS* all ,
from *CI* all ,
from *PAT* all ,
from *REP* all ,
from *SEM* all ,
from *STKM* all ,
from *RTERR* all ,
from *STATE* all ,
from *GLOBAL* all ,
from *SCHDTP* all ,
from *TIMEMAP* all ,
from *TIMEPARSER* all

exports all

definitions
types

465.0 $\mathit{Instruction} = \mathit{CONTEXT} \mid \mathit{NOP} \mid \mathit{CLOSENV} \mid \mathit{UPDATECLOSENV} \mid$
.1 $\mathit{PUSHOS} \mid \mathit{POPOS} \mid$
.2 $\mathit{ExprInstr} \mid$
.3 $\mathit{StmtInstr} \mid$
.4 $\mathit{AuxInstr} \mid$
.5 $\mathit{POPBLKENV} \mid$
.6 $\mathit{RETURN} \mid$
.7 $\mathit{ISCHECK} \mid$
.8 $\mathit{EnvInstr} \mid$
.9 $\mathit{PatInstr} \mid$
.10 $\mathit{VdmPPInstr} \mid$
.11 $\mathit{INCRTIME} \mid$
.12 $\mathit{RUNTIME\text{-}INCRTIME\text{-}PREF} \mid$
.13 $\mathit{RUNTIME\text{-}INCRTIME\text{-}BIN} \mid$
.14 $\mathit{RUNTIME\text{-}INCRTIME\text{-}SETSEQMAP} \mid$
.15 $\mathit{RUNTIME\text{-}INCRTIME\text{-}NEW} \mid$
.16 $\mathit{RUNTIME\text{-}INCRTIME\text{-}STARTLIST} \mid$
.17 $\mathit{PUSHDURATION} \mid$
.18 $\mathit{POPDURATION} \mid$
.19 $\mathit{EvalStackInstr};$

466.0 $\mathit{BINOP} :: \mathit{op} : \mathit{AS}`\mathit{BinaryOp};$

467.0 $\mathit{UNOP} :: \mathit{op} : \mathit{AS}`\mathit{UnaryOp} \mid \mathrm{MAPINVERSE};$

The *CLOSENV* instruction is used to deal with the closure environment of locally defined functions. This is necessary because any free identifiers occuring inside such locally defined functions must know about the value of these from the defining context. These values are not known at compile time so this instruction is used to repair that situation. The *nm* component is the name of the function, but currently it is not used. This is caused by the fact that we believe that the closure environment for the corresponding pre and post-condition functions also should have their closure environments improved with the closure environment.

468.0 $\mathit{CLOSENV} :: \mathit{blkenv} : \mathit{SEM}`\mathit{BlkEnv}$
.1 $\mathit{body\text{-}m} : \mathit{AS}`\mathit{Name} \xrightarrow{m} (\mathit{AS}`\mathit{Expr} \mid \mathrm{NOTYETSPEC});$

469.0 $\mathit{UPDATECLOSENV} :: \mathit{expr} : \mathit{AS}`\mathit{Expr};$

470.0 $EnvInstr = LOOKUP \mid$
.1 $LOOKUPSTATIC \mid$
.2 $PRE \mid$
.3 $POST \mid$
.4 $POSTENV \mid$
.5 $EMPTYENV \mid$
.6 $EMPTYBLKENV \mid$
.7 $MULTBINDL \mid$
.8 $SELBLKENV \mid$
.9 $APPENDBLKENV \mid$
.10 $REMSTACKELEM \mid$
.11 $ADDTOBLKENV \mid$
.12 $SWAP;$

471.0 $PRE :: \ ;$

472.0 $POST :: \ ;$

473.0 $POSTENV :: resnmtps : AS`NameType^*$
.1 $ci : \text{-}CI`ContextId;$

474.0 $PUSHOS :: clmod : AS`Name;$

475.0 $POPOS :: \ ;$

476.0 $LOOKUP :: id : AS`Name \mid AS`OldName;$

477.0 $LOOKUPSTATIC :: id : AS`Name;$

478.0 $EMPTYENV :: \ ;$

479.0 $EMPTYBLKENV :: permis : SEM`Permission;$

480.0 $MULTBINDL :: length : \mathbb{N}$
.1 $part : PAT`PARTITION;$

481.0 $SELBLKENV :: n : \mathbb{N};$

482.0 $APPENDBLKENV :: id : AS`Name$
.1 $tp : AS`Type;$

483.0 $REMSTACKELEM :: n : \mathbb{N};$

484.0 $SWAP :: \ ;$

485.0 $POPBLKENV :: \ ;$

486.0 $ADDTOBLKENV :: \;\; ;$

487.0 $AuxInstr = CBR \mid CNBR \mid BR \mid EOCL \mid ERRINST \mid COPYVAL \mid DTC \mid DTCSET \mid SIZE \mid COMMENT \mid ISTART \mid IEND \mid NOBODY;$

488.0 $CBR :: length : \mathbb{Z};$

489.0 $BR :: length : \mathbb{Z};$

490.0 $CNBR :: length : \mathbb{Z};$

491.0 $ERRINST :: ident : RTERR`ERR;$

492.0 $COPYVAL :: \;\; ;$

493.0 $DTC :: tp : AS`Type;$

494.0 $DTCSET :: \;\; ;$

495.0 $SIZE :: n : \mathbb{N};$

496.0 $COMMENT :: txt : \mathsf{char}^*;$

497.0 $ISTART :: txt : \mathsf{char}^*$
.1 $\quad cid : CI`ContextId;$

498.0 $IEND :: txt : \mathsf{char}^*;$

499.0 $NOBODY :: err : \mathsf{char}^*$
.1 $\quad modname : AS`Name$
.2 $\quad name : AS`Name$
.3 $\quad parms : AS`Parameters;$

500.0 $DLCALL :: cls : AS`Name$
.1 $\quad op : AS`Name;$

501.0 $CONTEXT :: cid : \text{-}CI`ContextId$
.1 $\quad isStmt : \mathbb{B};$

502.0 $POPCONTEXT :: cid : \text{-}CI`ContextId$
.1 $\quad isStmt : \mathbb{B};$

503.0 $NOP :: \;\; ;$

504.0 $StmtInstr = PPCALL \mid CALLGUARD \mid$
.1 $MKEXIT \mid EXITVAL \mid ISCONT \mid ISEXIT \mid ISNEEXIT \mid REMEXITVAL \mid$
.2 $PUSHTH \mid POPTH \mid HANDID \mid INDEXEXPRINSTR \mid NONDETSTMT \mid$
.3 $RANDOM \mid ATOMIC;$

505.0 $CALLGUARD :: hasobj : \mathbb{B}$
.1 $oprt : AS`Name;$

506.0 $PPCALL :: \;\; ;$

507.0 $MKEXIT :: isnil : \mathbb{B};$

508.0 $EXITVAL :: \;\; ;$

509.0 $ISCONT :: \;\; ;$

510.0 $ISEXIT :: \;\; ;$

511.0 $ISNEEXIT :: \;\; ;$

512.0 $REMEXITVAL :: \;\; ;$

513.0 $PUSHTH :: handid : \mathbb{N};$

514.0 $POPTH :: \;\; ;$

515.0 $HANDID :: handid : \mathbb{N};$

516.0 $INDEXEXPRINSTR = VERIFYINDEXARGS \mid INCRCOUNTER \mid TESTCOUNTER;$

517.0 $VERIFYINDEXARGS :: \;\; ;$

518.0 $INCRCOUNTER :: \;\; ;$

519.0 $TESTCOUNTER :: \;\; ;$

520.0 $NONDETSTMT :: \;\; ;$

521.0 $RANDOM :: \;\; ;$

522.0 $ATOMIC :: number : \mathbb{N}_1;$

523.0 $ExprInstr = APPLY \mid BINOP \mid UNOP \mid APPENDSEQ \mid APPENDMAP \mid APPENDTUP \mid$

.1 $ADDSET \mid SELELEM \mid SELSEQELEM \mid SETRNG \mid SUBSEQ \mid RECCONS \mid RECMOD \mid$

.2 $FIELDSEL \mid ISEMPTYSET \mid ISEMPTYSEQ \mid TOKENVAL \mid POLYINST \mid$

.3 $POLYTYPEINST \mid SEQCOMPBIND \mid SEQMAPOVER \mid TUPSEL \mid$

.4 $TYPEJUDGE \mid LASTRES;$

524.0 $APPENDSEQ :: \;\;;$

525.0 $APPENDMAP :: \;\;;$

526.0 $APPENDTUP :: \;\;;$

527.0 $ADDSET :: \;\;;$

528.0 $SETRNG :: \;\;;$

529.0 $SUBSEQ :: \;\;;$

530.0 $RECCONS :: tag : AS`Name$

.1 $length : \mathbb{N};$

531.0 $RECMOD :: length : \mathbb{N};$

532.0 $FIELDSEL :: \;\;;$

533.0 $ISEMPTYSET :: n : \mathbb{N}$

.1 $tp : [\mathrm{SEM}];$

534.0 $ISEMPTYSEQ :: n : \mathbb{N}$

.1 $semval : [\mathrm{SEM}];$

535.0 $TOKENVAL :: \;\;;$

536.0 $POLYTYPEINST :: nm : AS`Name;$

537.0 $POLYINST :: inst : AS`Type^*;$

538.0 $SEQCOMPBIND :: \;\;;$

539.0 $SEQMAPOVER :: \;\;;$

540.0 $TUPSEL :: index : \mathbb{R};$

541.0 $\mathit{TYPEJUDGE} :: \mathit{tp} : \mathit{AS}`\mathit{Type};$

542.0 $\mathit{LASTRES} :: \;\; ;$

543.0 $\mathit{SELELEM} :: \;\; ;$

544.0 $\mathit{SELSEQELEM} :: \mathit{direction} : [\mathrm{REVERSE}];$

545.0 $\mathit{APPLY} :: \;\; ;$

546.0 $\mathit{RETURN} :: \;\; ;$

547.0 $\mathit{ISCHECK} :: \mathit{type} : \mathit{AS}`\mathit{BasicType} \mid \mathit{AS}`\mathit{Name};$

548.0 $\mathit{EOCL} :: \;\; ;$

549.0 $\mathit{EvalStackInstr} = \mathit{EMPTYLIST} \mid \mathit{APPENDESTCK} \mid \mathit{PUSH} \mid \mathit{POP};$

550.0 $\mathit{PatInstr} = \mathit{MATCHANDBIND} \mid \mathit{TRYMATCH} \mid \mathit{ENUMAPPEND} \mid \mathit{SETUNION} \mid$
.1 $\mathit{SEQCONC} \mid \mathit{FIELDSAPPEND} \mid \mathit{MATCHVAL} \mid \mathit{SEQELEMMATCH} \mid$
.2 $\mathit{TRYANYMATCH} \mid \mathit{FREF} \mid \mathit{MOSREF} \mid \mathit{ASSIGNSD};$

551.0 $\mathit{MATCHVAL} :: \;\; ;$

552.0 $\mathit{MATCHANDBIND} :: \;\; ;$

553.0 $\mathit{TRYMATCH} :: \;\; ;$

554.0 $\mathit{TRYANYMATCH} :: \;\; ;$

555.0 $\mathit{ENUMAPPEND} :: \;\; ;$

556.0 $\mathit{SETUNION} :: \mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

557.0 $\mathit{SEQCONC} :: \mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

558.0 $\mathit{FIELDSAPPEND} :: \;\; ;$

559.0 $\mathit{SEQELEMMATCH} :: \mathit{index} : \mathbb{N};$

560.0 $\mathit{FREF} :: \mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

561.0 $\mathit{MOSREF} :: \mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

562.0 $\mathit{ASSIGNSD} :: \;\; ;$

563.0 $EMPTYLIST :: \;\; ;$

564.0 $APPENDESTCK :: \;\; ;$

565.0 $PUSH :: val : SEM`VAL \mid SEM`VAL^* \mid STKM`Pattern \mid$ .1 $SEM`BlkEnv\text{-set} \mid \mathbb{N}^* \mid AS`Fct$

566.0 $POP :: n : \mathbb{N};$

567.0 $VdmPPInstr = NEWEXPRS \mid SELFEXPR \mid ISOFCLASS \mid$
.1 $ISOFBASECLASS \mid SAMEBASECLASS \mid SAMECLASS \mid$
.2 $HISTORY \mid STARTLIST \mid GUARD \mid THREADID \mid DLCALL;$

568.0 $NEWEXPRS = NEWOBJ \mid NEWPOSABSOBJ \mid BINDINSTVAR \mid INITCLASS \mid$
$NEWCOMPL \mid$
.1 $PUSHCLNMCUROBJ \mid POPCLNMCUROBJ;$

569.0 $NEWOBJ :: nm : AS`Name$
.1 $dlobject : [\mathsf{token}];$

570.0 $NEWPOSABSOBJ :: nm : AS`Name;$

571.0 $BINDINSTVAR :: nm : AS`Name;$

572.0 $INITCLASS :: nm : AS`Name$
.1 $initno : \mathbb{N};$

573.0 $NEWCOMPL :: checkinv : \mathbb{B};$

574.0 $PUSHCLNMCUROBJ :: cl : AS`Name$
.1 $origcl : AS`Name;$

575.0 $POPCLNMCUROBJ :: \;\; ;$

576.0 $SELFEXPR :: \;\; ;$

577.0 $ISOFCLASS :: nm : AS`Name;$

578.0 $ISOFBASECLASS :: nm : AS`Name;$

579.0 $SAMEBASECLASS :: \;\; ;$

580.0 $SAMECLASS :: \;\; ;$

581.0 $HISTORY :: kind : HistoryKind$
.1 $opnms : AS`Name^*;$

582.0 $\mathit{HistoryKind} = \mathit{req} \mid \mathit{act} \mid \mathit{fin} \mid \mathit{waiting} \mid \mathit{active}$;

583.0 $\mathit{req} ::$;

584.0 $\mathit{act} ::$;

585.0 $\mathit{fin} ::$;

586.0 $\mathit{waiting} ::$;

587.0 $\mathit{active} ::$;

588.0 $\mathit{STARTLIST} :: \mathit{islist} : \mathbb{B}$;

589.0 $\mathit{GUARD} ::$;

590.0 $\mathit{THREADID} ::$;

The instruction *INCRTIME* represents the increment in time due to the duration of the preceding computation.

591.0 $\mathit{INCRTIME} :: \mathit{time} : \mathbb{N}$;

The instruction *RUNTIME_INCRTIME_PREF* is used to represent prefix operations whose execution time depends on runtime values.

592.0 $\mathit{RUNTIME\text{-}INCRTIME\text{-}PREF} :: \mathit{opr} : \mathit{AS}`\mathit{UnaryOp}$
.1 $\quad \mathit{oh} : \mathbb{N}$;

Here the field *oh* represents the overhead for the operation, which will be multiplied by some value to be determined at runtime depending on actual runtime values and the prefix operation being executed.

The operation *RUNTIME_INCRTIME_BIN* is similar, but for binary operations.

593.0 $\mathit{RUNTIME\text{-}INCRTIME\text{-}BIN} :: \mathit{opr} : \mathit{AS}`\mathit{BinaryOp}$
.1 $\quad \mathit{oh}1 : \mathbb{N}$
.2 $\quad \mathit{oh}2 : [\mathbb{N}]$;

The operation *RUNTIME_INCRTIME_SETSEQMAP* is used to represent those set, sequence and map operations whose execution time depends on the runtime size of the set, sequence or map in question.

594.0 $\mathit{RUNTIME\text{-}INCRTIME\text{-}SETSEQMAP} :: \mathit{oh} : \mathbb{N}$;

The operation *RUNTIME_INCRTIME_SETSEQMAP* is used to represent the time taken to start a list of threads. Here the field *oh* represents the overhead for starting a thread, which will be multiplied by the number of threads determined in the startlist statement at runtime.

595.0 $RUNTIME\text{-}INCRTIME\text{-}STARTLIST :: oh : \mathbb{N};$

596.0 $RUNTIME\text{-}INCRTIME\text{-}NEW :: oh : \mathbb{N}$
.1 $nm : AS`Name;$

597.0 $PUSHDURATION ::\ ;$

598.0 $POPDURATION :: time : \mathbb{R}$

end $INSTRTP$

1.12 Instruction Set Operations

module $INSTR$

imports

599.0 from AS all ,
600.0 from CI all ,
601.0 from IO all ,
602.0 from AUX all ,
603.0 from PAT all ,
604.0 from REP all ,
605.0 from SEM all ,
606.0 from $EXPR$ all ,
607.0 from $FREE$ all ,
608.0 from $SCHD$ all ,
609.0 from $STKM$ all ,
610.0 from $DEBUG$ all ,
611.0 from $RTERR$ all ,
612.0 from $STATE$ all ,
613.0 from $GLOBAL$ all ,
614.0 from $SCHDTP$ all ,
615.0 from $DEBUGTP$ all ,
616.0 from $INSTRTP$ all ,
617.0 from $TIMEMAP$ all ,
618.0 from $SETTINGS$ all ,
619.0 from $TIMEPARSER$ all

exports all

definitions

1.12.1 Auxiliary Instruction

operations

620.0 $\mathit{ExeCBR} : \mathbb{Z} \xrightarrow{o} ()$

.1 $\mathit{ExeCBR}\,(n) \triangleq$

.2 $\quad$(let $[b] = \mathit{STKM}`\mathit{Pop}\,(1)$ in

.3 $\quad$ if is-$\mathit{SEM}`\mathit{BOOL}\,(b)$

.4 $\quad$ then (if $b.v$

.5 $\qquad\quad$ then $\mathit{STKM}`\mathit{IncrPC}(n)$)

.6 $\quad$ else $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{BOOL\text{-}EXPECTED}$, nil , nil , $[]$));

621.0 $\mathit{ExeCNBR} : \mathbb{Z} \xrightarrow{o} ()$

.1 $\mathit{ExeCNBR}\,(n) \triangleq$

.2 $\quad$(let $[b] = \mathit{STKM}`\mathit{Pop}\,(1)$ in

.3 $\quad$ if is-$\mathit{SEM}`\mathit{BOOL}\,(b)$

.4 $\quad$ then (if $\neg\, b.v$

.5 $\qquad\quad$ then $\mathit{STKM}`\mathit{IncrPC}(n)$)

.6 $\quad$ else $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{BOOL\text{-}EXPECTED}$, nil , nil , $[]$));

622.0 $\mathit{ExeBR} : \mathbb{Z} \xrightarrow{o} ()$

.1 $\mathit{ExeBR}\,(n) \triangleq$

.2 $\quad \mathit{STKM}`\mathit{IncrPC}(n)$;

623.0 $\mathit{ExeERRINST} : \mathit{RTERR}`\mathit{ERR} \xrightarrow{o} ()$

.1 $\mathit{ExeERRINST}\,(\mathit{err}) \triangleq$

.2 $\quad \mathit{RTERR}`\mathit{Error}(\mathit{err}$, nil , nil , $[]$) ;

624.0 $\mathit{ExePRE} : () \xrightarrow{o} ()$

.1 $\mathit{ExePRE}\,() \triangleq$

.2 $\quad \mathit{STKM}`\mathit{Push}($mk-$\mathit{SEM}`\mathit{BOOL}\,(\mathit{SETTINGS}`\mathit{PreCheck}\,()))$;

625.0 $\mathit{ExePOST} : () \xrightarrow{o} ()$

.1 $\mathit{ExePOST}\,() \triangleq$

.2 $\quad \mathit{STKM}`\mathit{Push}($mk-$\mathit{SEM}`\mathit{BOOL}\,(\mathit{SETTINGS}`\mathit{PostCheck}\,()))$;

```
626.0 ExePOSTENV : AS`NameType* × CI`ContextId →o ()
  .1  ExePOSTENV (resnmtps, ci) ≜
  .2    let [resval] = STKM`Pop (1) in
  .3    if len resnmtps = 0
  .4    then STKM`PushBlkEnv
  .5              (mk-SEM`BlkEnv ({mk-AS`Name (["RESULT"], ci) ↦
  .6                                 mk-SEM`ValTp (resval, nil )},
  .7                                READ_ONLY))
  .8    elseif len resnmtps = 1
  .9    then STKM`PushBlkEnv
  .10             (mk-SEM`BlkEnv ({(hd resnmtps).nm ↦
  .11                                mk-SEM`ValTp (resval, (hd resnmtps).tp)},
  .12                               READ_ONLY))
  .13   elseif is-SEM`TUPLE (resval) ∧ len resnmtps = len resval.v
  .14   then (STKM`PushBlkEnv
  .15              (mk-SEM`BlkEnv ({(hd resnmtps).nm ↦
  .16                                 mk-SEM`ValTp (resval.v (1), (hd resnmtps).tp)},
  .17                                READ_ONLY)) ;
  .18         for i = 2 to len resnmtps
  .19         do let mk-AS`NameType (nm, tp, -) = resnmtps (i) in
  .20              STKM`AppendToTopBlkEnv(nm, resval.v (i), tp) )
  .21   else RTERR`Error(RTERR`WRONG-NO-RES, nil , nil , []) ;


627.0 ExeNOBODY : char* × AS`Name × AS`Name × AS`Parameters →o ()
  .1  ExeNOBODY (err, modname, name, parms) ≜
  .2    (dcl arg-lv : SEM`VAL* := [];
  .3     if ∀ parm ∈ elems parms · is-AS`PatternName (parm)
  .4     then (for mk-AS`PatternName (nm, -) in parms
  .5             do arg-lv := arg-lv ⁀ [if nm = nil
  .6                                    then mk-SEM`NIL ()
  .7                                    else STATE`LookUp (nm)];
  .8           if (IsVdmStdLib (modname, name, arg-lv))
  .9           then skip
  .10          else RTERR`Error(err, nil , nil , []) )
  .11    else RTERR`Error(err, nil , nil , []) );


628.0 IsVdmStdLib : AS`Name × AS`Name × SEM`VAL* →o 𝔹
  .1  IsVdmStdLib (modname, name, arg-lv) ≜
  .2    return false;
```

1.12.2 Evaluation Stack Instructions

EMPTYLIST

The operation *ExeEMPTYLIST* pushes an empty list (of semantic values) on the evaluation list. The instruction is used in connection with evaluation of apply expressions.

629.0 $\mathit{ExeEMPTYLIST} : () \xrightarrow{o} ()$
.1 $\mathit{ExeEMPTYLIST}\ () \triangleq$
.2 $\quad \mathit{STKM}`\mathit{Push}([])\ ;$

APPENDESTCK

The operation *ExeAPPENDESTCK* pops two elements of the evaluation stack. The second element is assumed to be a list of semantic values, and the first element a semantic value. The operation appends the first element to the list of the semantics values and pushes the list on the evaluation stack again.

630.0 $\mathit{ExeAPPENDESTCK} : () \xrightarrow{o} ()$
.1 $\mathit{ExeAPPENDESTCK}\ () \triangleq$
.2 $\quad \textsf{let}\ [\mathit{arg\text{-}l}, \mathit{arg}] = \mathit{STKM}`\mathit{Pop}\ (2)\ \textsf{in}$
.3 $\quad \mathit{STKM}`\mathit{Push}(\mathit{arg\text{-}l} \frown [\mathit{arg}])$
.4 $\textsf{pre}\ \ \mathit{STKM}`\mathit{LenES}\ (2) \wedge$
.5 $\quad\ \ \textsf{let}\ [\mathit{arg}, \mathit{arg\text{-}l}] = \mathit{STKM}`\mathit{GetES}\ (2)\ \textsf{in}$
.6 $\quad\ \ \mathit{SEM}`\mathit{IsSemVal}\ (\mathit{arg}) \wedge \mathit{SEM}`\mathit{IsSemValSeq}\ (\mathit{arg\text{-}l})\ ;$

1.12.3 Context Stack Instructions

The operation *ExeCONTEXT* pushes an item on the context stack and increments the coverage information for the AST node it represents.

Furthermore it handles breakpoints by reading the debug flag of the stack evaluator and checks if the context id *cid* is in the set of active breakpoints. The *BREAK* state of the evaluator is set to signal whether the evaluation should be terminated.

631.0 $\mathit{ExeCONTEXT} : \mathit{CI}`\mathit{ContextId} \times \mathbb{B} \xrightarrow{o} ()$
.1 $\mathit{ExeCONTEXT}\ (\mathit{cid}, \mathit{isStmt}) \triangleq$
.2 $\quad (\mathit{STKM}`\mathit{SetCid}(\mathit{cid})\ ;$
.3 $\quad\ \textsf{if}\ \mathit{cid} \neq \mathit{CI}`\mathit{NilContextId}$
.4 $\quad\ \textsf{then}\ (\mathit{CI}`\mathit{IncTestCoverageInfo}(\mathit{cid})\ ;$
.5 $\qquad\qquad \textsf{if}\ \mathit{DEBUG}`\mathit{ActiveBreakpoint}\ (\mathit{cid})$
.6 $\qquad\qquad \textsf{then}\ \mathit{STKM}`\mathit{SetBREAK}(\textsf{true})$

.7 else if cases $STKM`GetDebugFlag\,()$:
.8 mk-$STKM`Step\,(level) \rightarrow isStmt \wedge STKM`CallStackLevel\,() \leq level$,
.9 mk-$STKM`StepIn\,() \rightarrow$ true,
.10 mk-$STKM`SingleStep\,(level) \rightarrow STKM`CallStackLevel\,() \leq level$,
.11 mk-$STKM`Finish\,(level) \rightarrow STKM`CallStackLevel\,() \leq level - 1$,
.12 others $\rightarrow$ false
.13 end
.14 then $STKM`SetBREAK$(true)));

1.12.4 Environment Instructions

632.0 $ExeSELBLKENV : \mathbb{N} \xrightarrow{o} ()$
.1 $ExeSELBLKENV\,(n) \triangleq$
.2 let $val\text{-}l = STKM`Pop\,(n)$ in
.3 let $env \in$ hd $val\text{-}l$ in
.4 $(STKM`PushBlkEnv(env)$;
.5 $STKM`Push($hd $val\text{-}l \setminus \{env\})$;
.6 for $i = 2$ to n
.7 do $STKM`Push(val\text{-}l\,(i))$)
.8 pre $STKM`LenES\,(n)$;

633.0 $ExeAPPENDBLKENV : AS`Name \times [AS`Type] \xrightarrow{o} ()$
.1 $ExeAPPENDBLKENV\,(id, tp) \triangleq$
.2 let $[val] = STKM`Pop\,(1)$ in
.3 $STKM`AppendToTopBlkEnv(id, val, tp)$
.4 pre $STKM`LenES\,(1) \wedge$
.5 let $[val] = STKM`GetES\,(1)$ in
.6 $SEM`IsSemVal\,(val)$;

634.0 $ExeREMSTACKELEM : \mathbb{N} \xrightarrow{o} ()$
.1 $ExeREMSTACKELEM\,(n) \triangleq$
.2 let $val\text{-}l = STKM`Pop\,(n)$ in
.3 for $i = 2$ to n
.4 do $STKM`Push(val\text{-}l\,(i))$
.5 pre $STKM`LenES\,(n)$;

635.0 $ExeSWAP : () \xrightarrow{o} ()$
.1 $ExeSWAP\,() \triangleq$
.2 let $[elem1, elem2] = STKM`Pop\,(2)$ in
.3 $(STKM`Push(elem2)$;
.4 $STKM`Push(elem1)$);

636.0 $ExeMULTBINDL : \mathbb{N} \times PAT`PARTITION \xrightarrow{o} ()$
.1 $ExeMULTBINDL\,(n, part) \triangleq$
.2 (dcl $pat\text{-}lp : STKM`Pattern^* := []$,
.3 $seq\text{-}lv : SEM`VAL^* := []$;
.4 let $valpat\text{-}l = STKM`Pop\,(n)$ in
.5 (for $valpat$ in $valpat\text{-}l$
.6 do if $SEM`IsSemVal\,(valpat)$
.7 then $seq\text{-}lv := seq\text{-}lv \curvearrowright [valpat]$
.8 else $(pat\text{-}lp := pat\text{-}lp \curvearrowright [valpat]$;
.9 if len $pat\text{-}lp >$ len $seq\text{-}lv$
.10 then $seq\text{-}lv := seq\text{-}lv \curvearrowright [seq\text{-}lv$ (len $seq\text{-}lv)])$;
.11 let $env\text{-}s = PAT`EvalMultBindSeq\,(pat\text{-}lp, seq\text{-}lv, part)$ in
.12 if $part = \mathrm{DO\_PARTITION}$
.13 then let $envs \in env\text{-}s$ in
.14 $STKM`Push(envs)$
.15 else $STKM`Push(env\text{-}s)$));

637.0 $ExePOPBLKENV : () \xrightarrow{o} ()$
.1 $ExePOPBLKENV\;() \triangleq$
.2 $STKM`PopBlkEnv()$
.3 pre $\neg\, STKM`IsEmptyEnvL\,()$;

638.0 $ExeADDTOBLKENV : () \xrightarrow{o} ()$
.1 $ExeADDTOBLKENV\;() \triangleq$
.2 let $[found\text{-}s] = STKM`Pop\,(1)$ in
.3 $STKM`Push(found\text{-}s \cup \{STKM`TopBlkEnv\,()\})$;

1.12.5 Statement Instructions

639.0 $ExeMKEXIT : \mathbb{B} \xrightarrow{o} ()$
.1 $ExeMKEXIT\,(isnil) \triangleq$
.2 if $isnil$
.3 then $STKM`Push($mk-$SEM`EXIT$ (nil))
.4 else let $[val] = STKM`Pop\,(1)$ in
.5 $STKM`Push($mk-$SEM`EXIT\,(val))$;

640.0 $ExeEXITVAL : () \xrightarrow{o} ()$
.1 $ExeEXITVAL\,() \triangleq$
.2 $STKM`GotoTrapHandler()$;

641.0 $ExeISCONT : () \xrightarrow{o} ()$

.1 $ExeISCONT\,() \triangleq$

.2 $\quad$ let $[val] = STKM`GetES\,(1)$ in

.3 $\quad STKM`Push(\mathsf{mk\text{-}}SEM`BOOL\,(\mathsf{is\text{-}}SEM`CONT\,(val)))$

.4 pre $STKM`LenES\,(1) \land$

.5 $\quad$ let $[val] = STKM`GetES\,(1)$ in

.6 $\quad SEM`IsSemVal\,(val)$;

642.0 $ExeISEXIT : () \xrightarrow{o} ()$

.1 $ExeISEXIT\,() \triangleq$

.2 $\quad$ let $[val] = STKM`GetES\,(1)$ in

.3 $\quad STKM`Push(\mathsf{mk\text{-}}SEM`BOOL\,(\mathsf{is\text{-}}SEM`EXIT\,(val)))$

.4 pre $STKM`LenES\,(1) \land$

.5 $\quad$ let $[val] = STKM`GetES\,(1)$ in

.6 $\quad SEM`IsSemVal\,(val)$;

643.0 $ExeISNEEXIT : () \xrightarrow{o} ()$

.1 $ExeISNEEXIT\,() \triangleq$

.2 $\quad$ let $[val] = STKM`GetES\,(1)$ in

.3 $\quad STKM`Push(\mathsf{mk\text{-}}SEM`BOOL\,(\mathsf{is\text{-}}SEM`EXIT\,(val) \land val.v \neq \mathsf{nil}\,))$

.4 pre $STKM`LenES\,(1) \land$

.5 $\quad$ let $[val] = STKM`GetES\,(1)$ in

.6 $\quad SEM`IsSemVal\,(val)$;

644.0 $ExeREMEXITVAL : () \xrightarrow{o} ()$

.1 $ExeREMEXITVAL\,() \triangleq$

.2 $\quad$ let $[val] = STKM`GetES\,(1)$ in

.3 $\quad STKM`Push(val.v)$

.4 pre $STKM`LenES\,(1) \land$

.5 $\quad$ let $[val] = STKM`GetES\,(1)$ in

.6 $\quad \mathsf{is\text{-}}SEM`EXIT\,(val) \land val.v \neq \mathsf{nil}$;

645.0 $ExePUSHTH : \mathbb{Z} \xrightarrow{o} ()$

.1 $ExePUSHTH\,(movePC) \triangleq$

.2 $\quad STKM`PushTS(movePC)$;

646.0 $ExePOPTH : () \xrightarrow{o} ()$

.1 $ExePOPTH\,() \triangleq$

.2 $\quad STKM`PopTS()$

.3 pre $\neg$ *STKM‘EmptyTS* () ;

1.12.6 Expression Instructions

LookUp Instruction

The operation *ExeLOOKUP* looks up the name or oldname and pushes the result on the evaluation stack.

647.0 *ExeLOOKUP* : *AS‘Name* | *AS‘OldName* $\xrightarrow{o}$ ()
.1 *ExeLOOKUP* (*name*) $\triangleq$
.2 (let *semname* = if is-*AS‘Name* (*name*)
.3 then *STATE‘LookUp* (*name*)
.4 else *STKM‘EvalOldName* (*name*) in
.5 *STKM‘Push*(*semname*));

648.0 *ExeLOOKUPSTATIC* : *AS‘Name* $\xrightarrow{o}$ ()
.1 *ExeLOOKUPSTATIC* (*name*) $\triangleq$
.2 let mk- (*access*, *val*) = *STATE‘LookUpStatic* (*name*) in
.3 if *access*
.4 then *STKM‘Push*(*val*)
.5 else *RTERR‘Error*(*RTERR‘STATIC-NOT-IN-SCOPE*, nil , nil , []) ;

Apply Expression

The instruction *APPLY* pops two elements from the evaluation stack: the semantic value of the ”applicator” (a semantic value of an operation/function, map or sequence) and a sequence consisting of the semantic values corresponding to the application parameters.

Consider the function application of *f* below, assume that *a* is bound to the value *true*.

```
f( mk_(3,4), a)
```

The evaluation stack would then look like:

"mk_SEM‘Fn(...) | mk_SEM‘MAP(...) | mk_SEM‘SEQ(...)"
[mk_SEM‘TUPLE([mk_SEM‘NUM(3), mk_SEM‘NUM(4)]), mk_SEM‘Bool(true)]

```
649.0 ExeAPPLY : () -o-> ()
ExeAPPLY () ≜
  let [fct-v, arg-lv] = STKM`Pop (2) in
  ApplyOpFnMapSeq(fct-v, arg-lv) ;
```

```
650.0 ApplyOpFnMapSeq : SEM`VAL × (SEM`VAL* | SEM`VAL) -o-> ()
ApplyOpFnMapSeq (fct-v, arg-lv) ≜
  cases true:
    (is-SEM`MAP (fct-v)) → STKM`Push(EXPR`EvalMapApply (fct-v, arg-lv)) ,
    (is-SEM`SEQ (fct-v)) → STKM`Push(EXPR`EvalSeqApply (fct-v, arg-lv)) ,
    (is-SEM`CompExplFN (fct-v)) → EvalCompExplFnApply(fct-v, arg-lv) ,
    (is-SEM`ExplOP (fct-v)) → EvalExplOpApply(fct-v, arg-lv) ,
    (is-SEM`OverOPFN (fct-v)) → EvalOverOpFnApply(fct-v, arg-lv) ,
    (is-SEM`ImplFN (fct-v)) →
         RTERR`Error(RTERR`IMPL-FN-APPLY, nil , nil , []) ,
    (is-SEM`ImplOP (fct-v)) →
         RTERR`Error(RTERR`IMPL-OP-CALL, nil , nil , []) ,
    others → RTERR`Error(RTERR`INTERNAL-ERROR, nil , nil , [])
  end
pre is_(arg-lv, SEM`VAL) ⇒ is_(fct-v, SEM`CompExplFN) ;
```

Function Applications

Function application involves setting up the right block environment. If the function is a curried function the block-environment of the semantic value is updated with the now bound free variables. The block-environment of the function pushed again on the evaluation stack.

```
651.0 EvalCompExplFnApply : SEM`CompExplFN × (SEM`VAL | SEM`VAL*) -o-> ()
EvalCompExplFnApply (mk-SEM`CompExplFN (fn-l, curobjref), arg) ≜
  let arg-lv = if SEM`IsSemVal (arg)
               then [arg]
               else arg in
  let new-fn-l = [μ (fn-l (i), objref ↦ curobjref) | i ∈ inds fn-l] in
  if len new-fn-l = 1
  then EvalExplFnApply(hd new-fn-l, arg-lv)
  else (STKM`Push(mk-SEM`CompExplFN (tl new-fn-l, nil )) ;
        STKM`IncrPC(− 1) ;
        EvalExplFnApply(hd new-fn-l, arg-lv) );
```

```
652.0 EvalExplFnApply : SEM`ExplFN × SEM`VAL* -o-> ()
   .1 EvalExplFnApply (fct-v, arg-lv) △
   .2   if fct-v.instr = nil
   .3   then if len arg-lv = 1
   .4        then STKM`Push(hd arg-lv)
   .5        else STKM`Push(mk-SEM`TUPLE (arg-lv))
   .6   else (if len fct-v.parms > 1
   .7        then skip
   .8        else let clmodName = fct-v.modName,
   .9                 fnName = fct-v.fnName,
  .10                 newNm = if fnName = nil
  .11                         then mk-AS`Name (["lambdaapplication"], CI`NilContextId)
  .12                         else AUX`ConstructDoubleName (clmodName, fnName) in
  .13             STKM`PushCS(fct-v, newNm, arg-lv, FNOP) ;
  .14        EnvSetUpExplFnApply(fct-v, arg-lv) );
```

The operation *EnvSetUpExplFnApply* sets up the block environment and for VDM++ the right object scope for a function application.

For an external dlclass method, we push the argument parameters on the evaluation stack, because it is simpler for the external code to be passed a sequence of parameters than a blkenv. The DLCALL instruction is supposed to pop the sequence of parameters and pass them on to the external C++ code.

```
653.0 EnvSetUpExplFnApply : SEM`ExplFN × SEM`VAL* -o-> ()
   .1 EnvSetUpExplFnApply (mk-SEM`ExplFN (tp, parms, instr, closenv,
   .2                                     tm, fnName, clmodName, -, objref), arg-lv) △
   .3   (STKM`PushCurObjTab2OS() ;
   .4    if objref ≠ nil
   .5    then (STKM`PushCurObj(objref, clmodName, clmodName) )
   .6    else (STKM`PushClNmCurObj(clmodName, clmodName) );
   .7    if parms ≠ [] ∧ ¬ len arg-lv = len parms (1)
   .8    then RTERR`Error(RTERR`WRONG-NO-OF-ARGS, nil , nil , []) ;
   .9    if SETTINGS`DTC ()
  .10    then if (¬ len arg-lv = len tp.fndom)
  .11         then RTERR`Error(RTERR`WRONG-NO-OF-ARGS, nil , nil , [])
  .12         elseif (¬ ∀ i ∈ inds tp.fndom ·
  .13                     STATE`SubType (arg-lv (i), tp.fndom (i)))
  .14         then RTERR`Error(RTERR`TYPE-INCOMP, nil , nil , []) ;
  .15    STKM`PushEmptyEnv() ;
  .16    STKM`PushBlkEnv(closenv) ;
  .17    let env-s = PAT`MatchLists (hd parms, arg-lv) in
  .18    if env-s = {}
  .19    then RTERR`Error(RTERR`EMPTY-ENV-S, nil , nil , [])
```

```
else let env ∈ env-s in
     (STKM‘PushBlkEnv(env) ;
      if len parms > 1
      then let newenv = AUX‘CombineBlkEnv (closenv, env) in
           (STKM‘PopEnvL() ;
            STKM‘PopOS() ;
            if objref = nil
            then (STKM‘PopClNmCurObj() )
            else (STKM‘PopCurObj() );
            let fn = mk-SEM‘ExplFN (tp.fnrng, tl parms, instr,
                                    newenv, tm, fnName, clmodName, nil ,
                                    objref) in
            STKM‘Push(SEM‘CompFN (fn)) )
      else (STKM‘PushTypeInst(tm) ;
           if fnName ≠ nil
           then if STATE‘IsDLOp (clmodName, fnName)
                then STKM‘Push(arg-lv) )));
```

HC: 990914: This REMARK should be deleted when the spec and implementation is consistent. In version 1.1.2.52. The *EvalExplOpApply* was specified like:

```
-- EvalExplOpApply: SEM‘ExplOP * seq of SEM‘VAL ==> ()
-- EvalExplOpApply(opsem, arg_lv) ==
-- #ifdef
--  let opnm = mk_AS‘Name([hd (opsem.modName.ids),hd
--                            (opsem.fnName.ids)],
--                        opsem.fnName.cid)
--  in
--  (STATE‘UpdateHistCount(opnm,<act>,opsem.objref);
--#endif
--  (StepCheck();
--   EnvSetUpExplOpApply(opsem, arg_lv);
--   STKM‘PushCS(opsem));
--#ifdef
--   );
--#endif
```

Updating history counter should not be updated before the operation is actually called. At this point we don't know a run-time error will occur because of e.g. wrong number of arguments to the operation call.

Secondly, the unpacking the module name and operation name are more or less done in the operation *EnvSetUpExplOpApply*. It seems therefore better to update the history counters in this function. This will make it easier to read and will work correctly.

In the implementation of instructions.vdm, instruction.cc, version 1.3, I have therefore it as it is in this version.

```
654.0 EvalExplOpApply : SEM`ExplOP × SEM`VAL* -o-> ()
EvalExplOpApply (opsem, arg-lv) △
  (let clmodName = opsem.modName,
       opName = opsem.fnName,
       newNm = AUX`ConstructDoubleName (clmodName, opName) in
   (if ¬ STATE`IsClassInit (clmodName)
    then STATE`InitClassName(clmodName) ;
    STKM`PushCS(opsem, newNm, arg-lv, FNOP) );
   EnvSetUpExplOpApply(opsem, arg-lv) );
```

For an external dlclass method, we push the argument parameters on the evaluation stack, because it is simpler for the external code to be passed a sequence of parameters than a blkenv. The DLCALL instruction is supposed to pop the sequence of parameters and pass them on to the external C++ code.

```
655.0 EnvSetUpExplOpApply : SEM`ExplOP × SEM`VAL* -o-> ()
EnvSetUpExplOpApply (mk-SEM`ExplOP (tp, parms, -, opName,
                                    clmodName, -, objref), arg-lv) △
  (STKM`PushCurObjTab2OS() ;
   if objref ≠ nil
   then (STKM`PushCurObj(objref, clmodName, clmodName) )
   else (STKM`PushClNmCurObj(clmodName, clmodName) );
   if ¬ len arg-lv = len parms
   then RTERR`Error(RTERR`WRONG-NO-OF-ARGS, nil , nil , []) ;
   if SETTINGS`DTC ()
   then if (¬ len arg-lv = len tp.opdom)
        then RTERR`Error(RTERR`WRONG-NO-OF-ARGS, nil , nil , [])
        elseif (¬ ∀ i ∈ inds tp.opdom ·
                     STATE`SubType (arg-lv (i), tp.opdom (i)))
        then RTERR`Error(RTERR`TYPE-INCOMP-APPLY, nil , nil , []) ;
   STKM`PushEmptyEnv() ;
   let env-s = PAT`MatchLists (parms, arg-lv) in
   if env-s = {}
   then RTERR`Error(RTERR`EMPTY-ENV-S, nil , nil , [])
   else let env ∈ env-s in
        STKM`PushBlkEnv(env) ;
   if STATE`IsDLOp (clmodName, opName)
   then STKM`Push(arg-lv) ;
   let opname = AUX`ConstructDoubleName (clmodName, opName) in
   STATE`UpdateHistCount(opname, mk-INSTRTP`act (), objref) );
```

The EvalOverOpFnApply operation is used for overloaded operations/functions. Depending upon the argument list the operation/function with the right type is selected (if none or more than one are found an error is reported).

```
656.0 EvalOverOpFnApply : SEM`OverOPFN × SEM`VAL* -o-> ()
   .1 EvalOverOpFnApply (fct-v, arg-lv) △
   .2   (dcl founds : AS`Name -m-> (SEM`FN | SEM`OP) := {↦};
   .3    let over = fct-v.overload,
   .4        objref = fct-v.objref,
   .5        arglen = len arg-lv in
   .6    (for all mk- (manglenm, clsnm) ∈ dom over
   .7     do let tp-l = over (mk- (manglenm, clsnm)) in
   .8         if len tp-l = arglen ∧
   .9            ∀ i ∈ inds arg-lv ·
  .10                 STATE`RealSubType (arg-lv (i), tp-l (i), false)
  .11         then let mk- (found, opfn) =
  .12                       STATE`LookupAllFnsOpsPolys (clsnm, manglenm) in
  .13                (if found
  .14                 then let mk- (opval, access) = opfn in
  .15                        if STATE`AccessOk (access, STKM`GetOrigCl (), clsnm)
  .16                        then if clsnm ∈ dom founds
  .17                             then RTERR`Error(RTERR`MULTIOVERLOADED, nil , nil , [])
  .18                             else founds := founds ⊎m {clsnm ↦ opval}
  .19                        else RTERR`Error(RTERR`NOT-IN-SCOPE, nil , nil , []) );
  .20      cases  dom founds:
  .21        {} → RTERR`Error(RTERR`NOOVERLOADED, nil , nil , []) ,
  .22        {cl} → (dcl f : SEM`FN | SEM`OP := founds (cl);
  .23                 if is_(f, SEM`ExplOP | SEM`ExplFN)
  .24                 then f.objref := objref;
  .25                 ApplyOpFnMapSeq(f, arg-lv) ),
  .26        - → let mk- (doesthere, child) = STATE`ExistsOneChild (dom founds) in
  .27             if doesthere
  .28             then (dcl f : SEM`FN | SEM`OP := founds (child);
  .29                   if is_(f, SEM`ExplOP | SEM`ExplFN)
  .30                   then f.objref := objref;
  .31                   ApplyOpFnMapSeq(f, arg-lv) )
  .32             else RTERR`Error(RTERR`MULTIOVERLOADED, nil , nil , [])
  .33     end));
```

Call Stmt

The call statement is special to VDM++

```
657.0 ExeCALLGUARD : 𝔹 × AS`Name -o-> ()
   .1 ExeCALLGUARD (hasobj, oprt) △
   .2   let [args] = STKM`Pop (1),
```

.3 $[obj] =$ if $hasobj$
.4 then $STKM`Pop\,(1)$
.5 else [nil] in
.6 (if $hasobj$
.7 then (if $\neg$ is-$SEM`OBJ\text{-}Ref\,(obj)$
.8 then $RTERR`Error(RTERR`OBJ\text{-}REF\text{-}EXP\text{-}CALL,$ nil , nil , []) ;
.9 $STKM`PushCurObj(obj,$ nil $, STKM`GetCurCl\,())$);
.10 (dcl $op\text{-}v : [SEM`VAL] := STATE`LookUp\,(oprt)$;
.11 if is-$SEM`OverOPFN\,(op\text{-}v)$
.12 then (let $over = op\text{-}v.overload$ in
.13 for all mk- $(manglenm, clsnm) \in$ dom $over$
.14 do let $tp\text{-}l = over$ (mk- $(manglenm, clsnm)$) in
.15 if len $tp\text{-}l =$ len $args \wedge$
.16 $\forall\, i \in$ inds $args \cdot$
.17 $STATE`RealSubType\,(args\,(i), tp\text{-}l\,(i),$ false)
.18 then let mk- $(found,$ mk- $(opfn, \text{-})) =$
.19 $STATE`LookupAllFnsOpsPolys\,(clsnm, manglenm)$ in
.20 if $found$
.21 then $op\text{-}v := opfn$);
.22 if $\neg$ is-$SEM`ExplOP\,(op\text{-}v)$
.23 then $RTERR`Error(RTERR`NOT\text{-}EXPL\text{-}OP\text{-}CALL,$ nil , nil , []) ;
.24 $GuardCheck(op\text{-}v, obj)$;
.25 if $hasobj$
.26 then $(op\text{-}v.objref := obj$;
.27 $STKM`PopCurObj()$);
.28 $STKM`Push(args)$;
.29 $STKM`Push(op\text{-}v)$));

658.0 $ExePPCALL : () \xrightarrow{o} ()$

.1 $ExePPCALL\,() \triangleq$
.2 let $[args, op\text{-}v] = STKM`Pop\,(2)$ in
.3 $EvalExplOpApply(op\text{-}v, args)$;

DLCALL Instruction

659.0 $ExeDLCALL : AS`Name \times AS`Name \xrightarrow{o} ()$

.1 $ExeDLCALL\,(clname, fnopname) \triangleq$
.2 (let $[args] = STKM`Pop\,(1)$,
.3 $name = AUX`SingleNameToString\,(AUX`ConstructDoubleName\,(clname, fnopname))$ in
.4 $RTERR`Error(RTERR`LOGDLCALL \curvearrowright name,$ nil , nil , []));

Return Instruction

The instruction *RETURN* that is used to return from a function call/application cleans the block environment and returns to origin caller. In case the value to be returned (present at the top of the evaluation stack) contains any references to objects which are not known in the calling environment references to these temporary objects must be added to the calling function environment. If this is not done the object references would be deleted because the reference counter would reach zero.

```
660.0  ExeRETURN : () -o-> ()
   .1  ExeRETURN () △
   .2    (CleanFunctionApply() ;
   .3     STKM`PopCS() )
   .4  pre  STKM`CallStackLevel () > 0 ;
```

```
661.0  ExeISCHECK : AS`BasicType | AS`Name -o-> ()
   .1  ExeISCHECK (Type) △
   .2    let [arg-v] = STKM`Pop (1),
   .3        nid = CI`NilContextId in
   .4    if is-AS`BasicType (Type)
   .5    then let v =
   .6                (is-SEM`BOOL (arg-v)∧Type = mk-AS`BasicType (BOOLEAN, nid)) ∨
   .7                (AUX`IsNat (arg-v) ∧ Type = mk-AS`BasicType (NAT, nid)) ∨
   .8                (AUX`IsNatOne (arg-v)∧Type = mk-AS`BasicType (NATONE, nid)) ∨
   .9                (AUX`IsInt (arg-v) ∧ Type = mk-AS`BasicType (INTEGER, nid)) ∨
   .10               (AUX`IsReal (arg-v) ∧ Type = mk-AS`BasicType (REAL, nid)) ∨
   .11               (AUX`IsReal (arg-v) ∧ Type = mk-AS`BasicType (RAT, nid)) ∨
   .12               (is-SEM`TOKEN (arg-v)∧Type = mk-AS`BasicType (TOKEN, nid)) ∨
   .13               (is-SEM`CHAR (arg-v) ∧ Type = mk-AS`BasicType (CHAR, nid)) in
   .14         STKM`Push(mk-SEM`BOOL (v))
   .15   elseif is-SEM`REC (arg-v)
   .16   then let mk-SEM`REC (tag, -, -) = arg-v in
   .17         let mk- (tagname, -) = AUX`ExtractTagName (Type) in
   .18         let mk- (bval, -) = AUX`IsRecSel (tag) in
   .19         if bval
   .20         then STKM`Push(mk-SEM`BOOL (tagname = tag))
   .21         else RTERR`Error(RTERR`RECORD-TAG-UNKNOWN, arg-v, nil , [])
   .22   else STKM`Push(mk-SEM`BOOL (false))
   .23 pre  STKM`CallStackLevel () > 0 ;
```

The instruction *EOCL* is solely used to signal the ending of the instruction sequence of the debugger command that initiated the evaluation. This instruction is appended to the instruction sequence returned by *CMPL‘E2I* to make sure that the call stack is “cleaned” properly. The operation *ExeEOCL* simply removes the final item from the call stack.

662.0 $ExeEOCL : () \xrightarrow{o} ()$
.1 $ExeEOCL\,() \triangleq$
.2 $\quad STKM‘PopCS()$
.3 pre $STKM‘CallStackLevel\,() > 0$;

663.0 $CleanFunctionApply : () \xrightarrow{o} ()$
.1 $CleanFunctionApply\,() \triangleq$
.2 $\quad$(let mk-$STKM‘CallStackItem\,(\mathrm{FNOP}, semval, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}) = STKM‘HeadCS\,()$ in
.3 $\quad$(cases true:
.4 $\qquad$(is-$SEM‘ExplFN\,(semval)) \rightarrow CleanExplFnApply(semval)$,
.5 $\qquad$(is-$SEM‘ExplOP\,(semval)) \rightarrow CleanExplOpApply(semval)$,
.6 $\qquad$others $\rightarrow RTERR‘Error(RTERR‘INTERNAL\text{-}ERROR,$ nil , nil , $[])$
.7 $\quad$end))
.8 pre $STKM‘CallStackLevel\,() > 0$;

664.0 $CleanExplFnApply : SEM‘ExplFN \xrightarrow{o} ()$
.1 $CleanExplFnApply\,($mk-$SEM‘ExplFN\,(tp, \text{-}, \text{-}, \text{-},$
.2 $\qquad\qquad \text{-}, \text{-}, \text{-}, \text{-}, objref)) \triangleq$
.3 $\quad$(let $res\text{-}v = STKM‘Head\,()$ in
.4 $\quad$(if $SETTINGS‘DTC\,() \wedge$
.5 $\qquad \neg\, STATE‘SubType\,(res\text{-}v, tp.fnrng)$
.6 $\quad$then $RTERR‘Error(RTERR‘TYPE\text{-}INCOMP, res\text{-}v, tp.fnrng, [])$;
.7 $\quad STKM‘PopEnvL()$;
.8 $\quad STKM‘PopOS()$;
.9 $\quad$if $objref =$ nil
.10 $\quad$then $STKM‘PopClNmCurObj()$;
.11 $\quad$if $objref \neq$ nil
.12 $\quad$then $STKM‘PopCurObj()$;
.13 $\quad STKM‘PopTypeInst()$))
.14 pre $STKM‘CallStackLevel\,() > 0$;

665.0 $CleanExplOpApply : SEM‘ExplOP \xrightarrow{o} ()$
.1 $CleanExplOpApply\,($mk-$SEM‘ExplOP\,(tp, \text{-}, \text{-}, fnName, modName, \text{-}, objref)) \triangleq$
.2 $\quad$let $res\text{-}v = STKM‘Head\,()$ in
.3 $\quad$(if is-$SEM‘RETURN\,(res\text{-}v)$
.4 $\quad$then let - $= STKM‘Pop\,(1)$ in
.5 $\qquad STKM‘Push($mk-$SEM‘CONT\,())$

.6 elseif $SETTINGS`DTC\,() \wedge \neg$ is-$SEM`EXIT\,(res\text{-}v) \wedge$
.7 $fnName \neq modName \wedge$
.8 $\neg\, STATE`SubType\,(res\text{-}v, tp.oprng)$
.9 then $RTERR`Error(RTERR`TYPE\text{-}INCOMP, res\text{-}v, tp.oprng, [\,])$;
.10 $STKM`PopEnvL()$;
.11 $STKM`PopOS()$;
.12 let $opnm =$ mk-$AS`Name\,([$hd $(modName.ids)$, hd $(fnName.ids)],$
.13 $fnName.cid)$ in
.14 $STATE`UpdateHistCount(opnm,$ mk-$INSTRTP`fin\,(), objref)$;
.15 if $objref =$ nil
.16 then $STKM`PopClNmCurObj()$;
.17 if $objref \neq$ nil
.18 then $STKM`PopCurObj()\,)$
.19 pre $STKM`CallStackLevel\,() > 0$;

Unary Expressions

666.0 $EvalUNOP : AS`UnaryOp \mid \mathrm{MAPINVERSE} \xrightarrow{o} ()$
.1 $EvalUNOP\,(opr) \triangleq$
.2 let $[arg\text{-}v] = STKM`Pop\,(1)$ in
.3 let $res\text{-}v =$
.4 cases opr :
.5 NUMPLUS,
.6 NUMMINUS,
.7 FLOOR,
.8 NUMABS $\rightarrow EXPR`EvalNumUnaryExpr\,(opr, arg\text{-}v)$,
.9 NOT $\rightarrow EXPR`EvalLogUnaryExpr\,(opr, arg\text{-}v)$,
.10 SETCARD,
.11 SETDISTRUNION,
.12 SETDISTRINTERSECT,
.13 SETPOWER $\rightarrow EXPR`EvalSetUnaryExpr\,(opr, arg\text{-}v)$,
.14 SEQHEAD,
.15 SEQTAIL,
.16 SEQLEN,
.17 SEQELEMS,
.18 SEQINDICES,
.19 SEQDISTRCONC $\rightarrow EXPR`EvalSeqUnaryExpr\,(opr, arg\text{-}v)$,
.20 MAPDOM,
.21 MAPRNG,
.22 MAPDISTRMERGE $\rightarrow EXPR`EvalMapUnaryExpr\,(opr, arg\text{-}v)$,
.23 MAPINVERSE $\rightarrow EXPR`EvalMapInverseExpr\,(arg\text{-}v)$,
.24 others $\rightarrow$ undefined
.25 end in
.26 $STKM`Push(res\text{-}v)$
.27 pre $STKM`LenES\,(1) \wedge$
.28 let $[arg\text{-}v] = STKM`GetES\,(1)$ in
.29 $SEM`IsSemVal\,(arg\text{-}v)$;

Binary Expressions

```
667.0 EvalBINOP : AS`BinaryOp -o-> ()
  .1  EvalBINOP (opr) ≜
  .2    let [left-v, right-v] = STKM`Pop (2) in
  .3    let res-v =
  .4          cases opr :
  .5            EQ,
  .6            NE,
  .7            EQUIV → EXPR`EvalEqNeBinaryExpr (left-v, opr, right-v),
  .8            NUMPLUS,
  .9            NUMMINUS,
  .10           NUMMULT,
  .11           NUMDIV,
  .12           NUMREM,
  .13           INTDIV,
  .14           NUMLT,
  .15           NUMLE,
  .16           NUMGT,
  .17           NUMGE,
  .18           NUMMOD → EXPR`EvalNumBinaryExpr (left-v, opr, right-v),
  .19           INSET,
  .20           NOTINSET,
  .21           SETUNION,
  .22           SETINTERSECT,
  .23           SETMINUS,
  .24           SUBSET,
  .25           PROPERSUBSET → EXPR`EvalSetBinaryExpr (left-v, opr, right-v),
  .26           SEQCONC → EXPR`EvalSeqBinaryExpr (left-v, opr, right-v),
  .27           MAPMERGE,
  .28           MAPDOMRESTTO,
  .29           MAPDOMRESTBY,
  .30           MAPRNGRESTTO,
  .31           MAPRNGRESTBY → EXPR`EvalMapBinaryExpr (left-v, opr, right-v),
  .32           COMPOSE → EXPR`EvalComposeExpr (left-v, right-v),
  .33           NUMEXP → EXPR`EvalIterateExpr (left-v, right-v),
  .34           others → undefined
  .35         end in
  .36     STKM`Push(res-v)
  .37 pre  STKM`LenES (2) ∧
  .38      let [left-v, right-v] = STKM`GetES (2) in
  .39      SEM`IsSemVal (left-v) ∧ SEM`IsSemVal (right-v) ;
```

Appending sequence values

668.0 $\mathit{ExeAPPENDSEQ} : () \xrightarrow{o} ()$
.1 $\mathit{ExeAPPENDSEQ}\,() \triangleq$
.2 $\quad$ let $[\mathit{seqval}, \mathit{val}] = \mathit{STKM}`\mathit{Pop}\,(2)$ in
.3 $\quad \mathit{STKM}`\mathit{Push}(\text{mk-}\mathit{SEM}`\mathit{SEQ}\,(\mathit{seqval}.v \curvearrowright [\mathit{val}]))$
.4 pre $\mathit{STKM}`\mathit{LenES}\,(2) \wedge$
.5 $\qquad$ let $[v, \mathit{seqv}] = \mathit{STKM}`\mathit{GetES}\,(2)$ in
.6 $\qquad \mathit{SEM}`\mathit{IsSemVal}\,(v) \wedge \text{is-}\mathit{SEM}`\mathit{SEQ}\,(\mathit{seqv})$;

Adding elements to set values

669.0 $\mathit{ExeADDSET} : () \xrightarrow{o} ()$
.1 $\mathit{ExeADDSET}\,() \triangleq$
.2 $\quad$ let $[\mathit{setval}, \mathit{val}] = \mathit{STKM}`\mathit{Pop}\,(2)$ in
.3 $\quad \mathit{STKM}`\mathit{Push}(\text{mk-}\mathit{SEM}`\mathit{SET}\,(\mathit{setval}.v \cup \{\mathit{val}\}))$
.4 pre $\mathit{STKM}`\mathit{LenES}\,(2) \wedge$
.5 $\qquad$ let $[v, \mathit{setv}] = \mathit{STKM}`\mathit{GetES}\,(2)$ in
.6 $\qquad \mathit{SEM}`\mathit{IsSemVal}\,(v) \wedge \text{is-}\mathit{SEM}`\mathit{SET}\,(\mathit{setv})$;

Appending mapping values

670.0 $\mathit{ExeAPPENDMAP} : () \xrightarrow{o} ()$
.1 $\mathit{ExeAPPENDMAP}\,() \triangleq$
.2 $\quad$ let $[\mathit{mapval}, \mathit{domval}, \mathit{rngval}] = \mathit{STKM}`\mathit{Pop}\,(3)$ in
.3 $\quad$ if is-$\mathit{SEM}`\mathit{MAP}\,(\mathit{mapval}) \wedge$
.4 $\qquad (\mathit{domval} \notin \text{dom } \mathit{mapval}.v \vee \mathit{mapval}.v\,(\mathit{domval}) = \mathit{rngval})$
.5 $\quad$ then $\mathit{STKM}`\mathit{Push}(\text{mk-}\mathit{SEM}`\mathit{MAP}\,(\mathit{mapval}.v \uplus \{\mathit{domval} \mapsto \mathit{rngval}\}))$
.6 $\quad$ else $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{DUPLICATES\text{-}NOT\text{-}EQUAL}, \mathit{mapval}, \text{nil}, [])$
.7 pre $\mathit{STKM}`\mathit{LenES}\,(3) \wedge$
.8 $\qquad$ let $[\mathit{rngval}, \mathit{domval}, \mathit{mapval}] = \mathit{STKM}`\mathit{GetES}\,(3)$ in
.9 $\qquad \mathit{SEM}`\mathit{IsSemVal}\,(\mathit{rngval}) \wedge$
.10 $\qquad \mathit{SEM}`\mathit{IsSemVal}\,(\mathit{domval}) \wedge$
.11 $\qquad$ is-$\mathit{SEM}`\mathit{MAP}\,(\mathit{mapval})$;

Selecting an element from a set

671.0 $\mathit{ExeSELELEM} : () \xrightarrow{o} ()$
.1 $\mathit{ExeSELELEM}\,() \triangleq$
.2 $\quad$ let $[\mathit{setval}] = \mathit{STKM}`\mathit{Pop}\,(1)$ in
.3 $\quad$ if $\neg$ is-$\mathit{SEM}`\mathit{SET}\,(\mathit{setval})$
.4 $\quad$ then $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{SET\text{-}EXPECTED}, \mathit{setval}, \text{nil}, [])$

.5 else let $e \in setval.v$ in
.6 $(STKM`Push($mk-$SEM`SET\ (setval.v \setminus \{e\}))$;
.7 $STKM`Push(e)\ )$
.8 pre $STKM`LenES\ (1) \wedge$
.9 let $[setval] = STKM`GetES\ (1)$ in
.10 $setval.v \neq \{\}$;

Selecting an element from a sequence

672.0 $ExeSELSEQELEM : [\mathrm{REVERSE}] \xrightarrow{o} ()$
.1 $ExeSELSEQELEM\ (direction) \triangleq$
.2 let $[seqval] = STKM`Pop\ (1)$ in
.3 if $\neg$ is-$SEM`SEQ\ (seqval)$
.4 then $RTERR`Error(RTERR`SEQ\text{-}EXPECTED, seqval,$ nil $, [])$
.5 else let $l = seqval.v,$
.6 mk-$(e, rest) =$ if $direction = \mathrm{REVERSE}$
.7 then mk-$(l\ ($len $l), l\ (1, \ldots,$ len $l - 1))$
.8 else mk-(hd l, tl l) in
.9 $(STKM`Push($mk-$SEM`SEQ\ (rest))$;
.10 $STKM`Push(e)\ )$
.11 pre $STKM`LenES\ (1) \wedge$
.12 let $[seqval] = STKM`GetES\ (1)$ in
.13 $seqval.v \neq []$;

Creating a set range value

673.0 $ExeSETRNG : () \xrightarrow{o} ()$
.1 $ExeSETRNG\ () \triangleq$
.2 let $[lbval, ubval] = STKM`Pop\ (2)$ in
.3 $STKM`Push(EXPR`EvalSetRangeExpr\ (lbval, ubval))$
.4 pre $STKM`LenES\ (2) \wedge$
.5 let $[v1, v2] = STKM`GetES\ (2)$ in
.6 $SEM`IsSemVal\ (v1) \wedge SEM`IsSemVal\ (v2)$;

674.0 $ExeSUBSEQ : () \xrightarrow{o} ()$
.1 $ExeSUBSEQ\ () \triangleq$
.2 let $[seqval, lbval, ubval] = STKM`Pop\ (3)$ in
.3 $STKM`Push(EXPR`EvalSubSequenceExpr\ (seqval, lbval, ubval))$
.4 pre $STKM`LenES\ (3) \wedge$
.5 let $[v1, v2, v3] = STKM`GetES\ (3)$ in
.6 $SEM`IsSemVal\ (v1) \wedge SEM`IsSemVal\ (v2) \wedge SEM`IsSemVal\ (v3)$;

675.0 $ExeRECCONS : AS`Name \times \mathbb{N} \xrightarrow{o} ()$
.1 $ExeRECCONS\,(tag, length) \triangleq$
.2 $\quad$ let $val\text{-}l = STKM`Pop\,(length)$ in
.3 $\quad STKM`Push(EXPR`EvalRecordConstructorExpr\,(tag, val\text{-}l))$
.4 pre $STKM`LenES\,(length) \wedge$
.5 $\quad$ let $v\text{-}l = STKM`GetES\,(length)$ in
.6 $\quad \forall\, v \in$ elems $v\text{-}l \cdot SEM`IsSemVal\,(v)$;

676.0 $ExeRECMOD : \mathbb{N} \xrightarrow{o} ()$
.1 $ExeRECMOD\,(length) \triangleq$
.2 $\quad$ let $val\text{-}l1 = STKM`Pop\,(length)$ in
.3 $\quad$ let $fid\text{-}l = [val\text{-}l1\,(i) \mid i \in$ inds $val\text{-}l1 \cdot i$ mod $2 = 1]$,
.4 $\qquad val\text{-}l = [val\text{-}l1\,(i) \mid i \in$ inds $val\text{-}l1 \cdot i$ mod $2 = 0]$,
.5 $\qquad [rec] = STKM`Pop\,(1)$ in
.6 $\quad STKM`Push(EXPR`EvalRecordModifierExpr\,(rec, fid\text{-}l, val\text{-}l))$
.7 pre $STKM`LenES\,(length + 1) \wedge$
.8 $\quad$ let $[rec] \curvearrowright v\text{-}l = STKM`GetES\,(length + 1)$ in
.9 $\quad SEM`IsSemVal\,(rec) \wedge$
.10 $\quad \forall\, i \in$ inds $v\text{-}l \cdot$ if i mod $2 = 0$
.11 $\qquad\qquad$ then $SEM`IsSemVal\,(v\text{-}l\,(i))$
.12 $\qquad\qquad$ else is-$AS`Name\,(v\text{-}l\,(i))$;

677.0 $ExeFIELDSEL : () \xrightarrow{o} ()$
.1 $ExeFIELDSEL\,() \triangleq$
.2 $\quad$ let $[recval, field] = STKM`Pop\,(2)$ in
.3 $\quad STKM`Push(EXPR`EvalFieldSelectExpr\,(recval, field))$
.4 pre $STKM`LenES\,(2) \wedge$
.5 $\quad$ let $[name, recval] = STKM`GetES\,(2)$ in
.6 $\quad SEM`IsSemVal\,(recval) \wedge$
.7 $\quad$ (is-$AS`Name\,(name) \vee$ is-$AS`FctTypeInstExpr\,(name))$;

678.0 $ExeISEMPTYSET : \mathbb{N} \times [\mathrm{SEM}] \xrightarrow{o} ()$
.1 $ExeISEMPTYSET\,(n, tp) \triangleq$
.2 $\quad$ let $val\text{-}l = STKM`GetES\,(n)$,
.3 $\qquad elm = val\text{-}l\,($len $val\text{-}l)$ in
.4 $\quad STKM`Push($mk-$SEM`BOOL\,(tp = \mathrm{SEM} \wedge elm =$ mk-$SEM`SET\,(\{\}) \vee$
.5 $\qquad\qquad tp \neq \mathrm{SEM} \wedge elm = \{\}))$;

679.0 $\mathit{ExeISEMPTYSEQ} : \mathbb{N} \times [\mathrm{SEM}] \xrightarrow{o} ()$

.1 $\mathit{ExeISEMPTYSEQ}\,(n, \mathit{sem}) \triangleq$

.2 let $\mathit{val\text{-}l} = \mathit{STKM}`\mathit{GetES}\,(n)$ in

.3 $\mathit{STKM}`\mathit{Push}(\text{mk-}\mathit{SEM}`\mathit{BOOL}\,(\mathit{sem} = \mathrm{SEM} \land \mathit{val\text{-}l}\,(n) = \text{mk-}\mathit{SEM}`\mathit{SEQ}\,([\,]) \lor$

.4 $\mathit{sem} \neq \mathrm{SEM} \land \mathit{val\text{-}l}\,(n) = [\,]))$;

Construting Token values

680.0 $\mathit{ExeTOKENVAL} : () \xrightarrow{o} ()$

.1 $\mathit{ExeTOKENVAL}\,() \triangleq$

.2 let $[\mathit{val}] = \mathit{STKM}`\mathit{Pop}\,(1)$ in

.3 $\mathit{STKM}`\mathit{Push}(\text{mk-}\mathit{SEM}`\mathit{TOKEN}\,(\mathit{val}))$

.4 pre $\mathit{STKM}`\mathit{LenES}\,(1) \land$

.5 $\mathit{SEM}`\mathit{IsSemVal}\,(\text{hd } \mathit{STKM}`\mathit{GetES}\,(1))$;

Selecting an Index from a Tuple

681.0 $\mathit{ExeTUPSEL} : \mathbb{R} \xrightarrow{o} ()$

.1 $\mathit{ExeTUPSEL}\,(\mathit{index}) \triangleq$

.2 let $[\mathit{tupval}] = \mathit{STKM}`\mathit{Pop}\,(1)$ in

.3 if $\neg\,\text{is-}\mathit{SEM}`\mathit{TUPLE}\,(\mathit{tupval})$

.4 then $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{TUPLE\text{-}EXPECTED}, \mathit{tupval}, \text{nil}\,, [\,])$

.5 elseif $\mathit{index} \in \text{inds } \mathit{tupval.v}$

.6 then $\mathit{STKM}`\mathit{Push}((\mathit{tupval.v})\,(\mathit{index}))$

.7 else $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{TUPLE\text{-}OUTSIDE\text{-}INDEX}, \mathit{tupval}, \text{nil}\,, [\,])$

.8 pre $\mathit{STKM}`\mathit{LenES}\,(1) \land$

.9 $\mathit{SEM}`\mathit{IsSemVal}\,(\text{hd } \mathit{STKM}`\mathit{GetES}\,(1))$;

Type Judgements

682.0 $\mathit{ExeTYPEJUDGE} : \mathit{AS}`\mathit{Type} \xrightarrow{o} ()$

.1 $\mathit{ExeTYPEJUDGE}\,(\mathit{tp}) \triangleq$

.2 let $[\mathit{val}] = \mathit{STKM}`\mathit{Pop}\,(1)$ in

.3 $(\mathit{STATE}`\mathit{SetTypeJudgement}()$;

.4 $\mathit{STKM}`\mathit{Push}(\text{mk-}\mathit{SEM}`\mathit{BOOL}\,(\mathit{STATE}`\mathit{RealSubType}\,(\mathit{val}, \mathit{tp}, \text{true})))$;

.5 $\mathit{STATE}`\mathit{UnsetTypeJudgement}()\,)$;

Instatiating Polymorphic Functions

683.0 $ExePOLYTYPEINST : AS`Name \xrightarrow{o} ()$

.1 $ExePOLYTYPEINST\,(usedclass) \triangleq$

.2 let $[fn] = STKM`Pop\,(1)$ in

.3 if is-$SEM`CompExplFN\,(fn)$

.4 then let $fnL = [$let $fnval = fn.fl\,(i)$ in

.5 $\mu\,(fnval,$

.6

.7 $tp \mapsto UpdateTypeInfo\,(fnval.tp, usedclass))\mid$

.8 $i \in$ inds $fn.fl]$ in

.9 $STKM`Push(\mu\,(fn, fl \mapsto fnL))$

.10 else $STKM`Push(fn)$

.11 pre $STKM`LenES\,(1) \wedge$

.12 let $[fn] = STKM`GetES\,(1)$ in

.13 $SEM`IsSemVal\,(fn)$

functions

684.0 $ExtendTypeInfo : SEM`POLY \times AS`Name \to SEM`POLY$

.1 $ExtendTypeInfo\,(polyfn, usedclass) \triangleq$

.2 if is-$SEM`ImplPOLY\,(polyfn)$

.3 then $polyfn$

.4 else $\mu\,(polyfn, tp \mapsto UpdateTypeInfo\,(polyfn.tp, usedclass));$

685.0
```
UpdateTypeInfo : (AS`Type | AS`AllType) × AS`Name → AS`Type | AS`AllType
UpdateTypeInfo (tp, usedclass) △
  cases tp :
    mk-AS`BasicType (-, -),
    mk-AS`QuoteType (-, -),
    mk-AS`TypeVar (-, -),
    mk-AS`AllType (),
    mk-AS`CompositeType (-, -, -) →
        tp,
    mk-AS`BracketedType (t, ci) →
        mk-AS`BracketedType (UpdateTypeInfo (t, usedclass), ci),
    mk-AS`UnionType (tL, ci) →
        mk-AS`UnionType ([UpdateTypeInfo (tL (i), usedclass) |
                                  i ∈ inds tL],
                         ci),
    mk-AS`ProductType (tL, ci) →
        mk-AS`ProductType ([UpdateTypeInfo (tL (i), usedclass) |
                                  i ∈ inds tL],
                           ci),
    mk-AS`OptionalType (t, ci) →
        mk-AS`OptionalType (UpdateTypeInfo (t, usedclass), ci),
    mk-AS`SetType (t, ci) →
        mk-AS`SetType (UpdateTypeInfo (t, usedclass), ci),
    mk-AS`Seq0Type (t, ci) →
        mk-AS`Seq0Type (UpdateTypeInfo (t, usedclass), ci),
    mk-AS`Seq1Type (t, ci) →
        mk-AS`Seq1Type (UpdateTypeInfo (t, usedclass), ci),
    mk-AS`GeneralMapType (dt, rt, ci) →
        mk-AS`GeneralMapType (UpdateTypeInfo (dt, usedclass),
                              UpdateTypeInfo (dt, usedclass),
                              ci),
    mk-AS`InjectiveMapType (dt, rt, ci) →
        mk-AS`InjectiveMapType (UpdateTypeInfo (dt, usedclass),
                                UpdateTypeInfo (dt, usedclass),
                                ci),
    mk-AS`TypeName (nm, ci) →
        let newnm = if len nm.ids = 1 ∧
                       ¬ nm ∈ dom STATE`GetClasses ()
                    then AUX`ConstructDoubleName (usedclass, nm)
                    else nm in
        mk-AS`TypeName (newnm, ci),
    mk-AS`PartialFnType (tL, t, ci) →
        let newtL = [UpdateTypeInfo (tL (i), usedclass) |
                          i ∈ inds tL],
            newt = UpdateTypeInfo (t, usedclass) in
        mk-AS`PartialFnType (newtL, newt, ci),
    mk-AS`TotalFnType (tL, t, ci) →
        let newtL = [UpdateTypeInfo (tL (i), usedclass) |
                          i ∈ inds tL],
            newt = UpdateTypeInfo (t, usedclass) in
        mk-AS`TotalFnType (newtL, newt, ci)
  end
```

operations

686.0 $ExePOLYINST : AS`Type^* \xrightarrow{o} ()$
.1 $ExePOLYINST\ (inst) \triangleq$
.2 let $[polyfn] = STKM`Pop\ (1)$ in
.3 $STKM`Push(STATE`EvalFctTypeInstExpr\ (polyfn, inst))$
.4 pre $STKM`LenES\ (1) \wedge$
.5 let $[polyfn] = STKM`GetES\ (1)$ in
.6 is-$SEM`ExplPOLY\ (polyfn) \vee$ is-$SEM`ImplPOLY\ (polyfn)$;

Appending tuple values

687.0 $ExeAPPENDTUP : () \xrightarrow{o} ()$
.1 $ExeAPPENDTUP\ () \triangleq$
.2 let $[tupval, val] = STKM`Pop\ (2)$ in
.3 $STKM`Push($mk-$SEM`TUPLE\ (tupval.v \curvearrowright [val]))$
.4 pre $STKM`LenES\ (2) \wedge$
.5 let $[v, tupv] = STKM`GetES\ (2)$ in
.6 $SEM`IsSemVal\ (v) \wedge$ is-$SEM`TUPLE\ (tupv)$;

Checking Bindings for Sequence Comprehensions

688.0 $ExeSEQCOMPBIND : () \xrightarrow{o} ()$
.1 $ExeSEQCOMPBIND\ () \triangleq$
.2 let $[pat, set\text{-}v] = STKM`Pop\ (2)$ in
.3 if $\neg$ is-$STKM`PatternName\ (pat)$
.4 then $RTERR`Error(RTERR`PAT\text{-}NAME\text{-}IN\text{-}SEQCOMP,$ nil , nil , $[])$
.5 elseif is-$SEM`SET\ (set\text{-}v)$
.6 then if $\forall\ elm \in set\text{-}v.v \cdot AUX`IsInt\ (elm)$
.7 then $(STKM`Push(pat)$;
.8 $STKM`Push(AUX`SetToSeq\ (set\text{-}v.v)))$
.9 else $RTERR`Error(RTERR`NUMERIC\text{-}SET, set\text{-}v,$ nil , $[])$
.10 else $RTERR`Error(RTERR`SET\text{-}EXPECTED, set\text{-}v,$ nil , $[])$
.11 pre $STKM`LenES\ (2) \wedge$
.12 let $[set\text{-}v, pat] = STKM`GetES\ (2)$ in
.13 $SEM`IsSemVal\ (set\text{-}v) \wedge STKM`IsPat\ (pat)$;

Sequence and Map Override Instruction

689.0 $ExeSEQMAPOVER : () \xrightarrow{o} ()$

.1 $ExeSEQMAPOVER\,() \triangleq$

.2 $\quad \mathsf{let}\ [seqmap\text{-}v, map\text{-}v] = STKM`Pop\,(2)\ \mathsf{in}$

.3 $\quad STKM`Push(EXPR`EvalSeqModifyMapOverrideExpr\,(seqmap\text{-}v, map\text{-}v))$

.4 $\mathsf{pre}\ \ STKM`LenES\,(2) \wedge$

.5 $\quad\ \mathsf{let}\ [map\text{-}v, seqmap\text{-}v] = STKM`GetES\,(2)\ \mathsf{in}$

.6 $\quad\ SEM`IsSemVal\,(map\text{-}v) \wedge SEM`IsSemVal\,(seqmap\text{-}v)\ ;$

Pattern Instructions

690.0 $ExeEnumAppend : () \xrightarrow{o} ()$

.1 $ExeEnumAppend\,() \triangleq$

.2 $\quad \mathsf{let}\ [enumpat, newelem] = STKM`Pop\,(2)\ \mathsf{in}$

.3 $\quad STKM`Push(\mu\,(enumpat, els \mapsto enumpat.els \frown [newelem]))$

.4 $\mathsf{pre}\ \ STKM`LenES\,(2) \wedge$

.5 $\quad\ \mathsf{let}\ [right\text{-}v, left\text{-}v] = STKM`GetES\,(2)\ \mathsf{in}$

.6 $\quad\ (\mathsf{is}\text{-}STKM`SetEnumPattern\,(left\text{-}v) \vee$

.7 $\quad\ \ \mathsf{is}\text{-}STKM`SeqEnumPattern\,(left\text{-}v) \vee$

.8 $\quad\ \ \mathsf{is}\text{-}STKM`TuplePattern\,(left\text{-}v)) \wedge$

.9 $\quad\ (STKM`IsPat\,(right\text{-}v) \vee SEM`IsSemVal\,(right\text{-}v))\ ;$

691.0 $ExeSetUnion : CI`ContextId \xrightarrow{o} ()$

.1 $ExeSetUnion\,(cid) \triangleq$

.2 $\quad \mathsf{let}\ [leftpat, rightpat] = STKM`Pop\,(2)\ \mathsf{in}$

.3 $\quad STKM`Push(\mathsf{mk}\text{-}STKM`SetUnionPattern\,(leftpat, rightpat, cid))$

.4 $\mathsf{pre}\ \ STKM`LenES\,(2) \wedge$

.5 $\quad\ \mathsf{let}\ [left\text{-}v, right\text{-}v] = STKM`GetES\,(2)\ \mathsf{in}$

.6 $\quad\ (STKM`IsPat\,(left\text{-}v) \vee SEM`IsSemVal\,(left\text{-}v)) \wedge$

.7 $\quad\ (STKM`IsPat\,(right\text{-}v) \vee SEM`IsSemVal\,(right\text{-}v))\ ;$

692.0 $ExeSeqConc : CI`ContextId \xrightarrow{o} ()$

.1 $ExeSeqConc\,(cid) \triangleq$

.2 $\quad \mathsf{let}\ [leftpat, rightpat] = STKM`Pop\,(2)\ \mathsf{in}$

.3 $\quad STKM`Push(\mathsf{mk}\text{-}STKM`SeqConcPattern\,(leftpat, rightpat, cid))$

.4 $\mathsf{pre}\ \ STKM`LenES\,(2) \wedge$

.5 $\quad\ \mathsf{let}\ [left\text{-}v, right\text{-}v] = STKM`GetES\,(2)\ \mathsf{in}$

.6 $\quad\ (STKM`IsPat\,(left\text{-}v) \vee SEM`IsSemVal\,(left\text{-}v)) \wedge$

.7 $\quad\ (STKM`IsPat\,(right\text{-}v) \vee SEM`IsSemVal\,(right\text{-}v))\ ;$

693.0 $\mathit{ExeFieldsAppend} : () \xrightarrow{o} ()$
.1 $\mathit{ExeFieldsAppend}\,() \triangleq$
.2 let $[\mathit{recortuppat}, \mathit{newelem}] = \mathit{STKM}`\mathit{Pop}\,(2)$ in
.3 $\mathit{STKM}`\mathit{Push}(\mu\,(\mathit{recortuppat}, \mathit{fields} \mapsto \mathit{recortuppat}.\mathit{fields} \curvearrowright [\mathit{newelem}]))$
.4 pre $\mathit{STKM}`\mathit{LenES}\,(2) \wedge$
.5 let $[\mathit{right\text{-}v}, \mathit{left\text{-}v}] = \mathit{STKM}`\mathit{GetES}\,(2)$ in
.6 (is-$\mathit{STKM}`\mathit{SetEnumPattern}\,(\mathit{left\text{-}v}) \vee$
.7 is-$\mathit{STKM}`\mathit{SeqEnumPattern}\,(\mathit{left\text{-}v}) \vee$
.8 is-$\mathit{STKM}`\mathit{TuplePattern}\,(\mathit{left\text{-}v}) \vee$
.9 is-$\mathit{STKM}`\mathit{RecordPattern}\,(\mathit{left\text{-}v})) \wedge$
.10 $(\mathit{STKM}`\mathit{IsPat}\,(\mathit{right\text{-}v}) \vee \mathit{SEM}`\mathit{IsSemVal}\,(\mathit{right\text{-}v}))$;

694.0 $\mathit{ExeFREF} : \mathit{CI}`\mathit{ContextId} \xrightarrow{o} ()$
.1 $\mathit{ExeFREF}\,(\mathit{ci}) \triangleq$
.2 let $[\mathit{sd}, \mathit{nm}] = \mathit{STKM}`\mathit{Pop}\,(2)$ in
.3 $\mathit{STKM}`\mathit{Push}($mk-$\mathit{STKM}`\mathit{FieldRef}\,(\mathit{sd}, \mathit{nm}, \mathit{ci}))$
.4 pre $\mathit{STKM}`\mathit{LenES}\,(2) \wedge$
.5 let $[\mathit{left\text{-}v}, \mathit{right\text{-}v}] = \mathit{STKM}`\mathit{GetES}\,(2)$ in
.6 $\mathit{STKM}`\mathit{IsSD}\,(\mathit{right\text{-}v}) \wedge$ is-$\mathit{AS}`\mathit{Name}\,(\mathit{left\text{-}v})$;

695.0 $\mathit{ExeMOSREF} : \mathit{CI}`\mathit{ContextId} \xrightarrow{o} ()$
.1 $\mathit{ExeMOSREF}\,(\mathit{ci}) \triangleq$
.2 let $[\mathit{sd}, \mathit{val}] = \mathit{STKM}`\mathit{Pop}\,(2)$ in
.3 $\mathit{STKM}`\mathit{Push}($mk-$\mathit{STKM}`\mathit{MapOrSeqRef}\,(\mathit{sd}, \mathit{val}, \mathit{ci}))$
.4 pre $\mathit{STKM}`\mathit{LenES}\,(2) \wedge$
.5 let $[\mathit{left\text{-}v}, \mathit{right\text{-}v}] = \mathit{STKM}`\mathit{GetES}\,(2)$ in
.6 $\mathit{STKM}`\mathit{IsSD}\,(\mathit{right\text{-}v}) \wedge \mathit{SEM}`\mathit{IsSemVal}\,(\mathit{left\text{-}v})$;

696.0 $\mathit{ExeATOMIC} : \mathbb{N}_1 \xrightarrow{o} ()$
.1 $\mathit{ExeATOMIC}\,(\mathit{no}) \triangleq$
.2 let $\mathit{assignl} = \mathit{STKM}`\mathit{Pop}\,(2 \times \mathit{no})$ in
.3 (for $i = 1$ to len $\mathit{assignl}$ by 2
.4 do let $\mathit{rhs} = \mathit{assignl}\,(i)$,
.5 $\mathit{lhs} = \mathit{assignl}\,(i + 1)$ in
.6 if is-$\mathit{SEM}`\mathit{CONT}\,(\mathit{rhs})$
.7 then $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{OP\text{-}RETURNED\text{-}CONT}, \mathit{rhs},$ nil $, [])$
.8 else $\mathit{STATE}`\mathit{EvalStateDesignator}(\mathit{lhs}, \mathit{rhs})$;
.9 $\mathit{InvOK}()$)
.10 pre $\mathit{STKM}`\mathit{LenES}\,(2 \times \mathit{no})$;

697.0 $ExeASSIGNSD : () \xrightarrow{o} ()$
.1 $ExeASSIGNSD\,() \triangleq$
.2 let $[rhs, lhs] = STKM`Pop\,(2)$ in
.3 $(STATE`EvalStateDesignator(lhs, rhs)$;
.4 $InvOK()\,)$
.5 pre $STKM`LenES\,(2) \land$
.6 let $[left\text{-}v, right\text{-}v] = STKM`GetES\,(2)$ in
.7 $STKM`IsSD\,(left\text{-}v) \land SEM`IsSemVal\,(right\text{-}v)$;

698.0 $InvOK : () \xrightarrow{o} ()$
.1 $InvOK\,() \triangleq$
.2 let $invnotok = SETTINGS`INV\,() \land \neg\, STATE`CheckGlobInv\,()$ in
.3 if $invnotok$
.4 then $RTERR`Error(RTERR`INST\text{-}INV\text{-}BROKEN,$ nil , nil , $[])$;

699.0 $ExeSEQELEMMATCH : \mathbb{N} \xrightarrow{o} ()$
.1 $ExeSEQELEMMATCH\,(index) \triangleq$
.2 let $elem\text{-}l = STKM`Pop\,(index)$ in
.3 $(BindPat(elem\text{-}l\,(1),$ hd $elem\text{-}l\,(2))$;
.4 $STKM`Push(elem\text{-}l\,(1))$;
.5 $STKM`Push($tl $elem\text{-}l\,(2))$;
.6 for $i = 3$ to $index$
.7 do $STKM`Push(elem\text{-}l\,(i))\,)$
.8 pre $STKM`LenES\,(index) \land$
.9 let $elem\text{-}l = STKM`GetES\,(index)$ in
.10 $STKM`IsPat\,(elem\text{-}l\,(index))$;

700.0 $ExeEmptyBlkEnv : SEM`Permission \xrightarrow{o} ()$
.1 $ExeEmptyBlkEnv\,(permis) \triangleq$
.2 $STKM`PushEmptyBlkEnv(permis)$;

701.0 $ExeMatchAndBind : () \xrightarrow{o} ()$
.1 $ExeMatchAndBind\,() \triangleq$
.2 let $[val, pat] = STKM`Pop\,(2)$ in
.3 $BindPat(pat, val)$
.4 pre $STKM`LenES\,(2) \land$
.5 let $[p, v] = STKM`GetES\,(2)$ in
.6 $SEM`IsSemVal\,(v) \land STKM`IsPat\,(p)$;

702.0 $BindPat : STKM`Pattern \times SEM`VAL \xrightarrow{o} ()$
.1 $BindPat\,(pat, val) \triangleq$
.2 let $env\text{-}s = PAT`PatternMatch\,(pat, val)$ in
.3 if $env\text{-}s = \{\}$
.4 then $RTERR`Error(RTERR`EMPTY\text{-}ENV\text{-}S,$ nil , nil , $[])$
.5 else let $env \in env\text{-}s$ in
.6 $STKM`AddToTopBlkEnv(env)$;

703.0 $ExeCOPYVAL : () \xrightarrow{o} ()$
.1 $ExeCOPYVAL\,() \triangleq$
.2 let $val = STKM`Head\,()$ in
.3 $STKM`Push(val)$;

704.0 $ExeDTC : AS`Type \xrightarrow{o} ()$
.1 $ExeDTC\,(tp) \triangleq$
.2 if $SETTINGS`DTC\,()$
.3 then let $[val] = STKM`GetES\,(1)$ in
.4 if $\neg\, STATE`SubType\,(val, tp)$
.5 then $RTERR`Error(RTERR`TYPE\text{-}INCOMP, val, tp, [])$
.6 pre $STKM`LenES\,(1) \wedge$
.7 let $[val] = STKM`GetES\,(1)$ in
.8 $SEM`IsSemVal\,(val)$;

705.0 $ExeDTCSET : () \xrightarrow{o} ()$
.1 $ExeDTCSET\,() \triangleq$
.2 let [mk-$SEM`SET\,(setval)] = STKM`Pop\,(1)$ in
.3 if $SETTINGS`DTC\,()$
.4 then let $[val] = STKM`GetES\,(1)$ in
.5 if $val \notin setval$
.6 then $RTERR`Error(RTERR`VALUE\text{-}NOT\text{-}IN\text{-}SETBIND, val,$ nil , $[])$
.7 pre $STKM`LenES\,(2) \wedge$
.8 let $[setval, val] = STKM`GetES\,(2)$ in
.9 $SEM`IsSemVal\,(val) \wedge$ is-$SEM`SET\,(setval)$;

706.0 $ExeSIZE : \mathbb{N} \xrightarrow{o} ()$
.1 $ExeSIZE\,(n) \triangleq$
.2 let $[val] = STKM`GetES\,(1)$ in
.3 $STKM`Push($mk-$SEM`BOOL\,($card $val = n))$;

707.0 $\mathit{ExeTRYMATCH} : () \xrightarrow{o} ()$

.1 $\mathit{ExeTRYMATCH}\,() \triangleq$

.2 let $[val, pat] = \mathit{STKM}`\mathit{Pop}\,(2)$ in

.3 let $\mathit{env\text{-}s} = \mathit{PAT}`\mathit{PatternMatch}\,(pat, val)$ in

.4 if $\mathit{env\text{-}s} = \{\}$

.5 then $\mathit{STKM}`\mathit{Push}(\textsf{mk-}\mathit{SEM}`\mathit{BOOL}\,(\textsf{false}))$

.6 else let $env \in \mathit{env\text{-}s}$ in

.7 $(\mathit{STKM}`\mathit{AddToTopBlkEnv}(env)\,;$

.8 $\mathit{STKM}`\mathit{Push}(\mathit{env\text{-}s} \setminus \{env\})\,;$

.9 $\mathit{STKM}`\mathit{Push}(\textsf{mk-}\mathit{SEM}`\mathit{BOOL}\,(\textsf{true}))\,)$

.10 pre $\mathit{STKM}`\mathit{LenES}\,(2) \wedge$

.11 let $[p, v] = \mathit{STKM}`\mathit{GetES}\,(2)$ in

.12 $\mathit{SEM}`\mathit{IsSemVal}\,(v) \wedge \mathit{STKM}`\mathit{IsPat}\,(p)$;

708.0 $\mathit{ExeTRYANYMATCH} : () \xrightarrow{o} ()$

.1 $\mathit{ExeTRYANYMATCH}\,() \triangleq$

.2 let $[val, pat] = \mathit{STKM}`\mathit{Pop}\,(2)$ in

.3 let $\mathit{env\text{-}s} = \mathit{PAT}`\mathit{PatternMatch}\,(pat, val)$ in

.4 if $\mathit{env\text{-}s} = \{\}$

.5 then $\mathit{STKM}`\mathit{Push}(\textsf{mk-}\mathit{SEM}`\mathit{BOOL}\,(\textsf{false}))$

.6 else let $env \in \mathit{env\text{-}s}$ in

.7 $(\mathit{STKM}`\mathit{AddToTopBlkEnv}(env)\,;$

.8 $\mathit{STKM}`\mathit{Push}(\textsf{mk-}\mathit{SEM}`\mathit{BOOL}\,(\textsf{true}))\,)$

.9 pre $\mathit{STKM}`\mathit{LenES}\,(2) \wedge$

.10 let $[p, v] = \mathit{STKM}`\mathit{GetES}\,(2)$ in

.11 $\mathit{SEM}`\mathit{IsSemVal}\,(v) \wedge \mathit{STKM}`\mathit{IsPat}\,(p)$;

709.0 $\mathit{ExeMATCHVAL} : () \xrightarrow{o} ()$

.1 $\mathit{ExeMATCHVAL}\,() \triangleq$

.2 let $[val] = \mathit{STKM}`\mathit{Pop}\,(1)$ in

.3 $\mathit{STKM}`\mathit{Push}(\textsf{mk-}\mathit{STKM}`\mathit{MatchVal}\,(val))$;

710.0 $\mathit{ExeCLOSENV} : \mathit{SEM}`\mathit{BlkEnv} \times \mathit{AS}`\mathit{Name} \xrightarrow{m} (\mathit{AS}`\mathit{Expr} \mid \mathrm{NOTYETSPEC}) \xrightarrow{o} ()$

.1 $\mathit{ExeCLOSENV}\,(blkenv, bodym) \triangleq$

.2 let $\mathit{closenv\text{-}m} = \{id \mapsto$ if $bodym\,(id) = \mathrm{NOTYETSPEC}$

.3 then $\{\mapsto\}$

.4 else $\mathit{FREE}`\mathit{FreeMapToBlkEnv}\,(\mathit{FREE}`\mathit{FreeInExpr}\,(bodym\,(id), \{\}))\mid$

.5 $id \in$ dom $bodym\}$ in

.6 if $\mathit{closenv\text{-}m} = \{\mapsto\}$

.7 then $\mathit{STKM}`\mathit{AddToTopBlkEnv}(blkenv)$

.8 else let $newenv = \mu\,(blkenv,$
.9
.10 $id\text{-}m \mapsto \{id \mapsto$ let mk-$SEM`ValTp\,(v, t) = blkenv.id\text{-}m\,(id)$ in
.11 let $val = SEM`UpdateClosEnv\,(v,$
.12 $closenv\text{-}m\,(id))$ in
.13 mk-$SEM`ValTp\,(val, t)\mid$
.14 $id \in$ dom $blkenv.id\text{-}m \cap$ dom $closenv\text{-}m\})$ in
.15 $STKM`AddToTopBlkEnv(newenv)$;

711.0 $ExeUPDATECLOSENV : AS`Expr \xrightarrow{o} ()$
.1 $ExeUPDATECLOSENV\,(expr) \triangleq$
.2 let $[fnval] = STKM`Pop\,(1),$
.3 $blkenv = FREE`FreeMapToBlkEnv\,(FREE`FreeInExpr\,(expr, \{\}))$ in
.4 $STKM`Push(SEM`UpdateClosEnv\,(fnval, blkenv))$;

712.0 $ExeVERIFYINDEXARGS : () \xrightarrow{o} ()$
.1 $ExeVERIFYINDEXARGS\,() \triangleq$
.2 let $[lb\text{-}v, ub\text{-}v, step\text{-}v] = STKM`GetES\,(3)$ in
.3 if $\neg$ is-$SEM`NUM\,(lb\text{-}v)$
.4 then $RTERR`Error(RTERR`LOWER\text{-}BOUND\text{-}NOT\text{-}A\text{-}NUMBER, lb\text{-}v,$ nil $, [])$
.5 elseif $\neg$ is-$SEM`NUM\,(ub\text{-}v)$
.6 then $RTERR`Error(RTERR`UPPER\text{-}BOUND\text{-}NOT\text{-}A\text{-}NUMBER, ub\text{-}v,$ nil $, [])$
.7 elseif $\neg$ is-$SEM`NUM\,(step\text{-}v)$
.8 then $RTERR`Error(RTERR`STEP\text{-}NOT\text{-}A\text{-}NUMBER, step\text{-}v,$ nil $, [])$
.9 else let mk-$SEM`NUM\,(lb) = lb\text{-}v,$
.10 mk-$SEM`NUM\,(ub) = ub\text{-}v,$
.11 mk-$SEM`NUM\,(step) = step\text{-}v$ in
.12 if $step = 0$
.13 then $RTERR`Error(RTERR`STEP\text{-}INDEX\text{-}IS\text{-}ZERO, step\text{-}v,$ nil $, [])$
.14 pre $STKM`LenES\,(3) \wedge$
.15 let $[v1, v2, v3] = STKM`GetES\,(3)$ in
.16 $SEM`IsSemVal\,(v1) \wedge SEM`IsSemVal\,(v2) \wedge SEM`IsSemVal\,(v3)$;

The *ExeTESTCOUNTER* and *ExeINCRCOUNTER* operations are used to respectively test whether the next iteration should be carried out and the increment the (step) counter. Both operations are used with an index-for-loop statement.

713.0 $ExeTESTCOUNTER : () \xrightarrow{o} ()$
.1 $ExeTESTCOUNTER\,() \triangleq$
.2 let [-, mk-$SEM`NUM\,(current),$ mk-$SEM`NUM\,(last),$ mk-$SEM`NUM\,(step)] = STKM`GetES\,(4),$
.3 $new = current + step,$
.4 $cont =$ if $step > 0$
.5 then $new \leq last + step$
.6 else $new \geq last + step$ in
.7 $(STKM`Push($mk-$SEM`BOOL\,(cont))\,)$

.8 pre $STKM`LenES\,(3) \wedge$
.9 let [-, $v1, v2, v3$] = $STKM`GetES\,(4)$ in
.10 is-$SEM`NUM\,(v1) \wedge$ is-$SEM`NUM\,(v2) \wedge$ is-$SEM`NUM\,(v3)$;

714.0 $ExeINCRCOUNTER : () \xrightarrow{o} ()$
.1 $ExeINCRCOUNTER\,() \triangleq$
.2 let [mk-$SEM`NUM\,(current), topElm] = STKM`Pop\,(2)$,
.3 [mk-$SEM`NUM\,(last)$, mk-$SEM`NUM\,(step)] = STKM`GetES\,(2)$,
.4 $new = current + step$ in
.5 ($STKM`Push$(mk-$SEM`NUM\,(new)$)) ;
.6 $STKM`Push(topElm)$)
.7 pre $STKM`LenES\,(3) \wedge$
.8 let [-, $v1, v2, v3$] = $STKM`GetES\,(4)$ in
.9 is-$SEM`NUM\,(v1) \wedge$ is-$SEM`NUM\,(v2) \wedge$ is-$SEM`NUM\,(v3)$;

715.0 $ExeNONDETSTMT : () \xrightarrow{o} ()$
.1 $ExeNONDETSTMT\,() \triangleq$
.2 let $[reljumps, stmtval] = STKM`Pop\,(2)$ in
.3 (if $\neg$ is-$SEM`CONT\,(stmtval)$
.4 then $STKM`Push(stmtval)$
.5 elseif $reljumps \neq []$
.6 then ($STKM`IncrPC$(hd $reljumps + 1$) ;
.7 $STKM`Push$(tl $reljumps$))
.8 else $STKM`Push(stmtval)$)
.9 pre $STKM`LenES\,(2)$;

716.0 $ExeRANDOM : () \xrightarrow{o} ()$
.1 $ExeRANDOM\,() \triangleq$
.2 if $SETTINGS`Random\,() \neq -1$
.3 then let $[indices] = STKM`Pop\,(1)$ in
.4 $STKM`Push(Permute\,(indices))$

functions

717.0 $Permute : \mathbb{N}^* \to \mathbb{N}^*$
.1 $Permute\,(indices) \triangleq$
.2 $indices$

The *Permute* function is going to permute the given sequence of indices according to the given seed if the random setting has been swichted on. At the specification level this is not taken into account.

VDM++ Instructions

operations

```
718.0  ExeNEWOBJ : AS`Name × [token] →o ()
   .1  ExeNEWOBJ (name, dlobject) △
   .2    (if SETTINGS`DTC () ∧ STATE`CheckIfAbstractClass (name)
   .3     then RTERR`Error(RTERR`INST-ABS-CL, nil , nil , []) ;
   .4     ExeNEWPOSABSOBJ(name, dlobject) )
   .5  pre  let mk-AS`Name (ids, -) = name in
   .6       len ids = 1 ;
```

```
719.0  ExeNEWPOSABSOBJ : AS`Name × [token] →o ()
   .1  ExeNEWPOSABSOBJ (name, dlclass) △
   .2    (if ¬ STATE`IsAClass (name)
   .3     then RTERR`Error(RTERR`CLNM-NOT-DEFINED, nil , nil , []) ;
   .4     let insstrct = STATE`GetInstInitVal (name),
   .5         tmp-obj = mk-SEM`OBJ (name, insstrct, {↦}),
   .6         tmp-ref = STKM`MakeNewObj (tmp-obj) in
   .7     (STKM`Push(tmp-ref) ;
   .8      STKM`PushCurObj(tmp-ref, nil , nil ) ))
   .9  pre  let mk-AS`Name (ids, -) = name in
  .10       len ids = 1 ;
```

```
720.0  ExeINITCLASS : AS`Name × ℕ →o ()
   .1  ExeINITCLASS (nm, initno) △
   .2    let const-vals ⁀ [obj-ref] = STKM`Pop (initno + 1) in
   .3    (STKM`Push(obj-ref) ;
   .4     STKM`Push(const-vals) ;
   .5     let prog = STATE`LookUpConstructor (nm, const-vals) in
   .6     STKM`PushCS(prog, "Runningconstructorfor"⁀AUX`SingleNameToString (nm), nil , INTERNAL));
```

```
721.0  ExeBINDINSTVAR : AS`Name →o ()
   .1  ExeBINDINSTVAR (nm) △
   .2    let [val] = STKM`Pop (1) in
   .3    STATE`SetInstanceVar(nm, val)
   .4  pre  STKM`LenES (1) ∧
   .5       let [val] = STKM`GetES (1) in
   .6       SEM`IsSemVal (val) ;
```

722.0 $\mathit{ExeNEWCOMPL} : \mathbb{B} \xrightarrow{o} ()$
.1 $\mathit{ExeNEWCOMPL}\,(\mathit{checkinv}) \triangleq$
.2 (if $\mathit{checkinv}$
.3 then if $\mathit{SETTINGS}`\mathit{DTC}\,() \land \neg\, \mathit{STATE}`\mathit{CheckInstanceInvariant}\,()$
.4 then $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{INST\text{-}INV\text{-}BROKEN}, \mathsf{nil}\ , \mathsf{nil}\ , [])$;
.5 $\mathit{STKM}`\mathit{PopCurObj}()$);

723.0 $\mathit{ExePUSHCLNMCUROBJ} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{Name} \xrightarrow{o} ()$
.1 $\mathit{ExePUSHCLNMCUROBJ}\,(\mathit{cl}, \mathit{origcl}) \triangleq$
.2 $\mathit{STKM}`\mathit{PushClNmCurObj}(\mathit{cl}, \mathit{origcl})$;

724.0 $\mathit{ExePOPCLNMCUROBJ} : () \xrightarrow{o} ()$
.1 $\mathit{ExePOPCLNMCUROBJ}\,() \triangleq$
.2 $\mathit{STKM}`\mathit{PopClNmCurObj}()$;

725.0 $\mathit{ExeSELFEXPR} : () \xrightarrow{o} ()$
.1 $\mathit{ExeSELFEXPR}\,() \triangleq$
.2 if $\mathit{STKM}`\mathit{GetObjLLen}\,() = 0$
.3 then $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{NOOBJECT}, \mathsf{nil}\ , \mathsf{nil}\ , [])$
.4 else $\mathit{STKM}`\mathit{Push}(\mathit{STKM}`\mathit{GetCurObjRef}\,())$;

The *ExeISOFCLASS* operation is responsible for checking whether the current element on top of the evaluation stack is of the class identified by *clfn*. If the element is not an object reference the expression evaluates to false, otherwise is evaluated if the class name *nm* is a super class of the object or if the class name is equal to the class name of the object.

726.0 $\mathit{ExeISOFCLASS} : \mathit{AS}`\mathit{Name} \xrightarrow{o} ()$
.1 $\mathit{ExeISOFCLASS}\,(\mathit{clnm}) \triangleq$
.2 let $[\mathit{arg\text{-}v}] = \mathit{STKM}`\mathit{Pop}\,(1)$ in
.3 if $\neg$ is-$\mathit{SEM}`\mathit{OBJ\text{-}Ref}\,(\mathit{arg\text{-}v})$
.4 then $\mathit{STKM}`\mathit{Push}($mk-$\mathit{SEM}`\mathit{BOOL}\,(\mathsf{false}))$
.5 else if $\mathit{clnm} \notin \mathit{STATE}`\mathit{GetAllClassNms}\,()$
.6 then $\mathit{RTERR}`\mathit{Error}(\mathit{RTERR}`\mathit{CLNM\text{-}NOT\text{-}DEFINED}, \mathsf{nil}\ , \mathsf{nil}\ , [])$
.7 else let mk-$\mathit{SEM}`\mathit{OBJ}\,(\mathit{objnm}, \text{-}, \text{-}) = \mathit{STATE}`\mathit{GetSemObjInTab}\,(\mathit{arg\text{-}v})$,
.8 $\mathit{pos\text{-}cls} = \{\mathit{objnm}\} \cup \mathit{STATE}`\mathit{GetAllSupers}\,(\mathit{objnm})$,
.9 $\mathit{found} =$ mk-$\mathit{SEM}`\mathit{BOOL}\,(\mathit{clnm} \in \mathit{pos\text{-}cls})$ in
.10 $\mathit{STKM}`\mathit{Push}(\mathit{found})$;

The operation *ExeISOFBASECLASS* takes a class name *clnm* and assumes that its argument is placed at the top of the evaluation stack. If the argument is not an object reference the expression evaluates to false, otherwise, it is evaluated if the class name *clnm* is a base class of the object.

```
727.0 ExeISOFBASECLASS : AS`Name ⟶ᵒ ()
ExeISOFBASECLASS (clnm) △
  let [arg-v] = STKM`Pop (1) in
  if ¬ is-SEM`OBJ-Ref (arg-v)
  then STKM`Push(mk-SEM`BOOL (false))
  else if clnm ∉ STATE`GetAllClassNms ()
       then RTERR`Error(RTERR`CLNM-NOT-DEFINED, nil , nil , [])
       else let mk-SEM`OBJ (objnm, -, -) = STATE`GetSemObjInTab (arg-v) in
            let mk- (existed, s) = STATE`LookupHchy (clnm),
                allsupers = STATE`GetAllSupers (objnm) ∪ {objnm} in
            if ¬ existed
            then RTERR`Error(RTERR`INTERNAL-ERROR, nil , nil , [])
            else let found = mk-SEM`BOOL (clnm ∈ allsupers ∧
                                              s = {}) in
                 STKM`Push(found) ;
```

The operation *ExeSAMEBASECLASS* takes two value *expr1-v* and *expr2-v* from the evaluation stack. If they are not both object references, the operation returns the semantics value of false, otherwise, it is evaluated if the two object references have any common base classes.

```
728.0 ExeSAMEBASECLASS : () ⟶ᵒ ()
ExeSAMEBASECLASS () △
  let [expr1-v, expr2-v] = STKM`Pop (2) in
  if ¬ is-SEM`OBJ-Ref (expr1-v) ∨ ¬ is-SEM`OBJ-Ref (expr2-v)
  then STKM`Push(mk-SEM`BOOL (false))
  else let mk-SEM`OBJ (objnm1, -, -) = STATE`GetSemObjInTab (expr1-v),
           mk-SEM`OBJ (objnm2, -, -) = STATE`GetSemObjInTab (expr2-v),
           mk- (ok1, -) = STATE`LookupHchy (objnm1),
           mk- (ok2, -) = STATE`LookupHchy (objnm2) in
       if ¬ (ok1 ∧ ok2)
       then RTERR`Error(RTERR`INTERNAL-ERROR, nil , nil , [])
       else let roots1 =
                      {cl | cl ∈
                            STATE`GetAllSupers (objnm1) ∪
                            {objnm1} ·
                                let mk- (-, s) = STATE`LookupHchy (cl) in
                                s = {}},
                roots2 =
                      {cl | cl ∈
                            STATE`GetAllSupers (objnm2) ∪
                            {objnm2} ·
                                let mk- (-, s) = STATE`LookupHchy (cl) in
                                s = {}},
                found = mk-SEM`BOOL ((roots1 ∩ roots2) ≠ {}) in
            STKM`Push(found) ;
```

The operation *ExeSAMECLASS* takes two values *expr1-v* and *expr2-v* from the evaluation stack. If both values are not object references the semantic value of false is returned, otherwise, it is checked if the objects are instances of the same class.

729.0 $ExeSAMECLASS : () \xrightarrow{o} ()$
.1 $ExeSAMECLASS\ () \triangleq$
.2 let $[expr1\text{-}v, expr2\text{-}v] = STKM`Pop\ (2)$ in
.3 if $\neg$ is-$SEM`OBJ\text{-}Ref\ (expr1\text{-}v) \vee \neg$ is-$SEM`OBJ\text{-}Ref\ (expr2\text{-}v)$
.4 then $STKM`Push($mk-$SEM`BOOL\ ($false$))$
.5 else let mk-$SEM`OBJ\ (objnm1, \text{-}, \text{-}) = STATE`GetSemObjInTab\ (expr1\text{-}v)$,
.6 mk-$SEM`OBJ\ (objnm2, \text{-}, \text{-}) = STATE`GetSemObjInTab\ (expr2\text{-}v)$,
.7 $found =$ mk-$SEM`BOOL\ (objnm1 = objnm2)$ in
.8 $STKM`Push(found)$;

730.0 $ExeHISTORY : INSTRTP`HistoryKind \times AS`Name^{*} \xrightarrow{o} ()$
.1 $ExeHISTORY\ (kind, opnms) \triangleq$
.2 (dcl $sum : \mathbb{N} := 0$;
.3 for nm in $opnms$
.4 do let mk-$SEM`NUM\ (val) = STATE`LookUpHistory\ (kind, nm)$ in
.5 $sum := sum + val$;
.6 $STKM`Push($mk-$SEM`NUM\ (sum))$);

731.0 $ExeSTARTLIST : \mathbb{B} \xrightarrow{o} ()$
.1 $ExeSTARTLIST\ (isset) \triangleq$
.2 let $[instr] = STKM`Pop\ (1)$,
.3 $instr\text{-}s =$ if $isset$
.4 then let mk-$SEM`SET\ (s) = instr$ in
.5 s
.6 else $\{instr\}$ in
.7 for all $inst \in instr\text{-}s$
.8 do $SCHD`StartNewThread(inst)$
.9 pre $STKM`LenES\ (1) \wedge$
.10 let $[instr] = STKM`GetES\ (1)$ in
.11 if $isset$
.12 then $\forall\ inst \in$ elems $instr \cdot$ is-$SEM`OBJ\text{-}Ref\ (inst)$
.13 else is-$SEM`OBJ\text{-}Ref\ (instr)$;

732.0 $ExeGUARD : () \xrightarrow{o} ()$
.1 $ExeGUARD\ () \triangleq$
.2 let $[\text{-}, fct\text{-}v] = STKM`GetES\ (2)$ in
.3 if is-$SEM`ExplOP\ (fct\text{-}v)$
.4 then $GuardCheck(fct\text{-}v, fct\text{-}v.objref)$;

733.0 $\mathit{GuardCheck} : \mathit{SEM}`\mathit{ExplOP} \times [\mathit{SEM}`\mathit{OBJ\text{-}Ref}] \xrightarrow{o} ()$

.1 $\mathit{GuardCheck}\,(\mathit{op\text{-}v}, \mathit{obj}) \triangleq$
.2 $\quad \mathsf{let}\ \mathit{opnm} = \mathit{AUX}`\mathit{ConstructDoubleName}\,(\mathit{op\text{-}v.modName}, \mathit{op\text{-}v.fnName}),$
.3 $\qquad \mathit{refcl} = \mathsf{if}\ \mathit{obj} \neq \mathsf{nil}$
.4 $\qquad\qquad \mathsf{then}\ \mathit{STATE}`\mathit{GetNameOfObjRef}\,(\mathit{obj})$
.5 $\qquad\qquad \mathsf{else}\ \mathsf{nil}\ ,$
.6 $\qquad \mathit{permis\text{-}opnm} = \mathsf{if}\ \mathit{obj} \neq \mathsf{nil}$
.7 $\qquad\qquad\qquad \mathsf{then}\ \mathit{AUX}`\mathit{ConstructDoubleName}\,(\mathit{refcl}, \mathit{op\text{-}v.fnName})$
.8 $\qquad\qquad\qquad \mathsf{else}\ \mathit{opnm}\ \mathsf{in}$
.9 $\quad (\mathit{STATE}`\mathit{UpdateHistCount}(\mathit{opnm}, \mathsf{mk\text{-}}\mathit{INSTRTP}`\mathit{req}\,(), \mathit{op\text{-}v.objref})\,;$
.10 $\quad\ \mathit{STKM}`\mathit{SetGuard}(\mathit{permis\text{-}opnm}, \mathit{obj})\,);$

734.0 $\mathit{ExeTHREADID} : () \xrightarrow{o} ()$

.1 $\mathit{ExeTHREADID}\,() \triangleq$
.2 $\quad \mathit{STKM}`\mathit{Push}(\mathsf{mk\text{-}}\mathit{SEM}`\mathit{NUM}\,(\mathit{SCHD}`\mathit{CurThreadId}\,()))\,;$

1.13 Timing Instructions

735.0 $\mathit{ExeINCRTIME} : \mathbb{N} \xrightarrow{o} ()$

.1 $\mathit{ExeINCRTIME}\,(n) \triangleq$
.2 $\quad \mathsf{if}\ \neg\ \mathit{STKM}`\mathit{InDuration}\,()$
.3 $\quad \mathsf{then}\ \mathit{STKM}`\mathit{IncrRelTime}(n)\,;$

736.0 $\mathit{ExeINCRTIME\text{-}PREF} : \mathit{AS}`\mathit{UnaryOp} \times \mathbb{N} \xrightarrow{o} ()$

.1 $\mathit{ExeINCRTIME\text{-}PREF}\,(\mathit{opr}, \mathit{oh}) \triangleq$
.2 $\quad \mathsf{let}\ \mathit{esval} = \mathsf{hd}\ \mathit{STKM}`\mathit{GetES}\,(1)\ \mathsf{in}$
.3 $\quad \mathsf{let}\ \mathit{val} = \mathit{GetVal}\,(\mathit{esval})\ \mathsf{in}$
.4 $\quad \mathsf{if}\ \mathit{val} = \mathsf{nil}$
.5 $\quad \mathsf{then}\ \mathit{TimeError}()$

```
else let mult =
          cases opr :
            SETDISTRUNION,
            SETDISTRINTERSECT →
                if is-SEM‘SET (val) ∧
                   ∀ v ∈ val.v · is-SEM‘SET (v)
                then dsetsize (val)
                else nil ,
            SETPOWER → if is-SEM‘SET (val)
                        then (2 ↑ card val.v)
                        else nil ,
            SEQDISTRCONC → if is-SEM‘SEQ (val) ∧
                               ∀ v ∈ elems val.v · is-SEM‘SEQ (v)
                            then dseqsize (val)
                            else nil ,
            SEQELEMS,
            SEQINDICES → if is-SEM‘SEQ (val)
                          then len val.v
                          else nil ,
            SEQTAIL → if is-SEM‘SEQ (val)
                       then (len val.v − 1)
                       else nil ,
            MAPDOM → if is-SEM‘MAP (val)
                      then (card dom val.v)
                      else nil ,
            MAPRNG → if is-SEM‘MAP (val)
                      then (card rng val.v)
                      else nil ,
            MAPDISTRMERGE → if is-SEM‘SET (val) ∧
                                ∀ v ∈ val.v · is-SEM‘MAP (v)
                             then dmapsize (val)
                             else nil ,
            others → undefined
          end in
   if ¬ STKM‘InDuration ()
   then (if mult = nil
         then TimeError()
         else STKM‘IncrRelTime(mult × oh) )
```

functions

```
737.0  dsetsize : SEM‘SET → ℕ

       dsetsize (s) ≜
         if s.v = {}
         then 0
         else let s1 ∈ s.v in
              card s1.v + dsetsize (mk-SEM‘SET (s.v \ {s1}));
```

738.0 $dseqsize : SEM`SEQ \to \mathbb{N}$

.1 $dseqsize\,(s) \triangleq$

.2 if $s.v = []$

.3 then 0

.4 else len (hd $(s.v)).v + dseqsize\,($mk-$SEM`SEQ\,($tl $(s.v)))$;

739.0 $dmapsize : SEM`SET \to \mathbb{N}$

.1 $dmapsize\,(m) \triangleq$

.2 if $m.v = \{\}$

.3 then 0

.4 else let $s1 \in m.v$ in

.5 card dom $s1.v + dmapsize\,($mk-$SEM`SET\,(m.v \setminus \{s1\}))$

operations

740.0 $GetVal : STKM`EvalStackItem \xrightarrow{o} [SEM`VAL]$

.1 $GetVal\,(esval) \triangleq$

.2 if is-$AS`Name\,(esval) \vee$ is-$AS`OldName\,(esval)$

.3 then return $STATE`LookUp\,(esval)$

.4 elseif $SEM`IsSemVal\,(esval)$

.5 then return $esval$

.6 else return nil ;

741.0 $ExeINCRTIME\text{-}BIN : AS`BinaryOp \times \mathbb{N} \times [\mathbb{N}] \xrightarrow{o} ()$

.1 $ExeINCRTIME\text{-}BIN\,(opr, oh1, oh2) \triangleq$

.2 let $[r\text{-}esval, l\text{-}esval] = STKM`GetES\,(2)$ in

.3 let $rval = GetVal\,(r\text{-}esval)$,

.4 $lval = GetVal\,(l\text{-}esval)$ in

.5 if nil $\in \{rval, lval\}$

.6 then $TimeError()$

```
else let time =
        cases opr :
          NUMEXP → if is-SEM`NUM (rval)
                   then rval.v × oh1
                   else nil ,
          SETUNION,
          SETINTERSECT,
          PROPERSUBSET,
          SETMINUS → if is-SEM`SET (lval) ∧ is-SEM`SET (rval)
                     then (card lval.v + card rval.v) × oh1
                     else nil ,
          SUBSET → if is-SEM`SET (rval)
                   then card rval.v × oh1
                   else nil ,
          INSET,
          NOTINSET → if is-SEM`SET (rval)
                     then card rval.v × oh1
                     else nil ,
          SEQCONC → if is-SEM`SEQ (lval) ∧ is-SEM`SEQ (rval)
                    then (len lval.v + len rval.v) × oh1
                    else nil ,
          MAPMERGE → if is-SEM`MAP (lval) ∧ is-SEM`MAP (rval)
                     then card (dom lval.v ∩ dom rval.v) × oh2 +
                          (card dom lval.v + card dom rval.v) × oh1
                     else nil ,
          MAPDOMRESTTO → if is-SEM`SET (lval)
                         then card lval.v × oh1
                         else nil ,
          MAPDOMRESTBY → if is-SEM`SET (lval)∧is-SEM`MAP (rval)
                         then card (lval.v ∩ dom rval.v) × oh2 +
                              card (dom rval.v \ lval.v) × oh1
                         else nil ,
          MAPRNGRESTTO → if is-SEM`MAP (lval)∧is-SEM`SET (rval)
                         then card (rng lval.v ∩ rval.v) × oh1
                         else nil ,
          MAPRNGRESTBY → if is-SEM`MAP (lval)∧is-SEM`SET (rval)
                         then card (rng lval.v ∩ rval.v) × oh2 +
                              card (rng lval.v \ rval.v) × oh1
                         else nil ,
          COMPOSE → if is-SEM`MAP (lval) ∧ is-SEM`MAP (rval)
                    then card dom lval.v × oh1
                    elseif is-SEM`CompExplFN (lval) ∧
                           is-SEM`CompExplFN (rval)
                    then 0
                    else nil ,
          others → undefined
        end in
  if ¬ STKM`InDuration ()
  then (if time = nil
        then TimeError()
```

```
.57                 else STKM`IncrRelTime(time) );


742.0  ExeINCRTIME-NEW : ℕ × AS`Name -o-> ()
  .1   ExeINCRTIME-NEW (oh, nm) △
  .2     if ¬ STKM`InDuration ()
  .3     then let inhstrct = STATE`GetInhStrct (nm) ⁀ [{nm}] in
  .4            let instvars = [{cnm ↦ len STATE`GetInstVars (cnm) |
  .5                                  cnm ∈ inhstrct (i)} |
  .6                                 i ∈ inds inhstrct] in
  .7            STKM`IncrRelTime(oh × num-instvars (instvars))
```

functions

```
743.0  num-instvars : (AS`Name -m-> ℕ)* → ℕ
  .1   num-instvars (ivs) △
  .2     if ivs = []
  .3     then 0
  .4     else map-sum (hd ivs) + num-instvars (tl ivs);


744.0  map-sum : (AS`Name -m-> ℕ) → ℕ
  .1   map-sum (m) △
  .2     if m = {↦}
  .3     then 0
  .4     else let d ∈ dom m in
  .5            m (d) + map-sum ({d} ⩤ m)
```

operations

```
745.0  ExeINCRTIME-SETSEQMAP : ℕ -o-> ()
  .1   ExeINCRTIME-SETSEQMAP (oh) △
  .2     let [r-esval, l-esval] = STKM`GetES (2) in
  .3     let lval = GetVal (l-esval),
  .4         rval = GetVal (r-esval) in
  .5     let mult = if is-SEM`NUM (lval) ∧ is-SEM`NUM (rval)
  .6                then (if rval.v > lval.v
  .7                      then rval.v − lval.v
  .8                      else 0)
  .9                elseif is-SEM`MAP (rval)
  .10               then card dom rval.v
  .11               else nil  in
  .12    if ¬ STKM`InDuration ()
  .13    then (if mult = nil
  .14          then TimeError()
  .15          else STKM`IncrRelTime(mult × oh) );
```

746.0 $ExeINCRTIME\text{-}STARTLIST : \mathbb{N} \xrightarrow{o} ()$

.1 $ExeINCRTIME\text{-}STARTLIST\,(oh) \triangleq$

.2 let $esval = \mathsf{hd}\ STKM`GetES\,(1)$ in

.3 let $val = GetVal\,(esval)$ in

.4 if $\neg$ is-$SEM`SET\,(val)$

.5 then $TimeError()$

.6 elseif $\neg\, STKM`InDuration\,()$

.7 then $STKM`IncrRelTime(\mathsf{card}\ val.v \times oh)$;

747.0 $TimeError : () \xrightarrow{o} ()$

.1 $TimeError\,() \triangleq$

.2 skip

end $INSTR$

Test Suite : rtinfo.ast
Module : INSTR

Name	**#Calls**	**Coverage**
INSTR‘ExeBR	**undefined**	**undefined**
INSTR‘InvOK	**undefined**	**undefined**
INSTR‘ExeCBR	**undefined**	**undefined**
INSTR‘ExeDTC	**undefined**	**undefined**
INSTR‘ExePRE	**undefined**	**undefined**
INSTR‘GetVal	**undefined**	**undefined**
INSTR‘BindPat	**undefined**	**undefined**
INSTR‘ExeCNBR	**undefined**	**undefined**
INSTR‘ExeEOCL	**undefined**	**undefined**
INSTR‘ExeFREF	**undefined**	**undefined**
INSTR‘ExePOST	**undefined**	**undefined**
INSTR‘ExeSIZE	**undefined**	**undefined**
INSTR‘ExeSWAP	**undefined**	**undefined**
INSTR‘Permute	**undefined**	**undefined**
INSTR‘map-sum	**undefined**	**undefined**
INSTR‘EvalUNOP	**undefined**	**undefined**
INSTR‘ExeAPPLY	**undefined**	**undefined**
INSTR‘ExeGUARD	**undefined**	**undefined**
INSTR‘ExePOPTH	**undefined**	**undefined**
INSTR‘dmapsize	**undefined**	**undefined**
INSTR‘dseqsize	**undefined**	**undefined**
INSTR‘dsetsize	**undefined**	**undefined**
INSTR‘EvalBINOP	**undefined**	**undefined**
INSTR‘ExeADDSET	**undefined**	**undefined**
INSTR‘ExeATOMIC	**undefined**	**undefined**

Name	#Calls	Coverage
INSTR‘ExeDLCALL	**undefined**	**undefined**
INSTR‘ExeDTCSET	**undefined**	**undefined**
INSTR‘ExeISCONT	**undefined**	**undefined**
INSTR‘ExeISEXIT	**undefined**	**undefined**
INSTR‘ExeLOOKUP	**undefined**	**undefined**
INSTR‘ExeMKEXIT	**undefined**	**undefined**
INSTR‘ExeMOSREF	**undefined**	**undefined**
INSTR‘ExeNEWOBJ	**undefined**	**undefined**
INSTR‘ExeNOBODY	**undefined**	**undefined**
INSTR‘ExePPCALL	**undefined**	**undefined**
INSTR‘ExePUSHTH	**undefined**	**undefined**
INSTR‘ExeRANDOM	**undefined**	**undefined**
INSTR‘ExeRECMOD	**undefined**	**undefined**
INSTR‘ExeRETURN	**undefined**	**undefined**
INSTR‘ExeSETRNG	**undefined**	**undefined**
INSTR‘ExeSUBSEQ	**undefined**	**undefined**
INSTR‘ExeTUPSEL	**undefined**	**undefined**
INSTR‘TimeError	**undefined**	**undefined**
INSTR‘ExeCLOSENV	**undefined**	**undefined**
INSTR‘ExeCONTEXT	**undefined**	**undefined**
INSTR‘ExeCOPYVAL	**undefined**	**undefined**
INSTR‘ExeERRINST	**undefined**	**undefined**
INSTR‘ExeEXITVAL	**undefined**	**undefined**
INSTR‘ExeHISTORY	**undefined**	**undefined**
INSTR‘ExeISCHECK	**undefined**	**undefined**
INSTR‘ExePOSTENV	**undefined**	**undefined**
INSTR‘ExeRECCONS	**undefined**	**undefined**
INSTR‘ExeSELELEM	**undefined**	**undefined**
INSTR‘ExeSeqConc	**undefined**	**undefined**
INSTR‘GuardCheck	**undefined**	**undefined**
INSTR‘ExeASSIGNSD	**undefined**	**undefined**
INSTR‘ExeFIELDSEL	**undefined**	**undefined**
INSTR‘ExeINCRTIME	**undefined**	**undefined**
INSTR‘ExeISNEEXIT	**undefined**	**undefined**
INSTR‘ExeMATCHVAL	**undefined**	**undefined**
INSTR‘ExeNEWCOMPL	**undefined**	**undefined**
INSTR‘ExePOLYINST	**undefined**	**undefined**
INSTR‘ExeSELFEXPR	**undefined**	**undefined**
INSTR‘ExeSetUnion	**undefined**	**undefined**
INSTR‘ExeTHREADID	**undefined**	**undefined**
INSTR‘ExeTOKENVAL	**undefined**	**undefined**
INSTR‘ExeTRYMATCH	**undefined**	**undefined**
INSTR‘IsVdmStdLib	**undefined**	**undefined**
INSTR‘ExeAPPENDMAP	**undefined**	**undefined**
INSTR‘ExeAPPENDSEQ	**undefined**	**undefined**
INSTR‘ExeAPPENDTUP	**undefined**	**undefined**
INSTR‘ExeCALLGUARD	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
INSTR‘ExeEMPTYLIST	**undefined**	**undefined**
INSTR‘ExeINITCLASS	**undefined**	**undefined**
INSTR‘ExeISOFCLASS	**undefined**	**undefined**
INSTR‘ExeMULTBINDL	**undefined**	**undefined**
INSTR‘ExePOPBLKENV	**undefined**	**undefined**
INSTR‘ExeSAMECLASS	**undefined**	**undefined**
INSTR‘ExeSELBLKENV	**undefined**	**undefined**
INSTR‘ExeSTARTLIST	**undefined**	**undefined**
INSTR‘ExeTYPEJUDGE	**undefined**	**undefined**
INSTR‘num-instvars	**undefined**	**undefined**
INSTR‘ExeEnumAppend	**undefined**	**undefined**
INSTR‘ExeISEMPTYSEQ	**undefined**	**undefined**
INSTR‘ExeISEMPTYSET	**undefined**	**undefined**
INSTR‘ExeNONDETSTMT	**undefined**	**undefined**
INSTR‘ExeREMEXITVAL	**undefined**	**undefined**
INSTR‘ExeSELSEQELEM	**undefined**	**undefined**
INSTR‘ExeSEQMAPOVER	**undefined**	**undefined**
INSTR‘ExeADDTOBLKENV	**undefined**	**undefined**
INSTR‘ExeAPPENDESTCK	**undefined**	**undefined**
INSTR‘ExeBINDINSTVAR	**undefined**	**undefined**
INSTR‘ExeEmptyBlkEnv	**undefined**	**undefined**
INSTR‘ExeINCRCOUNTER	**undefined**	**undefined**
INSTR‘ExeSEQCOMPBIND	**undefined**	**undefined**
INSTR‘ExeTESTCOUNTER	**undefined**	**undefined**
INSTR‘ExeTRYANYMATCH	**undefined**	**undefined**
INSTR‘ExtendTypeInfo	**undefined**	**undefined**
INSTR‘UpdateTypeInfo	**undefined**	**undefined**
INSTR‘ApplyOpFnMapSeq	**undefined**	**undefined**
INSTR‘EvalExplFnApply	**undefined**	**undefined**
INSTR‘EvalExplOpApply	**undefined**	**undefined**
INSTR‘ExeAPPENDBLKENV	**undefined**	**undefined**
INSTR‘ExeFieldsAppend	**undefined**	**undefined**
INSTR‘ExeINCRTIME-BIN	**undefined**	**undefined**
INSTR‘ExeINCRTIME-NEW	**undefined**	**undefined**
INSTR‘ExeLOOKUPSTATIC	**undefined**	**undefined**
INSTR‘ExeMatchAndBind	**undefined**	**undefined**
INSTR‘ExeNEWPOSABSOBJ	**undefined**	**undefined**
INSTR‘ExePOLYTYPEINST	**undefined**	**undefined**
INSTR‘ExeREMSTACKELEM	**undefined**	**undefined**
INSTR‘ExeSEQELEMMATCH	**undefined**	**undefined**
INSTR‘CleanExplFnApply	**undefined**	**undefined**
INSTR‘CleanExplOpApply	**undefined**	**undefined**
INSTR‘ExeINCRTIME-PREF	**undefined**	**undefined**
INSTR‘ExeISOFBASECLASS	**undefined**	**undefined**
INSTR‘ExePOPCLNMCUROBJ	**undefined**	**undefined**
INSTR‘ExeSAMEBASECLASS	**undefined**	**undefined**
INSTR‘ExeUPDATECLOSENV	**undefined**	**undefined**

Name	#Calls	Coverage
INSTR‘EvalOverOpFnApply	**undefined**	**undefined**
INSTR‘ExePUSHCLNMCUROBJ	**undefined**	**undefined**
INSTR‘CleanFunctionApply	**undefined**	**undefined**
INSTR‘ExeVERIFYINDEXARGS	**undefined**	**undefined**
INSTR‘EnvSetUpExplFnApply	**undefined**	**undefined**
INSTR‘EnvSetUpExplOpApply	**undefined**	**undefined**
INSTR‘EvalCompExplFnApply	**undefined**	**undefined**
INSTR‘ExeINCRTIME-SETSEQMAP	**undefined**	**undefined**
INSTR‘ExeINCRTIME-STARTLIST	**undefined**	**undefined**
Total Coverage		**0%**

1.14 Semantic Value Domain

In this chapter, we describe the module SEM, which contains the definitions for the semantic value domain. The semantic value domain is used by the interpreter. The result of an evaluation function of the interpreter is a semantic value.

module *SEM*

imports

from *AS* all ,

from *CI* all ,

from *PAT* all ,

from *POS* all ,

from *REP* all ,

from *STKM* all ,

from *RTERR* all ,

from *STATE* all ,

from *GLOBAL* all ,

from *SCHDTP* all ,

from *INSTRTP* all ,

from *TIMEMAP* all ,

from *TIMEPARSER* all

exports all

definitions

1.14.1 Evaluation Stack

In the CSK VDM-SL Interpreter, we use an environment based evaluation model. The complete evaluation stack is of type *ENVL*.

types

761.0 $ENVL = ENV^*$;

762.0 $ENV = BlkEnv^*$;

An ENV is a function application environment, i.e. for each function application or operation call, we push a new function application environment onto the evaluation stack. After the function application or operation call is evaluated, the function application environment is popped from the evaluation stack, removing all bindings within the function application environment.

763.0 $BlkEnv :: id\text{-}m : AS`Name \xrightarrow{m} ValTp$
.1 $\quad perm : Permission$;

A block environment, $BlkEnv$, contains the bindings introduced by pattern matching, or the bindings of a closure environment. An environment can be either read-only or read/write.

764.0 $ValTp :: val : VAL$
.1 $\quad tp : [AS`Type]$;

The $ValTp$ is a record of a sematnic value, and for the case of Read/Write variables, a type.

765.0 $Permission = \text{READ\_ONLY} \mid \text{READ\_WRITE}$;

1.14.2 Object Reference Environment

The object reference environment is a stack that contains temporary object reference. The stack is used to ensure that temporary object references are deleted properly.

766.0 $OBJENVL = OBJENV^*$;

767.0 $OBJENV = ObjBlkEnv^*$;

768.0 $ObjBlkEnv = SEM`VAL\text{-}\mathsf{set}$;

The $OBJENVL$ has the same overall structure as the evaluation stack ($ENVL$). It is implemented this way because the creation and deletion of temporary object references scope follows the environment stack. Actually, the access function to the environment stack calls the corresponding access functions to the temporary object reference stack.

1.14.3 Semantic Values

769.0 $VAL = BasicVal \mid FN \mid OP \mid POLY \mid SEQ \mid SET \mid MAP \mid TUPLE \mid$
.1 $\quad REC \mid TOKEN \mid UNDEF \mid EXIT \mid CONT \mid RETURN \mid$
.2 $\quad OBJ \mid OBJ\text{-}Ref$;

Basic Values

770.0 $BasicVal = BOOL \mid NUM \mid CHAR \mid QUOTE \mid NIL;$

Basic values are booleans, *BOOL*, numerals, *NUM*, characters, *CHAR*, quotes, *QUOTE*, and the nil value, *NIL*.

771.0 $BOOL :: v : \mathbb{B};$

772.0 $NUM :: v : \mathbb{R};$

773.0 $CHAR :: v : \mathsf{char};$

774.0 $QUOTE :: v : \mathsf{char}^*;$

775.0 $NIL :: \quad ;$

Undefined Value

776.0 $UNDEF :: \quad ;$

An undefined value, *UNDEF*, is assigned to an identifier if, when the identifier is introduced, it is not initialized.

Exception Handling Values

777.0 $EXIT :: v : [VAL];$

The exit value, *EXIT*, is used to signal that an exception occured within the evaluation of an operation call. If an exception is raised, an optional value can be given.

Return Values

778.0 $CONT :: \quad ;$

The continue value, *CONT*, is used to signal that an operation call did not return a value.

779.0 $RETURN :: \quad ;$

This is used in loop statements or empty return statements.

Non-Polymorphic Function Values

780.0 $FN = CompExplFN \mid ImplFN;$

Function values can denote either explicit or implicit functions or an external function declared in an implementation module. These functions cannot be polymorphic.

The semantic value of a function is basically the sequence of explicit function values which must be applied in the given order to the argument. The reason for it being a sequence is in order to capture compose operators. Thus in most cases the length of this sequence will be one.

781.0 $CompExplFN :: fl : ExplFN^{+}$
.1 $objref : [OBJ\text{-}Ref];$

An explicit function value contains the original type information in AS format; the actual patterns in stack-machine format (thus any match values which should not be used in the header of a function are replaced with don't care patterns); a pointer to the instruction code for the body (in case it is nil, the function should be understood as an identity function); a closure environment; a mapping with potential instatiations of a polymorphic function; fnName ????; the name of the module/class in which the function is defined; the names and the type of any result parameters (in case these are given); and finally an optional object reference ??

782.0 $ExplFN :: tp : AS`DynFnType$
.1 $parms : STKM`Pattern^{**}$
.2 $instr : [STKM`SubProgramId]$
.3 $env : BlkEnv$
.4 $tm : AS`TypeVar \xrightarrow{m} AS`Type$
.5 $fnName : [AS`Name]$
.6 $modName : AS`Name$
.7 $resvars : [AS`NameType^{*}]$
.8 $objref : [OBJ\text{-}Ref];$

An explicit function value consists of the type of the function, *tp*, the parameters, *parms*, the body expression, *body*, the optional pre (*fnpre*) and post (*fnpost*) conditions, a closure environment, *env*, containing the bindings for the free variables in the body expression, and a type variable map, *tm*. The type variable map is used to resolve the occurences of type variables within the body expression of the function. *modName* contains the name of the module, in which the expression is defined. *fnName* contains the name of the function. This information is used in the implementation to collect runtime information. (see the file rtinfo.cc). *resvars* contains the $AS`NameType^{*}$ for the return values in extended explicit functions. *resvars* is nil for ordinary explicit functions.

783.0 $ImplFN :: \;;$

As we cannot evaluate implicit functions, we only use an empty composite type.

Polymorphic Function Values

784.0 $POLY = ExplPOLY \mid ImplPOLY;$

Polymorphic function values can denote explicit or implicit functions.

785.0 $ExplPOLY :: tpparms : AS`TypeVarList$
.1 $tp : AS`FnType$
.2 $parms : STKM`Pattern^{**}$
.3 $instr : STKM`SubProgramId$
.4 $env : BlkEnv$
.5 $fnName : [AS`Name]$
.6 $modName : AS`Name$
.7 $resvars : [AS`NameType^{*}]$
.8 $objref : [OBJ\text{-}Ref];$

An explicit polymorphic function value consists of a sequence of type variables, *tpparms*, the type of the function, *tp*, the parameters, *parms*, the body expression, *body*, the optional pre (*fnpre*) and post (*fnpost*) conditions, and a closure environment, *env*, containing the bindings for the free variables in the body expression. *fnName* contain the name of the function. This information is used in the implementation to collect runtime information. (see the file rtinfo.cc). *resvars* contains the $AS`NameType^{*}$ for the return values in extended explicit functions. *resvars* is nil for ordinary explicit functions.

786.0 $ImplPOLY :: \quad ;$

As we cannot evaluate implicit polymorphic functions, we only use an empty composite type.

787.0 $OP = ExplOP \mid ImplOP \mid$
.1 $OverOPFN;$

Operation values denote either implicit or explicit operations or operations declared in an implementation module.

788.0 $ExplOP :: tp : AS`OpType$
.1 $parms : STKM`Pattern^{*}$
.2 $instr : STKM`SubProgramId$
.3 $fnName : AS`Name$
.4 $modName : AS`Name$
.5 $resvars : [AS`NameType^{*}]$
.6 $objref : [OBJ\text{-}Ref];$

An explicit operation value is defined in terms of the type of the operation, *tp*, the parameters, *parms*, the body of the operation, *body*, and the optional pre (*oppre*) and post (*oppost*) conditions. *module* is the name of the module, in which the operation is defined. *fnName* contains the name of the operation. This information is used in the implementation to collect runtime information. (see the file rtinfo.cc) *resvars* contains the $AS`NameType^{*}$ for the return values in extended explicit operations. *resvars* is nil for ordinary explicit operations.

789.0 $ImplOP ::\ \ ;$

As we cannot evaluate implicit operations, we only use an empty composite type.

The `OverOPFN` type represent the semantic value for an overloaded function/operation. It contains a mapping from pairs of operation name (normally mangled but need not be if we deal with inheritance and a class only have one version of the definition) and the class name in which it is defined. The corresponding range values represent the sequence of types needed to determine which of the actual overload operations/functions that should be used for a given application.

790.0 $OverOPFN :: overload : (AS`Name \times AS`Name) \xrightarrow{m} AS`Type^*$
.1 $\qquad objref : [OBJ\text{-}Ref];$

The semantic value for an overloaded operation/function is a mapping from pairs of mangled name and class name to sequences of types. This information may be used to look up the proper semantic value depending on the actual parameter values.

Sequence Value

791.0 $SEQ :: v : VAL^*;$

Set Value

792.0 $SET :: v : VAL\text{-}\mathsf{set};$

Map Value

793.0 $MAP :: v : VAL \xrightarrow{m} VAL;$

Tuple Value

794.0 $TUPLE :: v : VAL^*;$

Record Value

Record values have a *tag* and two mappings corresponding to the normal fields and the "don't care" fields. These two mappings go from the the index in the record type definition to the corresponding concrete value.

795.0 $REC :: tag : AS`Name$
.1 $\qquad v : \mathbb{N} \xrightarrow{m} VAL$
.2 $\qquad v\text{-}dc : \text{-}\mathbb{N} \xrightarrow{m} VAL$

.3 $\mathsf{inv}\ \mathsf{mk}\text{-}REC\,(tag, \text{-}, \text{-}) \triangleq$
.4 $\quad \mathsf{let}\ \mathsf{mk}\text{-}AS`Name\,(ids, \text{-}) = tag\ \mathsf{in}$
.5 $\quad \mathsf{len}\ ids = 2;$

The invariant expresses that for record values the tag name should describe the class name where the type definition of the record is defined in addition with the real tag.

In the implementation the semantic record is implemented like this

```
SemRecord ::
  value   : vdmlib_Record(tag,size)
  checked : bool
```

The checked field is initialised to false when a semantic record is created and set true when it has been dynamic type checked, to avoid repeated dynamic type check of the same value.

The SemRecord is implemented by the SemRecTable class and SemRecord class. The "checked" field was previously part of the "value" where it was implemented as a boolean in a class derived from a RecordVal class.

Token Value

796.0 $TOKEN :: v : VAL;$

Object Value

THis is in essense only used for VDM++ but in order to minimise the differences between VDM-SL and VDM++ it has been kept here.

797.0 $OBJ\text{-}Ref :: \ \mathbb{N};$

This value is a refence value to the "real" semantic value of an object.

798.0 $OBJ :: tp : AS`Name$
.1 $\quad ins : InsStrct$
.2 $\quad hist : AS`Name \xrightarrow{m} History$
.3 $\mathsf{inv}\ obj \triangleq \mathsf{len}\ obj.tp.ids = 1;$

799.0 $InsStrct = AS`Name \xrightarrow{m} GLOBAL`ValueMap;$

800.0 $History :: req : \mathbb{N}$
.1 $\quad act : \mathbb{N}$
.2 $\quad fin : \mathbb{N}$

The semantics value of an object contains information about

- tp: the class name of which the object is an instance of, and
- ins: a map describing the values of the instance variable of the object. The map goes from class names to the value map.

```
class A
 instance variables
    a: nat;
    b: nat;
    init a == skip; !!
    init b == b := 1
types
   AType ::
      f1 : nat
methods
   ma1 () value r: nat ==
    (a:=434;
     return a-b)

end A


class B is subclass of A
  instance variables
    a: nat;
    c: bool

  methods
    m1(i: nat) ==
     a := i;

    m2(i: nat) value r: bool ==
     if a = i
     then return true
     else ( a := i;
            return false )

end B


class C is subclass of A, B, --B[<YELLOW>], B[true], B[1], B[3, ..., 5]
  instance variables
    a: nat;
    c: nat

  inherit
    from B :: m1

  methods mc1() value r: nat ==
    ( a:= 1;
      A`a := 2;
      B`a := 3;
      c:=self!A`ma1();
      return a
    )

end C
```

Figure 1.2: An example of a class structure.

Note, that class A is inherited twice in the class C, both directly by C and by its superclass B.

An object of class C could have the following semantic value:

```
mk_OBJ(
 mk_AS`Name(["C"], nil),
 mk_AS`Name(["C"], nil) |-> { mk_AS`Name(["a"], nil) |-> mk_SEM`NUM(2),
                              mk_AS`Name(["b"], nil) |-> mk_SEM`NUM(3) },
 mk_AS`Name(["B"], nil) |-> { mk_AS`Name(["a"], nil) |-> mk_SEM`NUM(45),
                              mk_AS`Name(["c"], nil) |-> mk_SEM`BOOL(true) },
 mk_AS`Name(["A"], nil) |-> { mk_AS`Name(["a"], nil) |-> mk_SEM`NUM(3),
                              mk_AS`Name(["b"], nil) |-> mk_SEM`NUM(6) } )
```

Figure 1.3: An example of the semantic value of an object of class C.

functions

801.0 $\mathit{CompFN} : \mathit{ExplFN} \to \mathit{CompExplFN}$

.1 $\mathit{CompFN}\,(\mathit{efn}) \triangleq$

.2 $\quad \textsf{mk-}\mathit{CompExplFN}\,([\mathit{efn}], \textsf{nil}\ );$

802.0 $\mathit{UpdateClosEnv} : \mathit{VAL} \times \mathit{BlkEnv} \to \mathit{VAL}$

.1 $\mathit{UpdateClosEnv}\,(\mathit{fnval}, \mathit{blkenv}) \triangleq$

.2 $\quad \textsf{if is-}\mathit{CompExplFN}\,(\mathit{fnval})$

.3 $\quad \textsf{then mk-}\mathit{CompExplFN}$

.4 $\qquad\qquad ($

.5 $\qquad\qquad [\mu\,(\mathit{fnval.fl}\,(i), \mathit{env} \mapsto \mathit{blkenv}) \mid$

.6 $\qquad\qquad\qquad i \in \textsf{inds}\ \mathit{fnval.fl}],$

.7 $\qquad\qquad \textsf{nil}\ )$

.8 $\quad \textsf{else}\ \mathit{fnval};$

803.0 $\mathit{IsSemValSeq} : \mathit{STKM}`\mathit{EvalStackItem} \to \mathbb{B}$

.1 $\mathit{IsSemValSeq}\,(\mathit{item}) \triangleq$

.2 $\quad \textsf{cases}\ \mathit{item} :$

.3 $\qquad [\,] \to \textsf{true},$

.4 $\qquad [\mathit{val}] \curvearrowright \mathit{val\text{-}l} \to \mathit{IsSemVal}\,(\mathit{val}) \wedge$

.5 $\qquad\qquad\qquad \forall\, v \in \textsf{elems}\ \mathit{val\text{-}l} \cdot \mathit{IsSemVal}\,(v),$

.6 $\qquad \textsf{others} \to \textsf{false}$

.7 $\quad \textsf{end};$

804.0 $\mathit{IsSemVal} : \mathit{STKM}`\mathit{EvalStackItem} \to \mathbb{B}$
.1 $\mathit{IsSemVal}\,(\mathit{val}) \triangleq$
.2 $\text{is-}\mathit{BOOL}\,(\mathit{val}) \lor$
.3 $\text{is-}\mathit{NUM}\,(\mathit{val}) \lor$
.4 $\text{is-}\mathit{CHAR}\,(\mathit{val}) \lor$
.5 $\text{is-}\mathit{QUOTE}\,(\mathit{val}) \lor$
.6 $\text{is-}\mathit{NIL}\,(\mathit{val}) \lor$
.7 $\text{is-}\mathit{CompExplFN}\,(\mathit{val}) \lor$
.8 $\text{is-}\mathit{ImplFN}\,(\mathit{val}) \lor$
.9 $\text{is-}\mathit{ExplOP}\,(\mathit{val}) \lor$
.10 $\text{is-}\mathit{ImplOP}\,(\mathit{val}) \lor$
.11 $\text{is-}\mathit{ExplPOLY}\,(\mathit{val}) \lor$
.12 $\text{is-}\mathit{ImplPOLY}\,(\mathit{val}) \lor$
.13 $\text{is-}\mathit{SEQ}\,(\mathit{val}) \lor$
.14 $\text{is-}\mathit{SET}\,(\mathit{val}) \lor$
.15 $\text{is-}\mathit{MAP}\,(\mathit{val}) \lor$
.16 $\text{is-}\mathit{TUPLE}\,(\mathit{val}) \lor$
.17 $\text{is-}\mathit{REC}\,(\mathit{val}) \lor$
.18 $\text{is-}\mathit{TOKEN}\,(\mathit{val}) \lor$
.19 $\text{is-}\mathit{UNDEF}\,(\mathit{val}) \lor$
.20 $\text{is-}\mathit{EXIT}\,(\mathit{val}) \lor$
.21 $\text{is-}\mathit{CONT}\,(\mathit{val}) \lor$
.22 $\text{is-}\mathit{RETURN}\,(\mathit{val}) \lor$
.23 $\text{is-}\mathit{OverOPFN}\,(\mathit{val}) \lor$
.24 $\text{is-}\mathit{OBJ}\,(\mathit{val}) \lor$
.25 $\text{is-}\mathit{OBJ\text{-}Ref}\,(\mathit{val})$

end SEM

1.15 Global Definitions and Types

In this chapter, we describe the module GLOBAl, which contains the global definitions used in the interpreter. Global definitions include global type definitions.
module GLOBAL

imports

805.0 from AS all ,

806.0 from CI all ,

807.0 from PAT all ,

808.0 from POS all ,

809.0 from REP all ,

810.0 from SEM all ,

811.0 from $STKM$ all ,

812.0 from $RTERR$ all ,

813.0 from $STATE$

814.0 operations $GetAllSupers : AS`Name \xrightarrow{o} AS`Name$-set;

.1 $GetSemObjInTab : SEM`OBJ\text{-}Ref \xrightarrow{o} SEM`OBJ$,

815.0 from $SCHDTP$ all ,

816.0 from $INSTRTP$ all

exports all

definitions

1.15.1 State Value

types

817.0 $State :: val : SEM`VAL$

.1 $\qquad tp : AS`Type$;

A state value, $State : Exp$, consists of a semantic value, val, and a type, tp.

The State type is used for local states.

1.15.2 Global Values

818.0 $ValueMap = AS`Name \xrightarrow{m} (SEM`VAL \times AS`Access)$

.1 inv $valmap \triangleq$

.2 $\quad \forall\, nm \in$ dom $valmap \cdot$

.3 $\qquad$ len $nm.ids = 1$;

The $ValuMap$ data type is used to store semantics values of values and instance variables and their access rigths.

1.15.3 Class Values

819.0 $\mathit{All\text{-}Fns\text{-}Ops\text{-}Polys} = \mathit{AS`Name} \xrightarrow{m} (\mathit{SEM`CompExplFN} \times \mathit{AS`Access} \mid$
.1 $\mathit{SEM`ImplFN} \times \mathit{AS`Access} \mid$
.2 $\mathit{SEM`ExplOP} \times \mathit{AS`Access} \mid$
.3 $\mathit{SEM`ImplOP} \times \mathit{AS`Access} \mid$
.4 $\mathit{SEM`ExplPOLY} \times \mathit{AS`Access} \mid$
.5 $\mathit{SEM`ImplPOLY} \times \mathit{AS`Access});$

820.0 $\mathit{SigmaClass} :: \mathit{inhcon} : \mathit{InhCon}$
.1 $\mathit{instvars} : \mathit{AS`InstAssignDef}^*$
.2 $\mathit{instvars\text{-}tp} : \mathit{AS`Name} \xrightarrow{m} \mathit{AS`Type}$
.3 $\mathit{inst\text{-}inv} : \mathit{AS`InstanceInv}^*$
.4 $\mathit{inst\text{-}init\text{-}val} : \mathit{SEM`InsStrct}$
.5 $\mathit{vls\text{-}def} : \mathit{AS`ValueDef}^*$
.6 $\mathit{vls\text{-}init} : \mathit{ValueMap}$
.7 $\mathit{explfns} : \mathit{AS`Name} \xrightarrow{m} (\mathit{SEM`CompExplFN} \times \mathit{AS`Access})$
.8 $\mathit{implfns} : \mathit{AS`Name} \xrightarrow{m} (\mathit{SEM`ImplFN} \times \mathit{AS`Access})$
.9 $\mathit{explops} : \mathit{AS`Name} \xrightarrow{m} (\mathit{SEM`ExplOP} \times \mathit{AS`Access})$
.10 $\mathit{implops} : \mathit{AS`Name} \xrightarrow{m} (\mathit{SEM`ImplOP} \times \mathit{AS`Access})$
.11 $\mathit{overloaded} : \mathit{AS`Name} \xrightarrow{m} \mathit{Overloaded}$
.12 $\mathit{explpolys} : \mathit{AS`Name} \xrightarrow{m} (\mathit{SEM`ExplPOLY} \times \mathit{AS`Access})$
.13 $\mathit{implpolys} : \mathit{AS`Name} \xrightarrow{m} (\mathit{SEM`ImplPOLY} \times \mathit{AS`Access})$
.14 $\mathit{localtps} : \mathit{AS`Name} \xrightarrow{m} \mathit{AS`TypeDef}$
.15 $\mathit{recsel} : \mathit{AS`Name} \xrightarrow{m} (\mathit{RecSel} \times \mathit{AS`Access})$
.16 $\mathit{localhchy} : \mathit{AS`Name} \xrightarrow{m} \mathit{AS`Name}\text{-}\mathsf{set}$
.17 $\mathit{isinit} : \mathbb{B}$
.18 $\mathit{constructor} : \mathit{AS`Type}^* \xrightarrow{m} (\mathit{STKM`DebugCmd} \times \mathit{AS`Access})$
.19 $\mathit{defaultcons} : \mathbb{B}$
.20 $\mathit{statics} : \mathit{AS`Name} \xrightarrow{m} (\mathit{SEM`VAL} \times \mathit{AS`Access})$
.21 $\mathit{perm\text{-}pred} : \mathit{AS`Name} \xrightarrow{m} \mathit{STKM`SubProgram}$
.22 $\mathit{thread} : [\mathit{STKM`SubProgram} \times [\mathbb{N}]]$
.23 $\mathit{all\text{-}fns\text{-}ops\text{-}polys} : \mathit{All\text{-}Fns\text{-}Ops\text{-}Polys}$

.24 inv $sgmcl \triangleq$
.25 $(\forall\, name \in \mathsf{dom}\ sgmcl.recsel \cdot$
.26 $\quad \mathsf{let\ mk\text{-}}AS`Name\,(idl, \text{-}) = name\ \mathsf{in}$
.27 $\quad \mathsf{len}\ idl = 1) \wedge$
.28 $(\forall\, typedef \in \mathsf{rng}\ sgmcl.localtps \cdot$
.29 $\quad \mathsf{let\ mk\text{-}}AS`TypeDef\,(nm, \text{-}, \text{-}, access, \text{-}) = typedef\ \mathsf{in}$
.30 $\quad \mathsf{len}\ nm.ids = 1) \wedge$
.31 $(\forall\, \mathsf{mk\text{-}}(\text{-}, access) \in$
.32 $\mathsf{rng}\ sgmcl.explfns \cup \mathsf{rng}\ sgmcl.implfns \cup$
.33 $\mathsf{rng}\ sgmcl.implops \cup$
.34 $\mathsf{rng}\ sgmcl.explops \cup$
.35 $\mathsf{rng}\ sgmcl.explpolys \cup$
.36 $\mathsf{rng}\ sgmcl.implpolys \cup$
.37 $\mathsf{rng}\ sgmcl.recsel \cup \mathsf{rng}\ sgmcl.vls\text{-}init \cdot$
.38 $\quad access \in \{\mathrm{PRIVATE\_AS}, \mathrm{PROTECTED\_AS}, \mathrm{PUBLIC\_AS}\});$

The *SigmaClass* describes:

- inhcon: The controlled inheritance of the class.

  821.0 $InhCon = AS`Name\text{-}\mathsf{set}$

  .1 inv $icon \triangleq \forall\, name \in icon \cdot$
  .2 $\quad \mathsf{let\ mk\text{-}}AS`Name\,(ids, \text{-}) = name\ \mathsf{in}$
  .3 $\quad \mathsf{len}\ ids = 1;$

  The Overloaded type is a mapping from the arity of the operation to a mapping from the mangled name to a sequence of types used in the parameters to the overloaded operation/function.

  822.0 $Overloaded = \mathbb{N} \xrightarrow{m} AS`Name \xrightarrow{m} AS`Type^*;$

  The map goes from the class name of the all the superclasses.

- instvars: The instance variables of the class.
- inst_inv: The instance invariant.
- inst_init_val: Contains the structure of the instance variables for the specific class. All instance variable are set to the semantic value undefined. This entry is used during initialisation of values, and for creating new objects (EvalNewExpr).
- vls_def. The value definitions.
- vls_init. The initialisation of values.
- fullmths. A map of the full methods in the class.
- prelmths: A map of the preliminary methods in the class.
- explfns: A map of the explicit functions in the class.
- implfns: A map of the implicit functions in the class.

- localtyps: A map of the types defined in the class.
- recsel: A map from the tag names of records to record selectors. Record selectors are used to extract the proper value from a record value.

  823.0 $RecSel = \mathbb{N} \times (AS`Name \xrightarrow{m} \mathbb{N}) \times AS`Type^*$;

  If the tuple: mk- $(n, pos, tp\text{-}l)$ is a record selector, we know that the corresponding record value has n components. The map pos contains for every component the corresponding index in the record value, and $tp\text{-}l$ is a sequence with the type for each of the components.
- localhchy: A map from class names to set of class names, describing the structure of the inheritance of this class.
- isinit: is a boolean describing if the class has been initialised.
- constructor: Contains a mapping of the stack machine instructions to initialize a new object of the class. The domain of this mapping reflects the types of the different constructors.
- defaultcons: This is a Boolean which indicates whether a default constructor (i.e. one without parameters) is present for the given class.

The next type is a map describing the hierarchy of the classes. It goes from a class name to all its superclasses including the classes that can be inherited through several levels.

824.0 $Hierarchy = AS`Name \xrightarrow{m} AS`Name\text{-}\mathsf{set}$;

The next type *InhStrct* is a type that describes the inheritance structure of certain class.

Consider the inheritance structure below where class A and D are super classes to G.

```
A - B - C
  \      \
   \      -G
    \    /
D -- E - F
```

A value of inheritance structure for G could be:

```
{ G |-> [{A,D},{B,E},{C,F}]}
```

825.0 $InhStrct = AS`Name \xrightarrow{m} (AS`Name\text{-}\mathsf{set}^*)$;

This value will be used to describe the order of initialising the objects of the classes.

826.0 $OBJscope :: obj : SEM`OBJ\text{-}Ref$
.1 $class : AS`Name^*$
.2 $origcl : OrigCl^*$;

The type *OBJscope* is used to describe the current object being evaluated. The *OBJscope* contains the semantic value of the object, and a stack of class names describing which class (scope) of the object that we are currently evaluating. Furthermore, it contains a class describing which class that has called the construction currently evaluating.

The type *OrigCl* is used to model the original class, that is the class from which the construction being evaluated is called from. See also the comments to the operation in the STATE moduel: *PushCurObj*. The original class can be a name of the original class, or it can be an empty record *Start*. The start record is used to model the very first element on the stack, that is, if the user writes

```
debug new A().Test()
```

In this case the original class is the debugger environment, and it will be modelled with the *mk-Start()* value.

827.0 $OrigCl = AS`Name \mid Start;$

828.0 $Start :: \; ;$

829.0 $OBJ\text{-}tab = SEM`OBJ\text{-}Ref \xrightarrow{m} OBJ\text{-}Desc;$

830.0 $OBJ\text{-}Desc :: ref\text{-}count : \mathbb{N}$
.1 $sem : SEM`OBJ$
.2 $threadid : [SCHDTP`ThreadId]$
.3 $DlClassInstancePtr : \mathsf{token};$

The type *OBJ-tab* is a table of all objects. The table goes from an object reference *OBJ-Ref* to a object description. The object description *OBJ-Desc* contains a reference counter denoting numbers of references to the object and the semantic value of the object.

The DlClassInstancePtr is used on the implementation level and is (naturally) a pointer to a DlClassInstance.

Furthermore it contains an optional threadid, denoting the thread that may have been started on this object.

Representation of Type Definitions

The type *Type* describes the various representation of type in VDM++. The abstract syntax of the VDM++ does not any longer have a representation of the object reference type. That is why the GLOBAL`Type is defined.

831.0 $Type = AS`Type \mid ObjRefType;$

832.0 $ObjRefType :: nm : AS`Name$

end $GLOBAL$

1.16 State Definition and Modification

The module STATE defines the state and contains all operations which access or modifiy the state.

module $STATE$

imports

833.0 from AS all ,

834.0 from CI all ,

835.0 from AUX

836.0 functions $ExtractName : AS`Name \rightarrow AS`Name$ renamed $ExtractName$;
.1 $ConstructName : AS`Id \times CI`ContextId \rightarrow AS`Name$;
.2 $ConstructDoubleName : AS`Name \times AS`Name \rightarrow AS`Name$

837.0 operations $IsInt : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsInt$;
.1 $IsNat : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsNat$;
.2 $IsRat : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsRat$;
.3 $IsReal : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsReal$;
.4 $ErrorOp$: char* $\xrightarrow{o}$
.5 $\mathbb{B} \mid SEM`VAL \mid AS`Name$-set $\mid$
.6 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL]) \mid AS`Name \mid$
.7 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL] \times [SEM`VAL] \times [AS`Name] \times [AS`Access]) \mid$
.8 $SEM`BlkEnv$-set $\mid (\mathbb{B} \times [GLOBAL`Type] \times [AS`Invariant] \times [AS`Name]) \mid$

.9 $GLOBAL`RecSel$ renamed $ErrorOp$;
.10 $IsNatOne : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsNatOne$;
.11 $IsRecSel : AS`Name \xrightarrow{o} \mathbb{B} \times [AS`Name \xrightarrow{m} \mathbb{N}]$;
.12 $IsTypeDef : AS`Name \xrightarrow{o}$
.13 $\mathbb{B} \times [GLOBAL`Type] \times [AS`Invariant] \times [AS`Name] \times [AS`Access]$ renamed $IsTypeDef$;
.14 $ErrorEmptyOp$: char* $\xrightarrow{o}$ ();
.15 $CombineBlkEnv : SEM`BlkEnv \times SEM`BlkEnv \xrightarrow{o} SEM`BlkEnv$ renamed $CombineBlkEnv$;
.16 $MkEmptyBlkEnv : SEM`Permission \xrightarrow{o} SEM`BlkEnv$ renamed $MkEmptyBlkEnv$;
.17 $ExtractTagName : AS`Name \xrightarrow{o} [AS`Name] \times \mathbb{B}$,

838.0 from DEF all ,

839.0 from PAT all ,

840.0 from POS all ,

841.0 from REP all ,

842.0 from SEM all ,

843.0 from $CMPL$ all ,

844.0 from $CPAT$ all ,

845.0 from $EXPR$

846.0 operations $ConvertPolyToFn : SEM`ExplPOLY \times AS`Type^* \xrightarrow{o} SEM`CompExplFN$ renamed $ConvertPolyTo$

,

847.0 from $SCHD$ all ,

848.0 from $STKM$ all ,

849.0 from $UTIL$ all ,

850.0 from $CLASS$

851.0 functions $TransLocalHchy : AS`Name \rightarrow AS`Name \xrightarrow{m} AS`Name$-set

852.0 operations $InitGV : AS`ValueDef^* \xrightarrow{o} GLOBAL`ValueMap$;
.1 $ExtOpDom : AS`OpDef \xrightarrow{o} AS`Type^*$;
.2 $GenInsMap : AS`Name \xrightarrow{o} ()$;
.3 $TransSyncs : AS`Document \xrightarrow{o} ()$;
.4 $EvalInhStrct : () \xrightarrow{o} ()$;
.5 $TransHierarchy : () \xrightarrow{o} ()$,

853.0 from $DEBUG$ all ,

854.0 from $INSTR$ all ,

855.0 from $RTERR$ all ,

856.0 from $GLOBAL$ all ,

857.0 from $MANGLE$ all ,

858.0 from $SCHDTP$ all ,

859.0 from $DEBUGTP$ all ,

860.0 from $INSTRTP$ all ,

861.0 from $TIMEMAP$ all ,

862.0 from $SETTINGS$

863.0 operations $DTC : () \xrightarrow{o} \mathbb{B}$ renamed DTC;
.1 $INV : () \xrightarrow{o} \mathbb{B}$ renamed INV ,

864.0 from $TIMETRACE$

865.0 operations $LogHistEvent{:}AS`Name \times AS`Ids \times INSTRTP`HistoryKind \times SEM`OBJ\text{-}Ref \xrightarrow{o} ()$,

866.0 from $TIMEPARSER$ all

exports all

definitions

To be able to evaluate expressions and statements, we have to maintain certain information available. This information is stored in the state used in the interpreter. This state is defined as:

types

867.0 $DLFactory ::$

In this state we maintain information for:

- A table of objects *obj_tab*
- A map from class names to the internal representation of classes (*classes*).

- A pool (*InitPool*) to be used in the initialisation of the classes. The stack contains the names of the classes that are to be initialised because the current class is dependent of this class.
- The hierarchy table (*hchy*).
- The *Init* is true if the specification should it be initialised. That is, it has not been initialised.

  The value of Init is used by CheckGlobInv() (called by INSTR‘InvOK) to determine if CheckStateInvariant() is called.
- A table of objects created in the debugger by the user (*d_objs*). This part of the state is only for the debugger. The table contains
  - the semantic value,
  - the abstract syntax of the expression or statement that the the variable was created with, and
  - the string of initialisation argument.

  This abstract syntax is used when initialising the specification, then the objects are initialised with the same expressions/statements. The string of the initialisation argument is used to write in the debugger what the initialisation argument is.
- A sequence of names (*d*-objs-order) which is the order in which the user defined objects should be initialised. This sequence of names should always be names which belongs to the domain of *d*-objs.
- inhstrct. This state contains for each class the inheritance structure of the superclasses of the class. See the type definition *GLOBAL‘InhStrct* for a description the structure.
- *obj-ref-no* is indicating the next reference number to be used for objects.
- *tm* a map representing the amount of time to increment for a given instruction.

`isTypeJudgement` is used to indicate whether we are currently evaluating a type judgement expression when we perform a subtype check.

868.0 $\mathsf{state}\ \mathit{Sigma}\ \mathsf{of}$
.1 $\mathit{obj\text{-}tab} : \mathit{GLOBAL}`\mathit{OBJ\text{-}tab}$
.2 $\mathit{classes} : \mathit{AS}`\mathit{Name} \xrightarrow{m} \mathit{GLOBAL}`\mathit{SigmaClass}$
.3 $\mathit{hchy} : \mathit{GLOBAL}`\mathit{Hierarchy}$
.4 $\mathit{Init} : \mathbb{B}$
.5 $\mathit{d\text{-}objs} : \mathit{AS}`\mathit{Name} \xrightarrow{m} (\mathit{SEM}`\mathit{OBJ\text{-}Ref} \times \mathit{AS}`\mathit{Expr} \times \mathsf{char}^*)$
.6 $\mathit{d\text{-}objs\text{-}order} : \mathit{AS}`\mathit{Name}^*$
.7 $\mathit{InitPool} : \mathit{AS}`\mathit{Name}\text{-}\mathsf{set}$
.8 $\mathit{inhstrct} : \mathit{GLOBAL}`\mathit{InhStrct}$
.9 $\mathit{abstracts} : \mathit{AS}`\mathit{Name}\text{-}\mathsf{set}$
.10 $\mathit{obj\text{-}ref\text{-}no} : \mathbb{N}$
.11 $\mathit{clsnms} : \mathit{AS}`\mathit{Name}\text{-}\mathsf{set}$
.12 $\mathit{dlops} : \mathit{AS}`\mathit{Name} \xrightarrow{m} \mathsf{token}$
.13 $\mathit{dlfactory} : \mathit{DLFactory}$
.14 $\mathit{isTypeJudgement} : \mathbb{B}$

.15 inv $s \triangleq s.abstracts \subseteq$ dom $s.classes \wedge$
.16 $\forall\, nm \in$ dom $s.classes \cdot$
.17 len $nm.ids = 1$
.18 init $s \triangleq s = GetInitSigma\,()$
.19 end

functions

869.0 $GetInitSigma : () \rightarrow Sigma$
.1 $GetInitSigma\,() \triangleq$
.2 mk-$Sigma$
.3 (
.4 $\{\mapsto\}$,
.5 $\{\mapsto\}$,
.6 $\{\mapsto\}$,
.7 true,
.8 $\{\mapsto\}$,
.9 $[]$,
.10 $\{\}$,
.11 $\{\mapsto\}$,
.12 $\{\}$,
.13 0,
.14 $\{\}$,
.15 $\{\mapsto\}$,
.16 mk-$DLFactory\,()$,
.17 false)

operations

870.0 $Init\text{-}Sigma : \mathbb{B} \xrightarrow{o} ()$
.1 $Init\text{-}Sigma\,(ast\text{-}is\text{-}new) \triangleq$
.2 (if $ast\text{-}is\text{-}new$
.3 then $(Sigma := GetInitSigma\,())$
.4 else $(RemoveClassValues()$;
.5 $obj\text{-}tab := \{\mapsto\}$;
.6 $obj\text{-}ref\text{-}no := 0))$;

1.16.1 Getting the obj_tab

871.0 $Get\text{-}obj\text{-}tab : () \xrightarrow{o} GLOBAL`OBJ\text{-}tab$
.1 $Get\text{-}obj\text{-}tab\,() \triangleq$
.2 return $obj\text{-}tab$;

872.0 $\mathit{Lookup\text{-}obj\text{-}tab} : \mathit{SEM}`\mathit{OBJ\text{-}Ref} \xrightarrow{o} \mathit{GLOBAL}`\mathit{OBJ\text{-}Desc}$
.1 $\mathit{Lookup\text{-}obj\text{-}tab}\,(o) \triangleq$
.2 $\quad$ return $\mathit{obj\text{-}tab}\,(o)$;

1.16.2 Document Translation

873.0 $\mathit{TranslateAST} : \mathit{AS}`\mathit{Document} \times \mathbb{B} \xrightarrow{o} ()$
.1 $\mathit{TranslateAST}\,(\mathit{cs}, \mathit{ast\text{-}is\text{-}new}) \triangleq$
.2 $\quad (\mathit{clsnms} := \{c.\mathit{nm} \mid c \in$ elems $\mathit{cs}\}$;
.3 $\quad$ for all $c \in$ elems cs
.4 $\quad$ do if $c.\mathit{useslib} \neq$ nil
.5 $\quad\quad$ then $\mathit{RegisterDLClass}(c.\mathit{nm}, c.\mathit{useslib}, c.\mathit{defs})$;
.6 $\quad$ if $\neg\, \mathit{ast\text{-}is\text{-}new}$
.7 $\quad$ then return ;
.8 $\quad$ for c in cs
.9 $\quad$ do let $\mathit{cnm} = c.\mathit{nm}$,
.10 $\quad\quad\quad \mathit{opm} =$ if $c.\mathit{defs} =$ nil
.11 $\quad\quad\quad\quad$ then $\{\mapsto\}$
.12 $\quad\quad\quad\quad$ else $c.\mathit{defs}.\mathit{opm}$ in
.13 $\quad\quad (\mathit{classes}(\mathit{cnm}) := \mathit{DEF}`\mathit{EmptySigmaClass}\,()$;
.14 $\quad\quad \mathit{classes}(\mathit{cnm}).\mathit{defaultcons} := \exists\, \mathit{opdef} \in$ rng $\mathit{opm} \cdot$
.15 $\quad\quad\quad \mathit{opdef}.\mathit{constr} \wedge \mathit{CLASS}`\mathit{ExtOpDom}\,(\mathit{opdef}) =$
[]);
.16 $\quad$ for c in cs
.17 $\quad$ do $\mathit{classes} := \mathit{classes} \dagger \mathit{DEF}`\mathit{ReadClasses}\,(c)$;
.18 $\quad \mathit{CLASS}`\mathit{TransHierarchy}()$;
.19 $\quad$ for c in cs
.20 $\quad$ do let $\mathit{cls} = c.\mathit{nm}$,
.21 $\quad\quad\quad \mathit{supercls} = c.\mathit{supercls}$,
.22 $\quad\quad\quad \mathit{opm} =$ if $c.\mathit{defs} =$ nil
.23 $\quad\quad\quad\quad$ then $\{\mapsto\}$
.24 $\quad\quad\quad\quad$ else $c.\mathit{defs}.\mathit{opm}$,
.25 $\quad\quad\quad \mathit{instvars} = \mathit{classes}\,(\mathit{cls}).\mathit{instvars}$,
.26 $\quad\quad\quad \mathit{cons} = \mathit{DEF}`\mathit{UpdateConstructors}\,(\mathit{cls}, \mathit{supercls}, \mathit{opm}, \mathit{instvars})$ in
.27 $\quad\quad \mathit{classes}(\mathit{cls}).\mathit{constructor} := \mathit{cons}$;
.28 $\quad \mathit{CLASS}`\mathit{EvalInhStrct}()$;
.29 $\quad \mathit{CLASS}`\mathit{TransSyncs}(\mathit{cs})$);

The operation *TranslateAST* translates a set of classes to the internal state s.

InitializeGSGV was formerly named InitializeGS

```
874.0  InitializeGSGV : 𝔹 -o-> ()
   .1  InitializeGSGV (ast-is-new) ≜
   .2    (let - = ast-is-new in
   .3     skip;
   .4     Init-classes() );
```

The operation *InitializeGSGV* initializes the global state of the users model.

1.16.3 Instance Invariants

```
875.0  CheckInstanceInvariant : () -o-> 𝔹
   .1  CheckInstanceInvariant () ≜
   .2    if ¬ INV ()
   .3    then return true
   .4    else (STKM`PushCurObjTab2OS() ;
   .5          let mk-SEM`OBJ (-, inststrct, -) = STKM`GetCurObj () in
   .6          for all cl ∈ dom inststrct
   .7          do (let Inv-cl-l = GetInstInv (cl) in
   .8              for mk-AS`InstanceInv (expr, access, -) in Inv-cl-l
   .9              do (STKM`PushBlkEnv(MkEmptyBlkEnv (READ_ONLY)) ;
  .10                  STKM`PushClNmCurObj(cl, cl) ;
  .11                  let mk- (eval-state, resval) =
  .12                          DEBUG`EvalUninterruptedCmd (expr, [], [], "CheckofInstanceInvariant") in
  .13                  (STKM`PopClNmCurObj() ;
  .14                   STKM`PopBlkEnv() ;
  .15                   let expr-v = if is-STKM`Success (eval-state)
  .16                                then resval
  .17                                else undefined  in
  .18                   (if is-SEM`BOOL (expr-v)
  .19                    then let mk-SEM`BOOL (b) = expr-v in
  .20                         if ¬ b
  .21                         then return false
  .22                         else skip
  .23                    else ErrorOp("Abooleanwasexpected") ))));
  .24          STKM`PopOS() ;
  .25          return true );
```

All invariants in the object (plus its superclasses) are transversed to check if they hold.

1.16.4 InitPool Operation

The next two operations *PushInitPool* and *PopInitPool* work on the pool *InitPool* by inserting and removing from the pool.

876.0 $\mathit{PushInitPool} : \mathit{AS}`\mathit{Name} \xrightarrow{o} ()$

.1 $\mathit{PushInitPool}\,(nm) \triangleq$

.2 $\quad \mathit{InitPool} := \mathit{InitPool} \cup \{nm\};$

877.0 $\mathit{PopInitPool} : \mathit{AS}`\mathit{Name} \xrightarrow{o} ()$

.1 $\mathit{PopInitPool}\,(nm) \triangleq$

.2 $\quad \mathsf{if}\ nm \in \mathit{InitPool}$

.3 $\quad \mathsf{then}\ \mathit{InitPool} := \mathit{InitPool} \setminus \{nm\}$

.4 $\quad \mathsf{else}\ \mathit{AUX}`\mathit{ErrorEmptyOp}(\texttt{"}\mathit{InternalErrorinPopInitPool}\texttt{"})\ ;$

1.16.5 Initialisation of the Sigma State

In the following it is assumed that the hierarchy state *hchy* has been updated. The initialisation of the *inst_init_val*, *vls_init*, and tps will be done so that the superclasses are initialised first. The *hchy* will be used to do this.

The *Sigma.classes* field is updated.

878.0 $\mathit{Init}\text{-}\mathit{classes} : () \xrightarrow{o} ()$

.1 $\mathit{Init}\text{-}\mathit{classes}\,() \triangleq$

.2 $\quad (\mathit{Init} := \mathsf{true};$

.3 $\quad\ \ \mathsf{for\ all}\ nm \in \mathsf{dom}\ \mathit{classes}$

.4 $\quad\ \ \mathsf{do}\ \mathit{SetClassInit}(nm, \mathsf{false})\ ;$

.5 $\quad\ \ \mathit{InitTheClasses}(\mathsf{dom}\ \mathit{classes})\ ;$

.6 $\quad\ \ \mathit{Init} := \mathsf{false});$

879.0 $\mathit{RemoveClassValues} : () \xrightarrow{o} ()$

.1 $\mathit{RemoveClassValues}\,() \triangleq$

.2 $\quad (\mathsf{for\ all}\ nm \in \mathsf{dom}\ \mathit{classes}$

.3 $\quad\ \ \mathsf{do}\ \mathit{classes}(nm).\mathit{vls}\text{-}\mathit{init} := \{\mapsto\});$

The operation *InitTheClasses* initialises the the classes in the order of the inheritance dependency, thus the super classes are initialised first. If one of the classes has associations (initialisation of instance varaibles or values) of an object of another class which has not yet been initialised, the call of initialisation is done in the operation *EvalNewStmt*. The pool *InitPool* is used to check if there is a cyclic dependency between the classes. Furthermore, each class in the state *Sigma* has a field *isinit* describing if it has been initialised.

880.0 $\mathit{InitTheClasses} : \mathit{AS}`\mathit{Name}\text{-}\mathsf{set} \xrightarrow{o} ()$

.1 $\mathit{InitTheClasses}\,(\mathit{not\text{-}done}) \triangleq$

.2 $\quad$ $\mathsf{if}\ \mathit{not\text{-}done} \neq \{\}$

.3 $\quad$ $\mathsf{then\ let}\ \ \mathit{nm} \in \mathit{not\text{-}done}\ \mathsf{be\ st}$

.4 $\qquad\qquad\qquad$ $\mathsf{let\ mk\text{-}}\,(\text{-}, s) = \mathit{LookupHchy}\,(\mathit{nm})\ \mathsf{in}$

.5 $\qquad\qquad\qquad$ $s \subseteq \mathit{InitClasses}\,()\ \mathsf{in}$

.6 $\qquad\quad$ $(\mathsf{if}\ \neg\, \mathit{IsClassInit}\,(\mathit{nm})$

.7 $\qquad\quad$ $\mathsf{then}\ \mathit{InitClassName}(\mathit{nm})\ ;$

.8 $\qquad\quad$ $\mathit{InitTheClasses}(\mathit{not\text{-}done} \setminus \{\mathit{nm}\})\ );$

It is assumed that the value definitions cannot contain references to instance variables. In the scope of the initialisation of instance variables it is legal to refer to values, both within the classes and values of super classes. Therefore, in the operation *InitClassName* we first initialises values and afterwards the instance variables are initialised. The local hierarchy is computed by the operation *CLASS'TransLocalHchy*, the initialisation of the values by *CLASS'InitClVls*, and the initialisation of the instance variables by the operation *CLASS'InitClInstVar*.

The operation *InitClVls* initialises the values defined in the value definitions *vl_def*. The operation calls the auxiliary operation *GenInsMap*. This operation creates all the instance variables in the *classes* state. The instance variables are set to the *mk-UNDEF()* value. It is assumed that the value definition must not depend of the instance variables. The strategy of the operation is to create a temporary object of the class and push it on the current object stack. The operation *GetVlsDef* computes values of the value definitions. The reason why we need to push the object on the stack is to create establish the right scope when comuting the value definition, that could depend on values from the super classes or of application of functions.

The *InitClassName* checks that all the super classes of the class *nm* is initialised. If not the super classes that are initialised will be initialised, otherwise the class is initialised. The name of the class is inserted in the *InitPool*, before it is initialised, and thus if there is a cyclic dependency of the initialisation of the class, the *InitPool* can be used to detect this (see *EvalNewStmt*).

```
881.0 InitClassName : AS`Name -o-> ()
InitClassName (nm) ≜
  (if IsClassInit (nm)
   then return ;
   if IsSuperClassesInit (nm)
   then (SetClassInit(nm, true) ;
         PushInitPool(nm) ;
         let localhchy = CLASS`TransLocalHchy (nm) in
         classes(nm) := μ (classes (nm),

                                 localhchy ↦ localhchy);
         let instvars = GetInstVars (nm) in
         (STKM`PushEmptyEnv() ;
          STKM`PushEmptyEnv() ;
          STKM`PushBlkEnv(MkEmptyBlkEnv (READ_WRITE)) ;
          CLASS`GenInsMap(nm) ;
          let tmp-inst = GetInstInitVal (nm) in
          let tmp-obj = mk-SEM`OBJ (nm, tmp-inst, {↦}) in
          let tmp-ref = STKM`MakeNewObj (tmp-obj) in
          (STKM`PushCurObj(tmp-ref, nil , nil ) ;
           let defaultcons = GetDefaultCons (nm) in
           InitStaticInsts(nm, instvars, defaultcons) ;
           STKM`ResetGuard() ;
           let vls-def = GetVlsDef (nm),
               vls-init = CLASS`InitGV (vls-def) in
           (STKM`PopCurObj() ;
            STKM`PopEnvL() ;
            STKM`PopEnvL() ;
            classes(nm).vls-init := vls-init));
          SetClassInit(nm, true) ;
          PopInitPool(nm) ))
   else let supers = {clnm | clnm ∈ GetAllSupers (nm) · ¬ IsClassInit (clnm)} in
        InitTheClasses(supers ∪ {nm}) );
```

The InitStaticInsts operation initialises all the instance variables which are declared static.

```
882.0 InitStaticInsts : AS`Name × AS`InstAssignDef* × 𝔹 -o-> ()
InitStaticInsts (clsnm, instdef-l, default) ≜
  (dcl statics : AS`Name -m-> (SEM`VAL × AS`Access) := {↦};
   for instdef in instdef-l
   do if instdef.static
      then classes(clsnm).statics := classes (clsnm).statics †
                                     {instdef.ad.var ↦ mk- (mk-SEM`UNDEF (), instdef.access)};
   for instdef in instdef-l
   do if instdef.static
      then let ad = instdef.ad in
           if ad.dclinit = nil
           then RTERR`InitError(RTERR`STATIC-IV-NO-VALUE, ad.var.cid)
```

```
            else let mk- (eval-state, res) =
                        DEBUG`EvalUninterruptedCmd (ad.dclinit, [], [],
                                                        "InitofStaticInstances"),
                  exp-v = if is-STKM`Success (eval-state)
                          then res
                          else undefined in
                (if DTC ()
                 then if ¬ SubType (exp-v, ad.tp)
                      then error
                      else skip;
                 classes(clsnm).statics := classes (clsnm).statics †
                                              {ad.var ↦ mk- (exp-v, instdef.access)}));
```

The operation *IsSuperClassesInit* checks of all the supers classes of class *name* has been changed.

```
883.0  IsSuperClassesInit : AS`Name →ᵒ 𝔹
       IsSuperClassesInit (name) ≜
         return ∀ nm ∈ GetAllSupers (name) · IsClassInit (nm);
```

The operation *SetInstanceVar* sets the value *val-v* to the instance variable identifier *nm*. All possible object references are incremented, and possible reference counters of the all oldvalue is decremented.

```
884.0  SetInstanceVar : AS`Name × SEM`VAL →ᵒ ()
       SetInstanceVar (nm, val-v) ≜
         (let mk- (isit, -, -, -, clnm, access) = IsInObjScope (nm, nil ),
              mk-SEM`OBJ (-, inststrct, -) = STKM`GetCurObj (),
              the-ref = STKM`GetCurObjRef (),
              thename = ExtractName (nm) in
          if isit ∧
             (clnm ∈ dom inststrct ∧ thename ∈ dom inststrct (clnm) ∨
              thename ∈ dom classes (clnm).statics)
          then (if thename ∈ dom classes (clnm).statics
                then let mk- (old-val, access) = classes (clnm).statics (thename) in
                       classes(clnm).statics := classes (clnm).statics †
                                                  {thename ↦ mk- (val-v, access)}
                else let mk- (old-val, access) = obj-tab (the-ref).sem.ins (clnm) (thename) in
                       obj-tab(the-ref).sem.ins(clnm)(thename) := mk- (val-v, access))
          else RTERR`Error(RTERR`INIT-NOT-POSSIBLE, nil , nil , []) );
```

```
885.0  IsDLClass : AS`Name →ᵒ 𝔹
       IsDLClass (n) ≜
         return n ∈ dom dlops;
```

886.0 $\mathit{InsertDLClass} : \mathit{AS}`\mathit{Name} \xrightarrow{o} ()$
.1 $\mathit{InsertDLClass}\,(\mathit{cl}) \triangleq$
.2 $\quad(\mathit{dlops}(\mathit{cl}) := \textsf{mk-token}\,(\textsf{nil}\,));$

887.0 $\mathit{InsertDLOp} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{Name} \xrightarrow{o} ()$
.1 $\mathit{InsertDLOp}\,(\mathit{cl}, \mathit{opnm}) \triangleq$
.2 $\quad(\textsf{let } \mathit{clOp} = \mathit{AUX}`\mathit{ConstructDoubleName}\,(\mathit{cl}, \mathit{opnm}) \textsf{ in}$
.3 $\quad\;\mathit{dlops}(\mathit{clOp}) := \textsf{mk-token}\,(\textsf{nil}\,));$

888.0 $\mathit{IsDLOp} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{Name} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{IsDLOp}\,(\mathit{cl}, \mathit{opnm}) \triangleq$
.2 $\quad(\textsf{let } \mathit{clOp} = \mathit{AUX}`\mathit{ConstructDoubleName}\,(\mathit{cl}, \mathit{opnm}) \textsf{ in}$
.3 $\quad\;\textsf{return } \mathit{clOp} \in \textsf{dom } \mathit{dlops}\;);$

889.0 $\mathit{RegisterDLClass} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{TextLit} \times [\mathit{AS}`\mathit{Definitions}] \xrightarrow{o} ()$
.1 $\mathit{RegisterDLClass}\,(\mathit{clnm}, \mathit{useslib}, \mathit{defs}) \triangleq$
.2 $\quad(\mathit{InsertDLClass}(\mathit{clnm})\,;$
.3 $\quad\;\textsf{if } \mathit{defs} \neq \textsf{nil}$
.4 $\quad\;\textsf{then } (\textsf{let } m = \mathit{defs}.\mathit{fnm} \uplus \mathit{defs}.\mathit{opm} \textsf{ in}$
.5 $\qquad\quad\textsf{for all } \mathit{nm} \in \textsf{ dom } m$
.6 $\qquad\quad\textsf{do if } m\,(\mathit{nm}).\mathit{body}.\mathit{body} = \mathrm{NOTYETSPEC}$
.7 $\qquad\qquad\textsf{then } \mathit{InsertDLOp}(\mathit{clnm}, \mathit{nm})\,));$

890.0 $\mathit{EvalFctTypeInstExpr} : \mathit{SEM}`\mathit{POLY} \times \mathit{AS}`\mathit{Type}^{*} \xrightarrow{o} \mathit{SEM}`\mathit{VAL}$
.1 $\mathit{EvalFctTypeInstExpr}\,(\mathit{fct\text{-}v}, \mathit{inst}) \triangleq$
.2 $\quad\textsf{if is-}\mathit{SEM}`\mathit{ImplPOLY}\,(\mathit{fct\text{-}v})$
.3 $\quad\textsf{then return mk-}\mathit{SEM}`\mathit{ImplFN}\,()$
.4 $\quad\textsf{elseif is-}\mathit{SEM}`\mathit{ExplPOLY}\,(\mathit{fct\text{-}v})$
.5 $\quad\textsf{then } (\textsf{dcl } \mathit{new\text{-}inst} : \mathit{AS}`\mathit{Type}^{*} := [\,];$
.6 $\qquad\quad\textsf{let } \mathit{tm} = \mathit{STKM}`\mathit{HdTypeInst}\,() \textsf{ in}$
.7 $\qquad\quad\textsf{for } \mathit{elm} \textsf{ in } \mathit{inst}$
.8 $\qquad\quad\textsf{do if is-}\mathit{AS}`\mathit{TypeVar}\,(\mathit{elm})$
.9 $\qquad\qquad\textsf{then if } \mathit{elm} \in \textsf{dom } \mathit{tm}$
.10 $\qquad\qquad\qquad\textsf{then } \mathit{new\text{-}inst} := \mathit{new\text{-}inst} \frown [\mathit{tm}\,(\mathit{elm})]$
.11 $\qquad\qquad\qquad\textsf{else error}$
.12 $\qquad\qquad\textsf{else } \mathit{new\text{-}inst} := \mathit{new\text{-}inst} \frown [\mathit{elm}];$
.13 $\qquad\quad\mathit{ConvertPolyToFn}(\mathit{fct\text{-}v}, \mathit{new\text{-}inst})\,)$
.14 $\quad\textsf{else error};$

This function instantiates a semantic polymorphic function value, and returns a semantic function value.

```
IsSubTypeName : SEM`VAL × AS`TypeName × 𝔹 →ᵒ 𝔹
IsSubTypeName (val-v, mk-AS`TypeName (tp, cid), checkinv) ≜
  (dcl res : 𝔹;
   let mk- (isinscope, type, Inv, defcl, access) = IsTypeDef (tp) in
   if ¬ isinscope
   then if access = nil
        then RTERR`ErrorVal(RTERR`TYPE-UNKNOWN, nil , nil , [])
        else RTERR`ErrorVal(RTERR`TYPE-NOT-IN-SCOPE, nil , nil , [])
   else (if ¬ RealSubType (val-v, type, checkinv)
         then res := false
         else if checkinv ∧ Inv ≠ nil
              then let env-s = PAT`PatternMatch (CPAT`P2P (Inv.pat),
                                                  val-v),
                       instr-pre = [mk-INSTRTP`NEWPOSABSOBJ (defcl)],
                       instr-post = [mk-INSTRTP`NEWCOMPL (false)] in
                   if env-s ≠ {}
                   then let env ∈ env-s in
                        (STKM`PushBlkEnv(env) ;
                         let mk- (eval-state, resval) =
                               DEBUG`EvalUninterruptedCmd (Inv.expr,
                                                            instr-pre,
                                                            instr-post, "DynamicTypeCheck") in
                        (let dummy = STKM`Pop (1) in
                          skip;
                          STKM`PopBlkEnv() ;
                          let Inv-v = if is-STKM`Success (eval-state)
                                      then resval
                                      else undefined  in
                          (if is-SEM`BOOL (Inv-v)
                           then let mk-SEM`BOOL (b) = Inv-v in
                                return b
                           else RTERR`ErrorVal(RTERR`BOOL-EXPECTED, Inv-v, nil , [])))))
                   else error
              else res := true);
   return res );
```

This operation is used in dynamic type checking, and is called by *SubType* in case the input type is a type name. The input value is checked against the type definition of the type name, and a possible invariant check is made. Input variables:

- val_v: A semantic value. *SubType* checks if it is a subtype of the type *tp*.
- tp: An AS'Type, or in case we are evaluating a a lambda expression an AllType, or in case we are evaluating the return type of an operation the optional type *nil*.

```
892.0 SubType : SEM`VAL × [GLOBAL`Type | AS`AllType] →ᵒ 𝔹
   .1 SubType (val-v, tp) ≜
   .2   if ¬ DTC () ∧ ¬ isTypeJudgement
   .3   then return true
   .4   else RealSubType(val-v, tp, INV ()) ;

893.0 SetTypeJudgement : () →ᵒ ()
   .1 SetTypeJudgement () ≜
   .2   isTypeJudgement := true;

894.0 UnsetTypeJudgement : () →ᵒ ()
   .1 UnsetTypeJudgement () ≜
   .2   isTypeJudgement := false;

895.0 RealSubType : SEM`VAL × [GLOBAL`Type | AS`AllType] × 𝔹 →ᵒ 𝔹
   .1 RealSubType (val-v, tp, checkinv) ≜
   .2    cases true:
   .3      (is-SEM`EXIT (val-v)),
   .4      (is-SEM`UNDEF (val-v)),
   .5      (is-SEM`TOKEN (val-v)),
   .6      ((is-SEM`CONT (val-v) ∨
   .7        is-SEM`RETURN (val-v)) ∧
   .8       tp = nil ),
   .9      (is-AS`AllType (tp)) →
  .10           return true,
  .11      (is-AS`BracketedType (tp)) → let mk-AS`BracketedType (btp, -) = tp in
  .12                                   RealSubType(val-v, btp, checkinv) ,
  .13      (is-AS`BasicType (tp)) → let mk-AS`BasicType (ttp, -) = tp in
  .14                               cases true:
  .15                                 (ttp = BOOLEAN) → return is-SEM`BOOL (val-v),
  .16                                 (ttp = CHAR) → return is-SEM`CHAR (val-v),
  .17                                 (ttp = NAT) → return IsNat (val-v),
  .18                                 (ttp = NATONE) → return IsNatOne (val-v),
  .19                                 (ttp = INTEGER) → return IsInt (val-v),
  .20                                 (ttp = REAL) → return IsReal (val-v),
  .21                                 (ttp = RAT) → return IsRat (val-v),
  .22                                 (ttp = TOKEN) → return true,
  .23                                 others → return false
  .24                               end,
  .25      (is-AS`QuoteType (tp)) → return if is-SEM`QUOTE (val-v)
  .26                                     then val-v.v = tp.lit.val
  .27                                     else false,
```

```
(is-AS`CompositeType (tp)) →
      let mk-AS`CompositeType (tag, fields-l, -) = tp in
      let mk- (the-tag, isinscope-tp) = AUX`ExtractTagName (tag) in
      if ¬ isinscope-tp
      then ErrorOp("Thetype * tp * isnotwithinthe" ↷
                "currentscope")
      else if ¬ is-SEM`REC (val-v)
           then return false
           else let mk-SEM`REC (rtag, v, v-dc) = val-v in
                let mk- (the-rtag, isinscope-val) =
                        AUX`ExtractTagName (rtag) in
                if isinscope-val ∧
                  (the-tag = the-rtag) ∧
                  (card dom v + card dom v-dc = len fields-l)
                then return (∀ i ∈ dom v ·
                                 RealSubType (v (i),
                                              fields-l (i).type,
                                              checkinv) ∧
                             ∀ i ∈ dom v-dc ·
                                 RealSubType (v-dc (i),
                                              fields-l (i).type,
                                              checkinv))
                else return false ,
(is-AS`UnionType (tp)) → let mk-AS`UnionType (tp-s, -) = tp in
                         return ∃ tp ∈ elems (tp-s) ·
                                   RealSubType (val-v, tp, checkinv),
(is-AS`ProductType (tp)) → let mk-AS`ProductType (tp-l, -) = tp in
                           return if is-SEM`TUPLE (val-v)
                                  then if (len val-v.v < 2) ∨
                                          (len tp-l < 2)
                                       then false
                                       elseif len val-v.v = len tp-l
                                       then let mk-SEM`TUPLE (v-tup) = val-v in
                                            ∀ i ∈ inds val-v.v ·
                                                RealSubType (v-tup (i),
                                                             tp-l (i),
                                                             checkinv)
                                       else false
                                  else false,
(is-AS`OptionalType (tp)) → let mk-AS`OptionalType (ttp, -) = tp in
                            return is-SEM`NIL (val-v) ∨
                                   RealSubType (val-v, ttp, checkinv),
(is-AS`SetType (tp)) → let mk-AS`SetType (ttp, -) = tp in
                       return if is-SEM`SET (val-v)
                              then ∀ sval-v ∈ val-v.v ·
                                       RealSubType (sval-v, ttp, checkinv)
                              else false,
```

```
(is-AS`Seq0Type (tp)) → let mk-AS`Seq0Type (ttp, -) = tp in
                        return if is-SEM`SEQ (val-v)
                               then let mk-SEM`SEQ (v-seq) = val-v in
                                    ∀ i ∈ inds val-v.v ·
                                          RealSubType (v-seq (i), ttp, checkinv)
                               else false,
(is-AS`Seq1Type (tp)) → let mk-AS`Seq1Type (ttp, -) = tp in
                        return if is-SEM`SEQ (val-v)
                               then (let mk-SEM`SEQ (v-seq) = val-v in
                                    (v-seq ≠ [] ∧
                                     (∀ i ∈ inds val-v.v ·
                                          RealSubType (v-seq (i), ttp, checkinv))))
                               else false,
(is-AS`GeneralMapType (tp)) → let mk-AS`GeneralMapType (dtp, rtp, -) = tp in
                              return if is-SEM`MAP (val-v)
                                     then (∀ dval-v ∈ dom val-v.v ·
                                                RealSubType (dval-v, dtp, checkinv))∧

                                          (∀ rval-v ∈ rng val-v.v ·
                                                RealSubType (rval-v, rtp, checkinv))
                                     else false,
(is-AS`InjectiveMapType (tp)) → let mk-AS`InjectiveMapType (dtp, rtp, -) = tp in
                                return if is-SEM`MAP (val-v)
                                       then (∀ dval-v ∈ dom val-v.v ·
                                                  RealSubType (dval-v, dtp, checkinv))∧

                                            (∀ rval-v ∈ rng val-v.v ·
                                                  RealSubType (rval-v, rtp, checkinv))∧

                                            (card dom val-v.v = card rng val-v.v)
                                       else false,
(is-AS`PartialFnType (tp)),
(is-AS`TotalFnType (tp)) → return is-SEM`CompExplFN (val-v) ∨ is-SEM`ImplFN (val-v),
(is-AS`TypeName (tp)) → IsSubTypeName(val-v, tp, checkinv) ,
(is-AS`TypeVar (tp)) → let tm = STKM`HdTypeInst () in
                       if tp ∈ dom tm
                       then RealSubType(val-v, tm (tp), checkinv)
                       else return false ,
(is-GLOBAL`ObjRefType (tp)) →
     let mk-GLOBAL`ObjRefType (nm) = tp in
     if ¬ is-SEM`OBJ-Ref (val-v)
     then return false
     else let mk-SEM`OBJ (nm-v, -, -) = GetSemObjInTab (val-v),
              allsupers = GetAllSupers (nm-v) ∪ {nm-v} in
          return nm ∈ allsupers ,
others → return false
end;
```

The operation *SubType* is used in dynamic type checking. The operation returns true if the input value is of the same type as the input type. If the input type is a polymorphic type variable, the proper type is looked up in the first element of the type instantiation map.

For the object reference types we only check if the object can be a subtype of the type *tp*. It is not checked if the instance variables in the object also are type correct. There are two reasons for this:

- The object can contain instance variables that are not yet initialised, that is the semantic value for these are UNDEF(). This can be the case if the object is instantiated with new, and the there is no initialisation statement for the instance variabels of the class.
- The instance variables will be checked anyway when they are changed, because this can only be done in an assign statement.

1.16.6 Names

```
LookUp : AS`Name -o-> SEM`VAL
LookUp (name) ≜
  (let mk- (isit, val) = STKM`IsLocalVal (name),
       mk- (isit-d-objs, val-obj) = IsDObjs (name) in
   if isit
   then ReturnLookUp(val, RTERR`INTERNAL-ERROR)
   else if STKM`IsEmptyObjL ()
        then if isit-d-objs
             then return val-obj
             else let mk- (isstatic, staticval) = LookUpStatic (name) in
                  if staticval = nil
                  then ErrorOp("Unknownidentifiername")
                  else return staticval
        else let mk- (isit-inst, local-inst, val-inst, -, cl-scope, access) =
                     IsInObjScope (name, nil ) in
             if isit-inst ∧ local-inst
             then return ReturnLookUp (val-inst, RTERR`INSTVAR-NOT-IN-SCOPE)
             else let mk- (isit-val, local-val, val-val) = IsValue (name) in
                  if isit-val ∧ local-val
                  then return ReturnLookUp (val-val, RTERR`VAL-NOT-IN-SCOPE)
                  else let mk- (isit-opfct, local-opfct, val-opfct) = LookOpFctPoly (name) in
                       if isit-opfct ∧ local-opfct
                       then ReturnLookUp(val-opfct, RTERR`OP-OR-FUN-NOT-IN-SCOPE)
                       else let mk- (isstatic, staticval) = LookUpStatic (name) in
                            cases true:
                              (isit-inst) → ReturnLookUp(val-inst, RTERR`INSTVAR-NOT-IN-SCOPE),
                              (isit-val) → ReturnLookUp(val-val, RTERR`VAL-NOT-IN-SCOPE),
                              (isit-opfct) → ReturnLookUp(val-opfct, RTERR`OP-OR-FUN-NOT-IN-SCOPE),
                              (isit-d-objs) → ReturnLookUp(val-obj, RTERR`INTERNAL-ERROR),
                              (isstatic) → if staticval = nil
                                           then RTERR`ErrorVal(RTERR`STATIC-NOT-IN-SCOPE, nil , nil , [])
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else\ return}\ \mathit{staticval}\ , \\
\qquad\qquad\qquad\qquad\qquad\qquad - \to \mathit{RTERR`ErrorVal}(\mathit{RTERR`NAME\text{-}UNKNOWN}, \mathsf{nil}\ , \mathsf{nil}\ , [\,]) \\
\qquad\qquad\qquad\qquad\qquad \mathsf{end});
\end{array}
$$

The operation *IsInObjScope* takes a *name* and returns indication if the *name* is an instance variable within the current scope, and if it is if the instance variable is defined within the current object or if it belongs to superclasses of the object. Lastly, the operation returns the semantics value of the instance variable, its type and class name where the instance variable is defined, if it exists within the scope.

The operation *IsInObjScope* may also take a second parameter *oldstate*, in this case this state is used to find the state of the object. This option is used to evaluate an old name.

The strategy of the operation is:

- If the name to look for is not qualified then look for the instance variable in the current class.
- Otherwise look find the class *classname* from where to from. It is either the qualification of the name or the current class.
- Look in the current object on the object list for all the instance variables that matches the name. Three cases can now occur:
  1. None instance variables were found.
  2. Only one instance variable was found, and that is the one to return.
  3. Several instance variables were found. If the name is qualified and the class contains the instance variable, this instance variable should be returned. Otherwise it should be investigated if the instance variables are overwritten in a one-line hierarchy. This is investigated with the operation *ExistsOneChild*. If only one-line hierarchy exists the instance variable in the lowest part of the hierarchy is returned, otherwise a run-time error is produced.

$$
\begin{array}{l}
\mathit{IsInObjScope} : \mathit{AS`Name} \times [\mathit{SEM`OBJ}] \xrightarrow{o} \\
\qquad\qquad\qquad \mathbb{B} \times \mathbb{B} \times [\mathit{SEM`VAL}] \times [\mathit{AS`Type}] \times [\mathit{AS`Name}] \times [\mathit{AS`Access}] \\
\mathit{IsInObjScope}\,(\mathit{name}, \mathit{oldstate}) \triangleq \\
\quad (\mathsf{if}\ \neg\, \mathit{STKM`HasCurCl}\,() \\
\quad\ \mathsf{then\ return\ mk\text{-}}(\mathsf{false}, \mathsf{false}, \mathsf{nil}\ , \mathsf{nil}\ , \mathsf{nil}\ , \mathsf{nil}\ )\ ; \\
\quad\ \mathsf{let\ mk\text{-}}\mathit{AS`Name}\,(l \curvearrowright [\text{-}], \mathit{cid}) = \mathit{name}, \\
\qquad\quad \mathit{origcl} = \mathit{STKM`GetOrigCl}\,(), \\
\qquad\quad \mathit{thename} = \mathit{ExtractName}\,(\mathit{name}), \\
\qquad\quad \mathit{clnm} = \mathsf{if}\ l = [\,] \\
\qquad\qquad\qquad\quad \mathsf{then}\ \mathit{STKM`GetCurCl}\,() \\
\qquad\qquad\qquad\quad \mathsf{else}\ \mathit{AUX`ConstructName}\,(\mathsf{hd}\ l, \mathit{cid}), \\
\qquad\quad \mathit{statics} = \mathsf{if}\ \mathit{clnm} \in \mathsf{dom}\ \mathit{classes} \\
\qquad\qquad\qquad\quad\ \mathsf{then}\ \mathit{classes}\,(\mathit{clnm}).\mathit{statics} \\
\qquad\qquad\qquad\quad\ \mathsf{else}\ \{\mapsto\},
\end{array}
$$

```
        isstatic =
            thename ∈ dom statics ∧
            let mk- (isopfct, -, -) = LookStaticOpFctPoly (clnm, thename) in
            ¬ isopfct,
        mk-SEM`OBJ (objnm, istrct, -) = if oldstate = nil
                                        then STKM`GetCurObj ()
                                        else oldstate,
        mk-AS`Name (- ⁀ [id], cid) = name in
    (if ¬ clnm ∈ dom istrct ∧
        ¬ isstatic
     then return mk- (false, false, nil , nil , nil , nil ) ;
     if ¬ STKM`HasCurCl ()
     then ErrorOp(" Unknownidentifier ∗ nm ∗ ")
     else (if l = [] ∨ isstatic
           then let local-inst =
                        if clnm ∈ dom istrct
                        then istrct (clnm)
                        else statics in
                (if thename ∈ dom local-inst
                 then let mk- (v, access) = local-inst (thename),
                          tp = GetInstVarsTp (clnm) (thename) in
                      if AccessOk (access, origcl, clnm)
                      then let real-v = if v = mk-SEM`UNDEF ()
                                        then let nm =
                                                  mk-AS`Name (clnm.ids ⁀
                                                              thename.ids,
                                                              name.cid),
                                                 mk- (-, v2) = LookUpStatic (nm) in
                                             if v2 = nil
                                             then v
                                             else v2
                                        else v in
                           return mk- (true, true, real-v, tp, clnm, access)
                      else return mk- (true, false, nil , tp, clnm, access) );
           let classname = if l ≠ []
                           then AUX`ConstructName (hd l, cid)
                           else clnm in
           (if ¬ classname ∈ dom istrct
            then return mk- (false, false, nil , nil , nil , nil ) ;
            let allsupers = GetAllSupers (classname) ∪ {classname} in
            let inst-vls = {clname ↦ istrct (clname) |
                                clname ∈ allsupers ·
                                    thename ∈ dom istrct (clname)} in
             cases dom inst-vls:
               {} → return mk- (false, false, nil , nil , nil , nil ),
               {cl} → let mk- (val, access) = inst-vls (cl) (thename),
                          tp = GetInstVarsTp (cl) (thename) in
                      if AccessOk (access, origcl, cl)
                      then return mk- (true, false, val, tp, cl, access)
                      else return mk- (true, false, nil , tp, cl, access) ,
```

.64 $\quad$ - $\rightarrow$ if $classname \in$ dom $inst\text{-}vls$
.65 $\quad\quad$ then let mk- $(val, access) = inst\text{-}vls\,(classname)\,(thename)$,
.66 $\quad\quad\quad\quad tp = GetInstVarsTp\,(classname)\,(thename)$ in
.67 $\quad\quad\quad$ if $AccessOk\,(access, origcl, classname)$
.68 $\quad\quad\quad$ then return mk- (true, false, $val, tp, classname, access$)
.69 $\quad\quad\quad$ else return mk- (true, false, nil , $tp, classname, access$)
.70 $\quad\quad$ else let mk- $(doesthere, child) = ExistsOneChild$ (dom $inst\text{-}vls$) in
.71 $\quad\quad\quad$ if $doesthere$
.72 $\quad\quad\quad$ then let mk- $(val, access) = inst\text{-}vls\,(child)\,(thename)$,
.73 $\quad\quad\quad\quad\quad tp = GetInstVarsTp\,(child)\,(thename)$ in
.74 $\quad\quad\quad\quad$ if $AccessOk\,(access, origcl, child)$
.75 $\quad\quad\quad\quad$ then return mk- (true, false, $val, tp, child, access$)
.76 $\quad\quad\quad\quad$ else return mk- (true, false, nil , $tp, child, access$)
.77 $\quad\quad\quad$ else $RTERR`ErrorVal(RTERR`MULT\text{-}DEF$, nil , nil , [])
.78 $\quad$ end))));

The operation *ExistsOneChild* takes a set of class names. The operation investigates if the classes inherit each other in one direct line, and if this is the case, the name of the subclass of them all is returned.

898.0 $ExistsOneChild : AS`Name$-set $\xrightarrow{o} \mathbb{B} \times [AS`Name]$
.1 $ExistsOneChild\,(cl\text{-}s) \triangleq$
.2 $\quad$ let $does\text{-}it = \exists\, cl \in cl\text{-}s \cdot$
.3 $\quad\quad\quad (cl\text{-}s \setminus \{cl\}) \subseteq GetAllSupers\,(cl)$ in
.4 $\quad$ if $does\text{-}it$
.5 $\quad$ then let $cl \in cl\text{-}s$ be st
.6 $\quad\quad\quad (cl\text{-}s \setminus \{cl\}) \subseteq GetAllSupers\,(cl)$ in
.7 $\quad\quad$ return mk- $(does\text{-}it, cl)$
.8 $\quad$ else return mk- (false, nil) ;

The operation *IsValue* investigates if the name *nm* is a values within the current scope. The operation returns three values:

- boolean: is a value found.
- boolean: is the value local. By locally is meant is the value bottom class of the hierarchy of the current object, or is the class qualified in the name *nm*.
- semantic value of the value if it is found.

The dynamic semantics supports that values can be static. That is, one can refer directly to class, without operating on an object. The strategy of the look up is more or less the same as in *IsInObjScope*, with the difference that we look in the initialised values for every class.

899.0 $IsValue : AS`Name \xrightarrow{o} \mathbb{B} \times \mathbb{B} \times [SEM`VAL]$
.1 $IsValue\,(nm) \triangleq$
.2 $\quad$ let mk-$AS`Name\,(l \curvearrowright [\text{-}], cid) = nm$,

```
      origcl = STKM`GetOrigCl (),
      thename = ExtractName (nm) in
  if l = [] ∧ ¬ STKM`HasCurCl ()
  then ErrorOp("Unknownidentifier * nm * ")
  else (if l = []
        then let clnm = STKM`GetCurCl (),
                 local-vls = GetVlsInit (clnm) in
             (if thename ∈ dom local-vls
              then let mk- (v, access) = local-vls (thename) in
                   if AccessOk (access, origcl, clnm)
                   then return mk- (true, true, v)
                   else return mk- (true, false, nil ) );
        let classname = if l ≠ []
                        then AUX`ConstructName (hd l, cid)
                        else nil  in
        let allsupers =
                if l = []
                then GetAllSupers (STKM`GetCurCl ())
                else GetAllSupers (classname) ∪ {classname} in
        let spcl-vls = {clname ↦ GetVlsInit (clname) |
                            clname ∈ allsupers ·
                               thename ∈ dom GetVlsInit (clname)} in
         cases  dom spcl-vls:
          {} → return mk- (false, false, nil ),
          {cl} → let mk- (val, access) = spcl-vls (cl) (thename) in
                 if AccessOk (access, origcl, cl)
                 then return mk- (true, false, val)
                 else return mk- (true, false, nil ) ,
          - → if classname ∈ dom spcl-vls
              then let mk- (val, access) = spcl-vls (classname) (thename) in
                    if AccessOk (access, origcl, classname)
                    then return mk- (true, false, val)
                    else return mk- (true, false, nil )
              else let mk- (doesthere, child) = ExistsOneChild (dom spcl-vls) in
                   if doesthere
                   then let mk- (val, access) = spcl-vls (child) (thename) in
                        if AccessOk (access, origcl, child)
                        then return mk- (true, false, val)
                        else return mk- (true, false, nil )
                   else RTERR`ErrorVal(RTERR`MULT-DEF, nil , nil , [])
        end);
```

The operation *LookOpFctPoly* looks for operations, functions and polymorphic functions. It takes as input the *name* of the operation or function, and it returns:

- a boolean: indicating if the function or operation was found.
- a boolean: indicating if the function or operation was found locally. By locally is meant in the first class in the search path of the current object.

- the semantic value if the function or operation was found. In case the function is found but access is not legal, because of the modifier on construct this value will be nil.

The strategy of operation is:

- Look first in the bottom of the object (if the name of the function/operation is not qualified).
- Otherwise look in hierarchy from either the qualification of the name or from the bottom of the object (objnm).
- There is now three cases:
    1. Either the function/operation was not found in the hierarchy.
    2. There is only one function in the hierarchy, and that must be the one.
    3. There is several functions/operations in the hierarchy. If the operation/function name is qualified and if there exists a function/operation in that class we should choose that function/operation. Otherwise we should investigate if the operations found is in a direct hierarchy in which they are overwriting each other. This is investigated with the operation (ExistsOneChild). If this is the case we should choose the function lowest in the hierarchy. If it is not the case: the operations/functions are found in several hierarchy lines, and a dynamic run time error is generated.

For every function/operation found it is checked if the functions/operations are accessible w.r.t. the modifiers of the functions/operations. This is checked with the operations: *ModifOk* and *IsClientOk*. The operation *IsClientOk* checks the accessibility of the function/operation if the caller of the function/operation is a client. The *ModifOk* checks the accessibility of the function/operation if the caller is not a client (that is, it is within the current object).

900.0 $LookOpFctPoly : AS`Name \xrightarrow{o} \mathbb{B} \times \mathbb{B} \times [SEM`VAL]$
.1 $LookOpFctPoly\,(name) \triangleq$
.2 (if $\neg$ $STKM`HasCurCl\,()$
.3 then return mk- (false, false, nil) ;
.4 let $thename = ExtractName\,(name)$,
.5 $clnm = STKM`GetCurCl\,()$,
.6 $objnm = STKM`GetCurObjName\,()$ in
.7 $LookOpFctPoly'(clnm, objnm, name, thename)$);

901.0 $LookStaticOpFctPoly : AS`Name \times AS`Name \xrightarrow{o} \mathbb{B} \times \mathbb{B} \times [SEM`VAL]$
.1 $LookStaticOpFctPoly\,(clnm, name) \triangleq$
.2 $(LookOpFctPoly'(clnm, clnm, name, name)$);

902.0 $LookOpFctPoly' : AS`Name \times AS`Name \times AS`Name \times AS`Name \xrightarrow{o} \mathbb{B} \times \mathbb{B} \times [SEM`VAL]$
.1 $LookOpFctPoly'\,(clnm, objnm, name, thename) \triangleq$
.2 (let mk-$AS`Name\,(l \curvearrowright [\text{-}], cid) = name$,

```
        origcl = STKM`GetOrigCl () in
    (if l = []
     then let mk-(found, opfn) = LookupAllFnsOpsPolys (objnm, thename) in
          (if found
           then let mk-(opval, access) = opfn in
                if AccessOk (access, origcl, objnm)
                then return mk-(true, true, opval)
                else return mk-(true, true, nil ) );
     let classname = if l ≠ []
                     then AUX`ConstructName (hd l, cid)
                     else objnm in
     let allsupers =
             if l = []
             then GetAllSupers (objnm)
             elseif classname ∉ GetAllSupers (clnm) ∧
                    classname ≠ clnm
             then {}
             else GetAllSupers (classname) ∪ {classname},
         opsfcts = {clname ↦ let mk-(found, opfn) =
                                 LookupAllFnsOpsPolys (clname, thename) in
                        opfn |
                            clname ∈ allsupers ·
                                 let mk-(found, opfn) =
                                          LookupAllFnsOpsPolys (clname, thename) in
                                 found} in
      cases dom opsfcts:
        {} → let overopfn = LookUpOverloaded (clnm, name, allsupers) in
             if overopfn ≠ nil
             then return mk-(true, true, overopfn)
             else return mk-(false, false, nil ) ,
        {cl} → let mk-(fnval, access) = opsfcts (cl),
                   overopfn = LookUpOverloaded (clnm, name, allsupers) in
               if overopfn = nil
               then if AccessOk (access, origcl, cl)
                    then return mk-(true, true, fnval)
                    else return mk-(true, false, nil )
               else let newover = overopfn.overload †
                                  {mk-(name, cl) ↦
                                   MANGLE`MethType2Seq (FindType (fnval))} in
                    return mk-(true, true, mk-SEM`OverOPFN (newover, nil )) ,
        - → if classname ∈ dom opsfcts
            then let mk-(fnval, access) = opsfcts (classname) in
                 if AccessOk (access, origcl, classname)
                 then return mk-(true, true, fnval)
                 else return mk-(true, false, nil )
            else let mk-(doesthere, child) = ExistsOneChild (dom opsfcts) in
                 if doesthere
                 then let mk-(fnval, access) = opsfcts (child) in
                      if AccessOk (access, origcl, child)
                      then return mk-(true, true, fnval)
```

```
.53                       else return mk- (true, false, nil )
.54                   else RTERR`ErrorVal(RTERR`MULT-DEF, nil , nil , [])
.55     end))
```

functions

```
903.0 FindType : SEM`CompExplFN | SEM`ExplOP →
   .1           AS`DynFnType | AS`FnType | AS`OpType
   .2 FindType (fnval) △
   .3   if is-SEM`CompExplFN (fnval)
   .4   then (hd fnval.fl).tp
   .5   else fnval.tp
```

operations

```
904.0 LookUpOverloaded : AS`Name × AS`Name × AS`Name-set -o-> [SEM`OverOPFN]
   .1 LookUpOverloaded (clsnm, name, supers) △
   .2   let localover = LookUpOverInClass (clsnm, name) in
   .3   (if localover ≠ nil
   .4    then return localover ;
   .5    let overloads =
   .6             {clname ↦ LookUpOverInClass (clname, name) |
   .7                  clname ∈ supers} ⩥
   .8             {nil } in
   .9     cases  dom overloads:
  .10       {} → return nil ,
  .11       {cl} → return overloads (cl),
  .12       - → if clsnm ∈ dom overloads
  .13           then return overloads (clsnm)
  .14           else let mk- (doesthere, child) = ExistsOneChild (dom overloads) in
  .15               if doesthere
  .16               then return overloads (child)
  .17               else RTERR`ErrorVal(RTERR`MULT-DEF, nil , nil , [])
  .18    end)
  .19 pre  {clsnm} ∪ supers ⊆ dom classes ;
```

```
905.0 LookUpOverInClass : AS`Name × AS`Name -o-> [SEM`OverOPFN]
   .1 LookUpOverInClass (clsnm, name) △
   .2   let overloaded = classes (clsnm).overloaded in
   .3   (if name ∈ dom overloaded
   .4    then let over = overloaded (name) in
   .5         (dcl load : (AS`Name × AS`Name) -m-> AS`Type* := {↦};
   .6          for all arit ∈  dom over
   .7          do load := load † {mk- (nm, clsnm) ↦ over (arit) (nm) |
   .8                             nm ∈ dom over (arit)};
   .9          return mk-SEM`OverOPFN (load, nil ) )
```

```
    else return nil )
pre clsnm ∈ dom classes ;
```

The LookUpStatic operation is used to look up statically declared constructs. In this case the argument name must always have two identifiers (where the first one is the name of the defining class and the second one is the construct which is defined statically).

```
906.0 LookUpStatic : AS`Name -o-> B × [SEM`VAL]
LookUpStatic (name) △
  if len name.ids = 2
  then let clsnm = AUX`ConstructName (name.ids (1), CI`NilContextId),
           memnm = AUX`ConstructName (name.ids (2), CI`NilContextId) in
       if clsnm ∈ dom classes
       then let statmap = classes (clsnm).statics,
                mk- (found, accessOk, val) = LookStaticOpFctPoly (clsnm, memnm) in
            if memnm ∈ dom statmap
            then let mk- (val, access) = statmap (memnm),
                     origcl = STKM`GetOrigCl () in
                 if AccessOk (access, origcl, clsnm)
                 then return mk- (true, val)
                 else return mk- (true, nil )
            elseif found
            then if accessOk
                 then return mk- (true, val)
                 else return mk- (true, nil )
            else return mk- (false, nil )
       else RTERR`ErrorVal(RTERR`CLNM-NOT-DEFINED, nil , nil , [])
  else LookUpStatic(mk-AS`Name ([(hd STKM`GetCurCl ().ids), hd name.ids], name.cid));
```

```
907.0 ReturnLookUp : [SEM`VAL] × RTERR`ERR -o-> SEM`VAL
ReturnLookUp (val, errno) △
  if is-SEM`UNDEF (val)
  then RTERR`ErrorVal(RTERR`UNDEF-ENCOUNTERED, nil , nil , [])
  elseif val = nil
  then RTERR`ErrorVal(errno, nil , nil , [])
  else return val ;
```

The operation *LookUp* returns the semantic value for given name. It searches first in the local values (block environment) then in the scope limited to only the object (that is, not is super classes). Following sequence is used: instance variables, values, methods and then functions. If the name is not in the local object scope, same order is used to search in the superclasses of the current object. If the name is not in this scope, it is investigated if the name is in the debugger objects *d_objs*.

The operation *IsDObjs* takes a name and returns an indication if the name is an object created by the user in the debugger, and if so, it return the semantic value of the object. The objects created by user in the debugger are stored in the state *d_objs*.

908.0 $IsDObjs : AS`Name \xrightarrow{o} \mathbb{B} \times [SEM`VAL]$
.1 $IsDObjs\,(name) \triangleq$
.2 if $name \in$ dom $d\text{-}objs$
.3 then let mk- $(val, \text{-}, \text{-}) = d\text{-}objs\,(name)$ in
.4 return mk- (true, val)
.5 else return mk- (false, nil) ;

1.16.7 State Designators

909.0 $EvalStateDesignator : STKM`StateDesignator \times SEM`VAL \xrightarrow{o} ()$
.1 $EvalStateDesignator\,(sd, val\text{-}v) \triangleq$
.2 let mk- $(scomp\text{-}v, index\text{-}l) = LookUpSD\,(sd)$ in
.3 let $id =$ hd $index\text{-}l$ in
.4 let $val = ModifyValue\,(scomp\text{-}v,$ tl $index\text{-}l, val\text{-}v)$ in
.5 $ModifyValueId(id, val)$;

This operation is used to evaluate a state designator. We first generate the current value and index sequence for the state designator. Next, the original value is modified, and the new value is saved.

910.0 $ModifyValueId : AS`Name \times SEM`VAL \xrightarrow{o} ()$
.1 $ModifyValueId\,(id, val\text{-}v) \triangleq$
.2 let mk- $(isit, ls\text{-}var) = STKM`IsLocalState\,(id)$ in
.3 if $isit$
.4 then if $DTC\,() \wedge \neg\, SubType\,(val\text{-}v, ls\text{-}var.tp)$
.5 then $RTERR`Error(RTERR`TYPE\text{-}INCOMP,$ nil , nil , [])
.6 else $STKM`SetLocalState(id,$ mk-$GLOBAL`State\,(val\text{-}v, ls\text{-}var.tp))$
.7 else let mk- $(isit, \text{-}, \text{-}, type, \text{-}, access) = IsInObjScope\,(id,$ nil) in
.8 if $isit$
.9 then if $DTC\,() \wedge \neg\, SubType\,(val\text{-}v, type)$
.10 then $RTERR`Error(RTERR`TYPE\text{-}INCOMP,$ nil , nil , [])
.11 else $SetInstanceVar(id, val\text{-}v)$
.12 else $AUX`ErrorEmptyOp($"InternalError"$)$;

This operation modifies the value of a state designator in either the local state or instance variables. If an instance variable is changed, possible invariant is also validated.

911.0 $ModifyValue : SEM`VAL \times (AS`Name \mid SEM`VAL)^{*} \times SEM`VAL \xrightarrow{o} SEM`VAL$
.1 $ModifyValue\,(scomp\text{-}v, index\text{-}l, val\text{-}v) \triangleq$
.2 if $index\text{-}l = []$
.3 then return $val\text{-}v$

```
.4     else if is-SEM`REC (scomp-v)
.5        then let mk-SEM`REC (tag, v, v-dc) = scomp-v in
.6           let mk- (isatag, pos) = AUX`IsRecSel (tag) in
.7           if isatag
.8           then if hd index-l ∈ dom pos
.9              then let index = pos (hd index-l) in
.10                if index ∈ dom v
.11                then return mk-SEM`REC (tag, v †
.12                                          {index ↦ ModifyValue (v (index),
.13                                                                tl index-l,
.14                                                                val-v)},
.15                                          v-dc)
.16                else return mk-SEM`REC (tag, v,
.17                                         v-dc †
.18                                         {index ↦ ModifyValue (v-dc (index),
.19                                                               tl index-l,
.20                                                               val-v)})
.21             else error
.22          else RTERR`ErrorVal(RTERR`RECORD-FIELD-ID-UNKNOWN, nil , nil , [])
.23       elseif is-SEM`MAP (scomp-v)
.24       then let mk-SEM`MAP (map-v) = scomp-v in
.25          if hd index-l ∈ dom map-v
.26          then return mk-SEM`MAP (map-v † {hd index-l ↦
.27                                   ModifyValue (map-v (hd index-l),
.28                                                tl index-l,
.29                                                val-v)})
.30          else if tl index-l ≠ []
.31             then RTERR`ErrorVal(RTERR`INDEXED-ASSIGN, nil , nil , [])
.32             else return mk-SEM`MAP (map-v † {hd index-l ↦ val-v})
.33       elseif is-SEM`SEQ (scomp-v)
.34       then let mk-SEM`SEQ (seq-v) = scomp-v in
.35          if IsNat (hd index-l)
.36          then let mk-SEM`NUM (index) = hd index-l in
.37             if index ∈ inds seq-v
.38             then return mk-SEM`SEQ (seq-v †
.39                                   {index ↦ ModifyValue (seq-v (index),
.40                                                         tl index-l,
.41                                                         val-v)})
.42             else RTERR`ErrorVal(RTERR`ILLEGAL-INDICES, nil , nil , [])
.43          else RTERR`ErrorVal(RTERR`NAT-EXPECTED, nil , nil , [])
.44       elseif is-SEM`OBJ-Ref (scomp-v)
.45       then ModifyInstanceVar(scomp-v, index-l, val-v)
.46       else RTERR`ErrorVal(RTERR`REF-UNKNOWN, nil , nil , []) ;
```

```
912.0  ModifyInstanceVar : SEM`OBJ-Ref × (AS`Name | SEM`VAL)* × SEM`VAL →(o)
   .1                      SEM`OBJ-Ref

   .2  ModifyInstanceVar (obj-ref, index-l, val-v) △
   .3    (dcl new-obj-l : SEM`VAL;
```

```
      let obj-v = obj-tab (obj-ref) in
      let obj = obj-v.sem,
          clsnm = obj.tp,
          index = hd index-l in
      (if is-AS`Name (index)
       then let insmap = obj.ins (clsnm) in
              if index ∈ dom insmap
              then let mk- (v, access) = insmap (index) in
                     if access = PUBLIC_AS
                     then let new-vm = insmap †
                                          {index ↦ mk-
                                              (
                                               ModifyValue (v, tl index-l, val-v),
                                               access)} in
                            let new-val = μ (obj,

                                             ins ↦ obj.ins † {clsnm ↦ new-vm}) in
                            let new-o = μ (obj-v, sem ↦ new-val) in
                            obj-tab := obj-tab † {obj-ref ↦ new-o}
                     else RTERR`ErrorVal(RTERR`NOT-IN-SCOPE, nil , nil , [])
              else RTERR`ErrorVal(RTERR`REF-UNKNOWN, nil , nil , [])
       else RTERR`ErrorVal(RTERR`REF-UNKNOWN, nil , nil , []) ;
       return obj-ref ))
  pre  obj-ref ∈ dom obj-tab ;
```

This operation is used to change the old value of the state designator. The index sequence is used to select the proper field in a record value, or the proper element in a sequence or map value.

```
913.0  LookUpSD : STKM`StateDesignator -o-> SEM`VAL × (AS`Name | SEM`VAL)*
       LookUpSD (sd) ≜
         (dcl tmp-type : GLOBAL`Type,
              tag : AS`Name,
              pval-v : SEM`VAL;
          if is-AS`Name (sd)
          then (let mk- (isit, ls-var) = STKM`IsLocalState (sd) in
                  if isit
                  then if is-SEM`UNDEF (ls-var.val)
                         then (tmp-type := ls-var.tp;
                                 while is-AS`TypeName (tmp-type)
                                 do let mk-AS`TypeName (tn,-) = tmp-type in
                                      let mk- (isit, thetype,-,-,-) = IsTypeDef (tn) in
                                      if isit
                                      then (tmp-type := thetype;
                                              tag := tn)
```

```
                else RTERR`ErrorVal(RTERR`TYPE-NOT-IN-SCOPE, nil , nil , []);
              if is-AS`CompositeType (tmp-type)
              then let mk-AS`CompositeType (-, fields-l, -) = tmp-type in
                  let v = {i ↦ mk-SEM`UNDEF () |
                              i ∈ inds fields-l · ¬ fields-l (i).dc},
                      v-dc = {i ↦ mk-SEM`UNDEF () |
                                i ∈ inds fields-l · ¬ fields-l (i).dc} in
                  let mk- (quotedtag, -) = AUX`ExtractTagName (tag) in
                  pval-v := mk-SEM`REC (quotedtag, v, v-dc)
              else pval-v := mk-SEM`UNDEF ();
              return mk- (pval-v, [sd]) )
          else return mk- (ls-var.val, [sd])
      else let mk- (isit, -, val, type, -, access) = IsInObjScope (sd, nil ) in
          if isit
          then if val = nil
              then RTERR`ErrorVal(RTERR`INSTVAR-NOT-IN-SCOPE, nil , nil , [])
              else if is-SEM`UNDEF (val)
                  then (tmp-type := type;
                      while is-AS`TypeName (tmp-type)
                      do let mk-AS`TypeName (tn, -) = tmp-type in
                          let mk- (isit, type, -, -, -) = IsTypeDef (tn) in
                          if isit
                          then (tmp-type := type;
                                tag := tn)
                          else AUX`ErrorEmptyOp(" Unknowntype * tn * ") ;
                      if is-AS`CompositeType (tmp-type)
                      then let mk-AS`CompositeType (-, fields-l, -) = tmp-type in
                          let v = {i ↦ mk-SEM`UNDEF () | i ∈ inds fields-l ·
                                          ¬ fields-l (i).dc},
                              v-dc = {i ↦ mk-SEM`UNDEF () |
                                          i ∈ inds fields-l · fields-l (i).dc} in
                          let mk- (quotedtag, -) = AUX`ExtractTagName (tag) in
                          pval-v := mk-SEM`REC (quotedtag, v, v-dc)
                      else pval-v := mk-SEM`UNDEF ();
                      return mk- (pval-v, [sd]) )
                  else return mk- (val, [sd])
          else RTERR`ErrorVal(RTERR`STATE-DESIG-UNKNOWN, nil , nil , []))
elseif is-STKM`FieldRef (sd)
then let mk-STKM`FieldRef (fsd, sel, -) = sd in
    let mk- (scomp-v, index-l) = LookUpSD (fsd) in
    return mk- (scomp-v, index-l ⁀ [sel])
elseif is-STKM`MapOrSeqRef (sd)
then let mk-STKM`MapOrSeqRef (msd, arg, -) = sd in
    let mk- (scomp-v, index-l) = LookUpSD (msd) in
    return mk- (scomp-v, index-l ⁀ [arg])
else error);
```

This operation returns the old value for the input state designator. It also returns a sequence that denotes the path to the value of the state designator. This path is used to change the old value of the state designator with the new value.

```
CheckGlobInv : () -o-> B
CheckGlobInv () △
  return ¬ Init ⇒ CheckInstanceInvariant ();
```

In the evaluation of a block statement, we first initialize the local state defined by the block statement. Next, we merge the resulting local state map with the current local state. Then, we evaluate each of the statements in the block statement.

1.16.8 Auxiliaries Operations for the state

The operation *GetFirstCIDOfFctOp* investigates if the function named *name* exists and if so returns the first contextid the module/class in which the function is defined, and the local name, that is, without qualification.

```
GetFirstCIDOfFctOp : AS`Name -o-> B×char*×[CI`ContextId]×[AS`Name]×[AS`Name]
GetFirstCIDOfFctOp (mk-AS`Name (ids, ci)) △
  (dcl sem : STKM`SubProgramId;
   if len ids = 2
   then let clnm = mk-AS`Name ([hd ids], ci),
            opfn = mk-AS`Name ([ids (2)], ci) in
        if clnm ∈ dom classes
        then let mk-GLOBAL`SigmaClass (-, -, -, -, -, -, -, explfns, -,
                                         explops, -, -, explpolys, -, -, -, -, -, -, -, -, -, -, -) =
                      classes (clnm) in
             (if opfn ∈ dom explfns
              then sem := let mk- (v, -) = explfns (opfn) in
                              (hd (v.fl)).instr
              elseif opfn ∈ dom explops
              then sem := let mk- (v, -) = explops (opfn) in
                              v.instr
              elseif opfn ∈ dom explpolys
              then sem := let mk- (v, -) = explpolys (opfn) in
                              v.instr
              else return mk- (false, "Nosuchmethod", nil , nil , nil ) ;
              let instr = CMPL`GetProgram (clnm, sem) in
              for i = 1 to len instr
              do if is-INSTRTP`CONTEXT (instr (i))
                 then return mk- (true, "", instr (i).cid, clnm, opfn) ;
              return mk- (false, "Internalerror", nil , nil , nil ) )
        else return mk- (false, "Nosuchclass", nil , nil , nil )
   else error)
pre len ids = 2 ;
```

1.16.9 Operations Regarding Synchronisation

916.0 $\mathit{LookUpPermis} : \mathit{AS}`\mathit{Name} \xrightarrow{o} [\mathit{STKM}`\mathit{SubProgram}]$

$\mathit{LookUpPermis}\,(\mathit{nm}) \triangleq$
 let mk-$\mathit{AS}`\mathit{Name}\,([\mathit{clnm}, \mathit{opnm}], \mathit{cid}) = \mathit{nm}$,
  $\mathit{perm\text{-}pred} = \mathit{classes}\,($mk-$\mathit{AS}`\mathit{Name}\,([\mathit{clnm}], \mathit{cid})).\mathit{perm\text{-}pred}$ in
 if mk-$\mathit{AS}`\mathit{Name}\,([\mathit{opnm}], \mathit{cid}) \in$ dom $\mathit{perm\text{-}pred}$
 then let $\mathit{pred} = \mathit{perm\text{-}pred}\,($mk-$\mathit{AS}`\mathit{Name}\,([\mathit{opnm}], \mathit{cid}))$ in
   if $\mathit{pred} = []$
   then return nil
   else return pred
 else return nil
pre len $\mathit{nm}.\mathit{ids} = 2$;

917.0 $\mathit{SetPermission} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{Name} \times \mathit{STKM}`\mathit{SubProgram} \xrightarrow{o} ()$

$\mathit{SetPermission}\,(\mathit{clnm}, \mathit{opnm}, \mathit{code}) \triangleq$
 $\mathit{classes}(\mathit{clnm}) := \mu\,(\mathit{classes}\,(\mathit{clnm}),$

  $\mathit{perm\text{-}pred} \mapsto \mathit{classes}\,(\mathit{clnm}).\mathit{perm\text{-}pred} \dagger \{\mathit{opnm} \mapsto \mathit{code}\});$

918.0 $\mathit{LookUpThread} : \mathit{AS}`\mathit{Name} \xrightarrow{o} [\mathit{STKM}`\mathit{SubProgram} \times [\mathbb{N}]]$

$\mathit{LookUpThread}\,(\mathit{clnm}) \triangleq$
 return $\mathit{classes}\,(\mathit{clnm}).\mathit{thread}$
pre len $\mathit{clnm}.\mathit{ids} = 1$;

919.0 $\mathit{SetPerThreadDef} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{PerObl} \xrightarrow{o} ()$

$\mathit{SetPerThreadDef}\,(\mathit{clnm}, \mathit{perobl}) \triangleq$
 let mk-$\mathit{AS}`\mathit{PerObl}\,(\mathit{dur}, \mathit{mtd}, \text{-}) = \mathit{perobl}$ in
 (dcl $\mathit{sp} : \mathit{STKM}`\mathit{SubProgram} := []$;
  $\mathit{sp} := \mathit{CMPL}`\mathit{SetContext}\,(\mathit{perobl}.\mathit{cid}, \mathsf{true})$;
  $\mathit{sp} := \mathit{sp} \frown [$mk-$\mathit{INSTRTP}`\mathit{EMPTYLIST}\,()$, mk-$\mathit{INSTRTP}`\mathit{CALLGUARD}\,(\mathsf{false}, \mathit{mtd})$,
    mk-$\mathit{INSTRTP}`\mathit{PPCALL}\,()$,
    mk-$\mathit{INSTRTP}`\mathit{EOCL}\,()]$;
  $\mathit{classes}(\mathit{clnm}) := \mu\,(\mathit{classes}\,(\mathit{clnm}), \mathit{thread} \mapsto$ mk-$\,(\mathit{sp}, \mathit{dur}.\mathit{val})))$;

920.0 $\mathit{SetThreadField} : \mathit{AS}`\mathit{Name} \times [\mathit{STKM}`\mathit{SubProgram} \times [\mathbb{N}]] \xrightarrow{o} ()$

$\mathit{SetThreadField}\,(\mathit{clnm}, \mathit{threadfield}) \triangleq$
 $\mathit{classes}(\mathit{clnm}) := \mu\,(\mathit{classes}\,(\mathit{clnm}), \mathit{thread} \mapsto \mathit{threadfield})$;

921.0 $SetThreadDef : AS`Name \times [STKM`SubProgram] \xrightarrow{o} ()$

.1 $SetThreadDef\,(clnm, code) \triangleq$

.2 $\quad classes(clnm) := \mu\,(classes\,(clnm),$

.3

.4 $\qquad thread \mapsto$ mk-

.5 $\qquad ($

.6 $\qquad code,$

.7 $\qquad$ nil $))$

.8 pre len $clnm.ids = 1$;

922.0 $GetThreadDef : AS`Name \xrightarrow{o} [STKM`SubProgram \times [\mathbb{N}]]$

.1 $GetThreadDef\,(clnm) \triangleq$

.2 $\quad$ return $classes\,(clnm).thread$;

1.16.10 Auxiliary Operations and Functions on the Sigma State

The operation *AddDObj* adds an entry in the table *d_objs*. The *d_objs* is a table of the objects created by user in the debugger.

The operation *GetSemObjInTab* returns the semantic value of a reference of an object in the object table *obj-tab*.

923.0 $GetSemObjInTab : SEM`OBJ\text{-}Ref \xrightarrow{o} SEM`OBJ$

.1 $GetSemObjInTab\,(nm) \triangleq$

.2 $\quad$ return $obj\text{-}tab\,(nm).sem$

.3 pre $nm \in$ dom $obj\text{-}tab$;

924.0 $GetClFromObjRef : SEM`OBJ\text{-}Ref \xrightarrow{o} AS`Name$

.1 $GetClFromObjRef\,(objref) \triangleq$

.2 $\quad$ if $objref \in$ dom $obj\text{-}tab$

.3 $\quad$ then return $obj\text{-}tab\,(objref).sem.tp$

.4 $\quad$ else error;

925.0 $GetObjRefAndClFromThreadId : SCHDTP`ThreadId \xrightarrow{o} SEM`OBJ\text{-}Ref \times AS`Name$

.1 $GetObjRefAndClFromThreadId\,(threadid) \triangleq$

.2 $\quad$ let $obj\text{-}ref \in$ dom $obj\text{-}tab$ be st $obj\text{-}tab\,(obj\text{-}ref).threadid = threadid$ in

.3 $\quad$ return (mk- $(obj\text{-}ref, obj\text{-}tab\,(obj\text{-}ref).sem.tp))$;

926.0 $GetObjTabLen : () \xrightarrow{o} \mathbb{N}$
.1 $GetObjTabLen\,() \triangleq$
.2 return card dom $obj\text{-}tab$;

The operation *GiveNextRefCountInTab* returns the next available reference number in the object table.

927.0 $GiveNextRefCountInTab : () \xrightarrow{o} SEM`OBJ\text{-}Ref$
.1 $GiveNextRefCountInTab\,() \triangleq$
.2 $(obj\text{-}ref\text{-}no := obj\text{-}ref\text{-}no + 1;$
.3 return mk-$SEM`OBJ\text{-}Ref\,(obj\text{-}ref\text{-}no)$);

The operation *MakeNewObj* creates a new object in the object table and returns an object reference.

928.0 $global\text{-}obj\text{-}tab\text{-}insert : SEM`OBJ \xrightarrow{o} SEM`OBJ\text{-}Ref$
.1 $global\text{-}obj\text{-}tab\text{-}insert\,(semobj) \triangleq$
.2 (let $ref = GiveNextRefCountInTab\,()$ in
.3 $(obj\text{-}tab := \{ref \mapsto$ mk-$GLOBAL`OBJ\text{-}Desc\,(1, semobj,$ nil , mk-token (nil))$\} \dagger obj\text{-}tab;$
.4 return ref));

929.0 $SetObjectThreadId : SEM`OBJ\text{-}Ref \times SCHDTP`ThreadId \xrightarrow{o} ()$
.1 $SetObjectThreadId\,(obj\text{-}ref, threadid) \triangleq$
.2 $obj\text{-}tab(obj\text{-}ref).threadid := threadid;$

The operation *GetAllClassNms* returns the names of all classes of the current state.

930.0 $GetAllClassNms : () \xrightarrow{o} AS`Name$-set
.1 $GetAllClassNms\,() \triangleq$
.2 return dom $classes$;

The auxiliry operation *IsClientModifOk* is used to check whether the use of a local identifier is a client or an inheritance one. The check is necessary to be able to distingush the situations where the protected access modifier is used. The modifier is all right is it is either public or if we have no sequence of calls (then it will always be ok when it is a local construct). Finally it is ok if the current scope and the calling scope follow the rules of the access modifier.

The operation *IsClient* checks if a function/operation called from class *origcl* is a client of a object that contains the class *cloffct*. The strategy of the operation is that *origcl* is not a client to *cloffct* if they are in each other hierarchy.

931.0 $IsClient : AS`Name \times AS`Name \xrightarrow{o} \mathbb{B}$

.1 $IsClient\,(origcl, cloffct) \triangleq$

.2 $\quad$ (let $orig\text{-}supers = GetAllSupers\,(origcl),$

.3 $\qquad cloffct\text{-}supers = GetAllSupers\,(cloffct)$ in

.4 $\quad$ return $\neg\,(origcl = cloffct \vee$

.5 $\qquad\qquad origcl \in cloffct\text{-}supers \vee$

.6 $\qquad\qquad cloffct \in orig\text{-}supers)$);

The operation *AccessOk* checks the accessibility of a function/operation in class *cloffct* with the assumption that the function/operation is called from class *origcl*.

The modifier on the function/operation and the fact if the caller of the function/operation is client or not affects the accessibility of a function/operation. Four cases can occur:

- The accessibility is public. In this case the function or operation can be accessed no matter if the caller of the function is client or not.
- The caller of the function is a client and the modifier on the function/operation is protected or private. In this case the function cannot be accessed.
- The caller of the function is not a client, and the modifier on the function is private. In this the class of the caller of the function *origcl* should be placed in the same class as the definition of the function *cloffct*, **or** the definition of the *cloffct* should be placed in a class that is a subclass to *cloffct*. The latter situation can only occur in the case where the function *cloffct* overwrites a function that is also defined in class *origcl*. An example of this represented in rep-61 in the test environment:

```
class A0

class A is subclass of A0

instance variables
  a: int := 0;
functions

public  op1: () -> int
  op1() ==
    op3();

  op3: () -> int
  op3() ==
     2;
end A

class B is subclass of A

functions
  op3: () -> int
  op3() ==
```

```
    1;

  Test: () -> int
  Test() ==
    op1();

end B
```

In this case *op3* in class B is private, and it is in fact the one that should be called.

- The caller of the function is not a client and the modifier of the function is protected. In this case the definition of the definition of the function could be up in the hierarchy of the caller of the function *origcl* or it could be down in the hierarchy. In the latter case the previous example is also an example of this.

932.0 $AccessOk : AS`Access \times GLOBAL`OrigCl \times AS`Name \xrightarrow{o} \mathbb{B}$

.1 $AccessOk\ (access, origcl, cloffct) \triangleq$

.2 $\quad$(if $origcl = \textsf{mk-}GLOBAL`Start\ ()$

.3 $\quad$ then return $access = \mathrm{PUBLIC\_AS}$;

.4 $\quad$ let $isClient = IsClient\ (origcl, cloffct)$,

.5 $\qquad origsupers = GetAllSupers\ (origcl)$,

.6 $\qquad cloffct\text{-}supers = GetAllSupers\ (cloffct)$ in

.7 $\quad$ cases true:

.8 $\qquad (access = \mathrm{PUBLIC\_AS}) \rightarrow$ return true,

.9 $\qquad (isClient \wedge$

.10 $\qquad ((\mathrm{PRIVATE\_AS} = access) \vee (\mathrm{PROTECTED\_AS} = access))) \rightarrow$

.11 $\qquad\quad$ return false,

.12 $\qquad (\neg\, isClient \wedge access = \mathrm{PRIVATE\_AS}) \rightarrow$

.13 $\qquad\quad$ return $(origcl = cloffct \vee$

.14 $\qquad\qquad \neg\,(cloffct \in origsupers))$,

.15 $\qquad (\neg\, isClient \wedge access = \mathrm{PROTECTED\_AS}) \rightarrow$ return true

.16 $\quad$ end;

.17 $\quad$ return $origcl = cloffct$);

The operation *GetNameOfObjRef* returns the name of the object that the object reference *objref* points at.

933.0 $GetNameOfObjRef : SEM`OBJ\text{-}Ref \xrightarrow{o} AS`Name$

.1 $GetNameOfObjRef\ (objref) \triangleq$

.2 $\quad$ let $\textsf{mk-}GLOBAL`OBJ\text{-}Desc\ (\text{-}, objsem, \text{-}, \text{-}) = obj\text{-}tab\ (objref)$ in

.3 $\quad$ let $\textsf{mk-}SEM`OBJ\ (nm, \text{-}, \text{-}) = objsem$ in

.4 $\quad$ return nm

.5 pre $objref \in$ dom $obj\text{-}tab$;

934.0
```
LookUpHistory : INSTRTP`HistoryKind × AS`Name →o SEM`NUM
LookUpHistory (kind, opnm) △
  let the-ref = STKM`GetCurObjRef (),
      realopnm = if len (opnm.ids) = 1
                 then mk-AS`Name ([hd (STKM`GetCurCl ().ids),
                                   hd opnm.ids],
                                  opnm.cid)
                 else opnm in
  let hist-m = obj-tab (the-ref).sem.hist in
  if realopnm ∉ dom hist-m
  then return mk-SEM`NUM (0)
  else let mk-SEM`History (r, a, f) = hist-m (realopnm),
           v = cases kind :
                 mk-INSTRTP`req () → r,
                 mk-INSTRTP`act () → a,
                 mk-INSTRTP`fin () → f,
                 mk-INSTRTP`active () → a − f,
                 mk-INSTRTP`waiting () → r − a
               end in
       return mk-SEM`NUM (v) ;
```

935.0
```
UpdateHistCount : AS`Name × INSTRTP`HistoryKind × [SEM`OBJ-Ref] →o ()
UpdateHistCount (opnm, kind, objref) △
  if STKM`GetObjLLen () ≠ 0
  then let the-ref = if objref ∉ dom obj-tab
                     then STKM`GetCurObjRef ()
                     else objref in
       let hist-m = obj-tab (the-ref).sem.hist in
       let new-hist = AddHist (opnm, hist-m, kind) in
       (TIMETRACE`LogHistEvent(GetClFromObjRef (the-ref), opnm.ids, kind,
                the-ref) ;
        obj-tab(the-ref).sem.hist := hist-m † {opnm ↦ new-hist})
pre kind ∈ {mk-INSTRTP`req (), mk-INSTRTP`act (), mk-INSTRTP`fin ()}
```

functions

936.0 $AddHist : AS`Name \times (AS`Name \xrightarrow{m} SEM`History) \times INSTRTP`HistoryKind \to$
.1 $\quad SEM`History$
.2 $AddHist\,(origopnm, hist\text{-}m, kind) \triangleq$
.3 let $opnm = origopnm$ in
.4 if $opnm \in$ dom $hist\text{-}m$
.5 then let mk-$SEM`History\,(r, a, f) = hist\text{-}m\,(opnm)$ in
.6 cases $kind$:
.7 mk-$INSTRTP`req\,() \to$ mk-$SEM`History\,(r + 1, a, f)$,
.8 mk-$INSTRTP`act\,() \to$ mk-$SEM`History\,(r, a + 1, f)$,
.9 mk-$INSTRTP`fin\,() \to$ mk-$SEM`History\,(r, a, f + 1)$
.10 end
.11 elseif $kind =$ mk-$INSTRTP`req\,()$
.12 then mk-$SEM`History\,(1, 0, 0)$
.13 elseif true
.14 then mk-$SEM`History\,(1, 1, 0)$
.15 else undefined
.16 pre $kind \in \{$mk-$INSTRTP`req\,()$, mk-$INSTRTP`act\,()$, mk-$INSTRTP`fin\,()\}$

The operation *LookupHchy* returns returns a boolean and the hierarhcy set for the class given by nm. The boolean indicates whether the name was a valid class.
operations

937.0 $LookupHchy : AS`Name \xrightarrow{o} \mathbb{B} \times AS`Name$-set
.1 $LookupHchy\,(nm) \triangleq$
.2 (if $nm \in$ dom $hchy$
.3 then return mk- (true, $hchy\,(nm)$)
.4 else return mk- (false, $\{\}$));

The operation *SetHchy* sets the global hiearchy state *hchy* to *clhchy*.

938.0 $SetHchy : GLOBAL`Hierarchy \xrightarrow{o} ()$
.1 $SetHchy\,(clhchy) \triangleq$
.2 $hchy := clhchy$;

The operation *GetInhStrct* returns the inheritance structure *inhstrct* for a given class name:

939.0 $GetInhStrct : AS`Name \xrightarrow{o} AS`Name$-set*
.1 $GetInhStrct\,(nm) \triangleq$
.2 return $inhstrct\,(nm)$;

The operation *SetInhStrct* returns the inheritance structure *inhstrct* for a given class name:

940.0 $SetInhStrct : GLOBAL`InhStrct \xrightarrow{o} ()$

.1 $SetInhStrct\,(new\text{-}inhstrct) \triangleq$

.2 $\quad inhstrct := new\text{-}inhstrct;$

The operation *GetAllSupers* returns the names of all the classes that are superclasses of the class *nm*, that is also the superclasses that are several levels above the class *nm*.

941.0 $GetAllSupers : AS`Name \xrightarrow{o} AS`Name\text{-}\mathsf{set}$

.1 $GetAllSupers\,(nm) \triangleq$

.2 $\quad \mathsf{if}\ nm \in \mathsf{dom}\ hchy$

.3 $\quad \mathsf{then\ return}\ hchy\,(nm)$

.4 $\quad \mathsf{else}\ RTERR`ErrorVal(RTERR`CLNM\text{-}NOT\text{-}DEFINED, \mathsf{nil}, \mathsf{nil}, [])$

functions

942.0 $AllSuperList : \mathbb{N} \times AS`Name^* \to AS`Name^*$

.1 $AllSuperList\,(index, supers) \triangleq$

.2 $\quad \mathsf{if}\ index = \mathsf{len}\ supers + 1$

.3 $\quad \mathsf{then}\ supers$

.4 $\quad \mathsf{else\ let}\ news = GetAllSupers\,(supers\,(index)),$

.5 $\qquad\qquad newlist = UTIL`set2seq[AS`Name]\,(\{nm \mid nm \in news \setminus \mathsf{elems}\ supers\})\ \mathsf{in}$

.6 $\qquad\quad AllSuperList\,(index + 1, supers \frown newlist)$

.7 $\mathsf{pre}\ \ index \in \mathsf{inds}\ supers \cup \{\mathsf{len}\ supers + 1\}$

1.16.11 Auxiliary Functions and Operations on SigmaClass Data Type

The operation *GetClasses* returns the value of the state *classes*.
operations

943.0 $GetClasses : () \xrightarrow{o} AS`Name \xrightarrow{m} GLOBAL`SigmaClass$

.1 $GetClasses\,() \triangleq$

.2 $\quad \mathsf{return}\ classes;$

944.0 $GetDefaultCons : AS`Name \xrightarrow{o} \mathbb{B}$

.1 $GetDefaultCons\,(nm) \triangleq$

.2 $\quad \mathsf{return}\ classes\,(nm).defaultcons$

.3 $\mathsf{pre}\ \ nm \in \mathsf{dom}\ classes\ ;$

945.0 $AddAbstract : AS`Name \xrightarrow{o} ()$

.1 $AddAbstract\,(clsid) \triangleq$

.2 $\quad abstracts := abstracts \cup \{clsid\};$

946.0 $CheckIfAbstractClass : AS`Name \xrightarrow{o} \mathbb{B}$
.1 $CheckIfAbstractClass\,(name) \triangleq$
.2 return $name \in abstracts$;

The operation *IsAClass* decides if the name *name* is a name of class.

947.0 $IsAClass : AS`Name \xrightarrow{o} \mathbb{B}$
.1 $IsAClass\,(name) \triangleq$
.2 return $name \in$ dom $classes$;

The operation *GetInhCon* returns classes that has been inherited.

948.0 $GetInhCon : AS`Name \xrightarrow{o} GLOBAL`InhCon$
.1 $GetInhCon\,(nm) \triangleq$
.2 return $classes\,(nm).inhcon$;

The operation *GetInstVars* returns a map of the instance variables names to its type.

949.0 $GetInstVars : AS`Name \xrightarrow{o} AS`InstAssignDef^*$
.1 $GetInstVars\,(nm) \triangleq$
.2 return $classes\,(nm).instvars$;

950.0 $GetInstVarsTp : AS`Name \xrightarrow{o} AS`Name \xrightarrow{m} AS`Type$
.1 $GetInstVarsTp\,(nm) \triangleq$
.2 return $classes\,(nm).instvars\text{-}tp$;

The operation *GetInstInv* returns the invariants expressions of the instance variables of class *nm*.

951.0 $GetInstInv : AS`Name \xrightarrow{o} AS`InstanceInv^*$
.1 $GetInstInv\,(nm) \triangleq$
.2 return $classes\,(nm).inst\text{-}inv$;

The operation *GetInstInitVal* returns the initial semantic values of the instance variables of the class named *name*

952.0 $GetInstInitVal : AS`Name \xrightarrow{o} SEM`InsStrct$

.1 $GetInstInitVal\,(name) \triangleq$

.2 $\quad$ return $classes\,(name).inst\text{-}init\text{-}val;$

The operation *SetInstInitVal* sets the initial semantics values of the instance variables of class *nm*.

953.0 $SetInstInitVal : AS`Name \times SEM`InsStrct \xrightarrow{o} ()$

.1 $SetInstInitVal\,(name, instval) \triangleq$

.2 $\quad classes(name).inst\text{-}init\text{-}val := \{nm \mapsto \{inm \mapsto \textsf{let mk-}\,(v, a) = (instval\,(nm))\,(inm)$ in

.3 $\qquad \textsf{mk-}\,(v, DEF`RealAccess\,(a, \mathrm{INST}))\mid$

.4 $\qquad inm \in \textsf{dom}\ instval\,(nm)\}\mid$

.5 $\qquad nm \in \textsf{dom}\ instval\};$

The operation *GetVlsDef* returns the value definitions contained in class *nm*.

954.0 $GetVlsDef : AS`Name \xrightarrow{o} AS`ValueDef^*$

.1 $GetVlsDef\,(nm) \triangleq$

.2 $\quad$ return $classes\,(nm).vls\text{-}def;$

The operation *GetVlsInit* returns the semantic values of the values in the class *nm*.

955.0 $GetVlsInit : AS`Name \xrightarrow{o} GLOBAL`ValueMap$

.1 $GetVlsInit\,(nm) \triangleq$

.2 $\quad$ return $classes\,(nm).vls\text{-}init;$

The operation *GetExplOps* returns all the explicit operatons in the class *name*

956.0 $GetExplOps : AS`Name \xrightarrow{o} AS`Name \xrightarrow{m} (SEM`ExplOP \times AS`Access)$

.1 $GetExplOps\,(name) \triangleq$

.2 $\quad$ return $classes\,(name).explops;$

The operation *GetImplOps* returns all the explicit operatons in the class *name*

957.0 $GetImplOps : AS`Name \xrightarrow{o} AS`Name \xrightarrow{m} (SEM`ImplOP \times AS`Access)$

.1 $GetImplOps\,(name) \triangleq$

.2 $\quad$ return $classes\,(name).implops;$

The operation *GetAllOps* returns all the operatons in the class *name*

958.0 $GetAllOps : AS`Name \xrightarrow{o} AS`Name \xrightarrow{m} ((SEM`ExplOP \mid SEM`ImplOP) \times AS`Access)$

.1 $GetAllOps\,(name) \triangleq$

.2 $\quad$ return $GetImplOps\,(name) \dagger GetExplOps\,(name);$

The operation *GetAllOpsNmsSupers* returns all the names of operations in the class *name* and in all its super classes.

959.0 $GetAllOpsNmsSupers : AS`Name \xrightarrow{o} AS`Name\text{-set}$

.1 $GetAllOpsNmsSupers\,(name) \triangleq$

.2 $\quad$ return dom $GetAllOps\,(name) \cup \bigcup \{\mathrm{dom}\ GetAllOps\,(nm) \mid nm \in GetAllSupers\,(name)\};$

The operation *LookupAllFnsOpsPolys* looks up a class and operation name and return the function/opration/poly. (Can somebody please find a nice type name for the rhs. function union type below.)

name

960.0 $LookupAllFnsOpsPolys : AS`Name \times AS`Name \xrightarrow{o} \mathbb{B} \times [SEM`CompExplFN \times AS`Access \mid$

.1 $\qquad SEM`ImplFN \times AS`Access \mid$

.2 $\qquad SEM`ExplOP \times AS`Access \mid$

.3 $\qquad SEM`ImplOP \times AS`Access \mid$

.4 $\qquad SEM`ExplPOLY \times AS`Access \mid$

.5 $\qquad SEM`ImplPOLY \times AS`Access]$

.6 $LookupAllFnsOpsPolys\,(clnm, fnnm) \triangleq$

.7 $\quad$ (if $clnm \in \mathrm{dom}\ classes \wedge$

.8 $\qquad fnnm \in \mathrm{dom}\ classes\,(clnm).all\text{-}fns\text{-}ops\text{-}polys$

.9 $\quad$ then return mk- (true, $classes\,(clnm).all\text{-}fns\text{-}ops\text{-}polys\,(fnnm)$)

.10 $\quad$ else return mk- (false, nil));

The operation *GetLocalTps* returns all the types defined in the class *nm*.

961.0 $GetLocalTps : AS`Name \xrightarrow{o} AS`Name \xrightarrow{m} AS`TypeDef$

.1 $GetLocalTps\,(nm) \triangleq$

.2 $\quad$ return $classes\,(nm).localtps;$

The operation *GetRecSel* returns the record selector information of all the record types in class *nm*.

962.0 $GetRecSel : AS`Name \xrightarrow{o} AS`Name \xrightarrow{m} (GLOBAL`RecSel \times AS`Access)$

.1 $GetRecSel\,(nm) \triangleq$

.2 $\quad$ return $classes\,(nm).recsel$

.3 $\textsf{pre}\ \ nm \in \textsf{dom}\ classes$;

The operation *GetLocalHchy* returns the local hierarchy of the class *nm*.

963.0 $GetLocalHchy : AS`Name \xrightarrow{o} AS`Name \xrightarrow{m} AS`Name\textsf{-set}$

.1 $GetLocalHchy\,(nm) \triangleq$

.2 $\quad \textsf{return}\ classes\,(nm).localhchy;$

964.0 $IsClassInit : AS`Name \xrightarrow{o} \mathbb{B}$

.1 $IsClassInit\,(nm) \triangleq$

.2 $\quad \textsf{return}\ nm \in \textsf{dom}\ classes \ \Rightarrow\ classes\,(nm).isinit;$

965.0 $InitClasses : () \xrightarrow{o} AS`Name\textsf{-set}$

.1 $InitClasses\,() \triangleq$

.2 $\quad \textsf{return}\ \{clnm \mid clnm \in \textsf{dom}\ classes \cdot IsClassInit\,(clnm)\};$

966.0 $SetClassInit : AS`Name \times \mathbb{B} \xrightarrow{o} ()$

.1 $SetClassInit\,(nm, val) \triangleq$

.2 $\quad classes(nm).isinit := val;$

This auxiliary operation *nm* computes the super classes names of a class *name*.

967.0 $GetSupers : AS`Name \xrightarrow{o} AS`Name\textsf{-set}$

.1 $GetSupers\,(nm) \triangleq$

.2 $\quad \textsf{return}\ classes\,(nm).inhcon;$

This operation returns the constructor code for the given class. Because constructors can be overloaded the argument list of values is included as a parameter to find the right constructor to use.

968.0 $LookUpConstructor : AS`Name \times SEM`VAL^* \xrightarrow{o} STKM`DebugCmd$

.1 $LookUpConstructor\,(nm, valL) \triangleq$

.2 $\quad \textsf{let}\ constlocal = LookUpConstructorLocal\,(nm, valL)\ \textsf{in}$

.3 $\quad (\textsf{if}\ constlocal \neq \textsf{nil}$

.4 $\quad\ \ \textsf{then return}\ constlocal$

```
    else let supers = GetAllSupers (nm) in
         let conssup =
                  {cl ↦ LookUpConstructorLocal (cl, valL) |
                        cl ∈ supers} ⩥
                  {nil } in
          cases  dom conssup:
            {} → RTERR‘ErrorVal(RTERR‘NOCONSTRUCTOR, nil , nil , []) ,
            {cl} → return conssup (cl),
            - → let mk- (doesthere, child) = ExistsOneChild (dom conssup) in
                  if doesthere
                  then return conssup (child)
                  else RTERR‘ErrorVal(RTERR‘MULT-DEF, nil , nil , [])
         end;
    RTERR‘ErrorVal(RTERR‘NOCONSTRUCTOR, nil , nil , []) )
pre  nm ∈ dom classes ;
```

```
969.0 LookUpConstructorLocal : AS‘Name × SEM‘VAL* -o->
                                   [STKM‘DebugCmd]
LookUpConstructorLocal (nm, valL) △
  (let consts = classes (nm).constructor,
       curlen = len valL,
       origcl = STKM‘GetOrigOldCl (),
       clnm = STKM‘GetCurCl () in
   for all typeL ∈  dom consts
   do if len typeL = curlen
      then if ∀ i ∈ inds valL · SubType (valL (i), typeL (i))
           then let mk- (code, access) = consts (typeL) in
                return code ;
   return nil  );
```

```
970.0 AClass : AS‘Name -o-> 𝔹
AClass (nm) △
  return nm ∈ clsnms
```

```
end STATE
```

Test Suite : rtinfo.ast
Module : STATE

Name	#Calls	Coverage
STATE‘AClass	**undefined**	**undefined**
STATE‘IsDLOp	**undefined**	**undefined**
STATE‘LookUp	**undefined**	**undefined**
STATE‘AddHist	**undefined**	**undefined**
STATE‘IsDObjs	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
STATE‘IsValue	**undefined**	**undefined**
STATE‘SetHchy	**undefined**	**undefined**
STATE‘SubType	**undefined**	**undefined**
STATE‘AccessOk	**undefined**	**undefined**
STATE‘FindType	**undefined**	**undefined**
STATE‘IsAClass	**undefined**	**undefined**
STATE‘IsClient	**undefined**	**undefined**
STATE‘LookUpSD	**undefined**	**undefined**
STATE‘GetAllOps	**undefined**	**undefined**
STATE‘GetInhCon	**undefined**	**undefined**
STATE‘GetRecSel	**undefined**	**undefined**
STATE‘GetSupers	**undefined**	**undefined**
STATE‘GetVlsDef	**undefined**	**undefined**
STATE‘IsDLClass	**undefined**	**undefined**
STATE‘GetClasses	**undefined**	**undefined**
STATE‘GetExplOps	**undefined**	**undefined**
STATE‘GetImplOps	**undefined**	**undefined**
STATE‘GetInstInv	**undefined**	**undefined**
STATE‘GetVlsInit	**undefined**	**undefined**
STATE‘Init-Sigma	**undefined**	**undefined**
STATE‘InsertDLOp	**undefined**	**undefined**
STATE‘LookupHchy	**undefined**	**undefined**
STATE‘AddAbstract	**undefined**	**undefined**
STATE‘GetInhStrct	**undefined**	**undefined**
STATE‘GetInstVars	**undefined**	**undefined**
STATE‘GetLocalTps	**undefined**	**undefined**
STATE‘Get-obj-tab	**undefined**	**undefined**
STATE‘InitClasses	**undefined**	**undefined**
STATE‘IsClassInit	**undefined**	**undefined**
STATE‘ModifyValue	**undefined**	**undefined**
STATE‘PopInitPool	**undefined**	**undefined**
STATE‘RealSubType	**undefined**	**undefined**
STATE‘SetInhStrct	**undefined**	**undefined**
STATE‘AllSuperList	**undefined**	**undefined**
STATE‘CheckGlobInv	**undefined**	**undefined**
STATE‘GetAllSupers	**undefined**	**undefined**
STATE‘GetInitSigma	**undefined**	**undefined**
STATE‘GetLocalHchy	**undefined**	**undefined**
STATE‘GetObjTabLen	**undefined**	**undefined**
STATE‘GetThreadDef	**undefined**	**undefined**
STATE‘Init-classes	**undefined**	**undefined**
STATE‘IsInObjScope	**undefined**	**undefined**
STATE‘LookUpPermis	**undefined**	**undefined**
STATE‘LookUpStatic	**undefined**	**undefined**
STATE‘LookUpThread	**undefined**	**undefined**
STATE‘PushInitPool	**undefined**	**undefined**
STATE‘ReturnLookUp	**undefined**	**undefined**

Name	#Calls	Coverage
STATE‘SetClassInit	**undefined**	**undefined**
STATE‘SetThreadDef	**undefined**	**undefined**
STATE‘TranslateAST	**undefined**	**undefined**
STATE‘GetInstVarsTp	**undefined**	**undefined**
STATE‘InitClassName	**undefined**	**undefined**
STATE‘InsertDLClass	**undefined**	**undefined**
STATE‘IsSubTypeName	**undefined**	**undefined**
STATE‘LookOpFctPoly	**undefined**	**undefined**
STATE‘LookUpHistory	**undefined**	**undefined**
STATE‘ModifyValueId	**undefined**	**undefined**
STATE‘SetPermission	**undefined**	**undefined**
STATE‘ExistsOneChild	**undefined**	**undefined**
STATE‘GetAllClassNms	**undefined**	**undefined**
STATE‘GetDefaultCons	**undefined**	**undefined**
STATE‘GetInstInitVal	**undefined**	**undefined**
STATE‘GetSemObjInTab	**undefined**	**undefined**
STATE‘InitTheClasses	**undefined**	**undefined**
STATE‘InitializeGSGV	**undefined**	**undefined**
STATE‘LookOpFctPoly’	**undefined**	**undefined**
STATE‘Lookup-obj-tab	**undefined**	**undefined**
STATE‘SetInstInitVal	**undefined**	**undefined**
STATE‘SetInstanceVar	**undefined**	**undefined**
STATE‘SetThreadField	**undefined**	**undefined**
STATE‘GetClFromObjRef	**undefined**	**undefined**
STATE‘GetNameOfObjRef	**undefined**	**undefined**
STATE‘InitStaticInsts	**undefined**	**undefined**
STATE‘RegisterDLClass	**undefined**	**undefined**
STATE‘SetPerThreadDef	**undefined**	**undefined**
STATE‘UpdateHistCount	**undefined**	**undefined**
STATE‘LookUpOverloaded	**undefined**	**undefined**
STATE‘SetTypeJudgement	**undefined**	**undefined**
STATE‘LookUpConstructor	**undefined**	**undefined**
STATE‘LookUpOverInClass	**undefined**	**undefined**
STATE‘ModifyInstanceVar	**undefined**	**undefined**
STATE‘RemoveClassValues	**undefined**	**undefined**
STATE‘SetObjectThreadId	**undefined**	**undefined**
STATE‘GetAllOpsNmsSupers	**undefined**	**undefined**
STATE‘GetFirstCIDOfFctOp	**undefined**	**undefined**
STATE‘IsSuperClassesInit	**undefined**	**undefined**
STATE‘UnsetTypeJudgement	**undefined**	**undefined**
STATE‘EvalFctTypeInstExpr	**undefined**	**undefined**
STATE‘EvalStateDesignator	**undefined**	**undefined**
STATE‘LookStaticOpFctPoly	**undefined**	**undefined**
STATE‘CheckIfAbstractClass	**undefined**	**undefined**
STATE‘LookupAllFnsOpsPolys	**undefined**	**undefined**
STATE‘GiveNextRefCountInTab	**undefined**	**undefined**
STATE‘global-obj-tab-insert	**undefined**	**undefined**

Name	#Calls	Coverage
STATE‘CheckInstanceInvariant	**undefined**	**undefined**
STATE‘LookUpConstructorLocal	**undefined**	**undefined**
STATE‘GetObjRefAndClFromThreadId	**undefined**	**undefined**
Total Coverage		**0%**

1.17 Classes

The module CLASS contains all functions and operations definitions related to the translation and handling of classes.

module *CLASS*

imports

971.0 from *AS* all ,

972.0 from *CI* all ,

973.0 from *AUX*

974.0 functions $ConstructDoubleName : AS`Name \times AS`Name \rightarrow AS`Name$

975.0 operations $ErrorEmptyOp$: char* $\xrightarrow{o}$ () renamed $ErrorEmptyOp$;

.1 $MkEmptyBlkEnv : SEM`Permission \xrightarrow{o} SEM`BlkEnv$,

976.0 from *DEF* all ,

977.0 from *PAT* all ,

978.0 from *POS* all ,

979.0 from *REP* all ,

980.0 from *SEM* all ,

981.0 from *CMPL* all ,

982.0 from *CPAT* all ,

983.0 from *STKM* all ,

984.0 from *TIME*

985.0 operations $GetCompilingTime : () \xrightarrow{o} \mathbb{B}$;

.1 $SetCompilingTime : \mathbb{B} \xrightarrow{o} ()$,

986.0 from *CEXPR* all ,

987.0 from *CSTMT* all ,

988.0 from *DEBUG* all ,

989.0 from *RTERR* all ,

990.0 from *STATE* all ,

991.0 from *GLOBAL* all ,

992.0 from *SCHDTP* all ,

993.0 from *DEBUGTP*

994.0 types *BreakInfo* ,

995.0 from $INSTRTP$ all ,

996.0 from $TIMEMAP$ all ,

997.0 from $SETTINGS$

998.0 operations $DTC : () \xrightarrow{o} \mathbb{B}$ renamed DTC ,

999.0 from $TIMEPARSER$ all

exports

1000.0 functions $TransLocalHchy : AS`Name \rightarrow AS`Name \xrightarrow{m} AS`Name$-set

1001.0 operations $InitGV : AS`ValueDef^* \xrightarrow{o} GLOBAL`ValueMap$;
.1 $ExtOpDom : AS`OpDef \xrightarrow{o} AS`Type^*$;
.2 $GenInsMap : AS`Name \xrightarrow{o} ()$;
.3 $TransSyncs : AS`Document \xrightarrow{o} ()$;
.4 $EvalInhStrct : () \xrightarrow{o} ()$;
.5 $TransHierarchy : () \xrightarrow{o} ()$;
.6 $CreateConstructor : AS`Name \times AS`InstAssignDef^* \times AS`Name^* \times AS`Name \xrightarrow{m} AS`OpDef \xrightarrow{o}$
.7 $AS`Type^* \xrightarrow{m} (STKM`DebugCmd \times AS`Access)$

definitions

1.17.1 Initialisiation of Instance Variables

This has been merge into the InitClassName operation from the STATE module because of the split between static and non-static instance variables. Statics need to be initialised first! InitClVls: AS'Name * seq of AS'InstAssignDef ==¿ GLOBAL'ValueMap

This has been moved to the STATE module because of static initialisers in Java InitStaticInsts: seq of AS'InstAssignDef * bool ==¿ map AS'Name to (SEM'VAL * AS'Access)

The operation *GenInsMap* introduces the instance variables into the scope. If the instance variables are not initialised the value *UNDEF* is assigned. Thus, the operation updates the field *inst_init_val*.
operations

1002.0 $GenInsMap : AS`Name \xrightarrow{o} ()$
.1 $GenInsMap\,(nm) \triangleq$
.2 let $allsupers = STATE`GetAllSupers\,(nm)$,
.3 $instvars = STATE`GetInstVars\,(nm)$ in
.4 let $own = \{nm \mapsto \{ins \mapsto$ mk- (mk-$SEM`UNDEF\,(), access) \mid$
.5 mk-$AS`InstAssignDef$
.6 (
.7 mk-$AS`AssignDef\,(ins, \text{-}, \text{-}, \text{-}), access, \text{-}, \text{-}) \in$
.8 elems $instvars\}\}$,
.9 $supers =$ merge $\{STATE`GetInstInitVal\,(cl) \mid cl \in allsupers\}$ in
.10 $STATE`SetInstInitVal(nm, own \dagger supers)$;

In this functions it is assumed that no cyclic dependencies are represented in this class inheritance. We know that this is the case because *STATE‘TransHierarchy* has been called before in the operation *STATE‘TranslateAST*.

The operation *InitGV* computes the value definitions of a class. By pushing the just evaluated value to the evaluation stack, a value can depend on the values which have been defined before it in the value list.

```
1003.0 InitGV : AS‘ValueDef* -o-> GLOBAL‘ValueMap
InitGV (val-l) ≜
  if val-l ≠ []
  then (dcl res-m : GLOBAL‘ValueMap := {↦};
        let mk-AS‘ValueDef (pat, tp, exp-e, access, -, -) = hd val-l in
        (let mk- (eval-state, res) = DEBUG‘EvalUninterruptedCmd (exp-e, [], [], "InitofGlobalValues"),
             exp-v = if is-STKM‘Success (eval-state)
                     then res
                     else undefined in
         (if DTC ()
          then if (tp ≠ nil ) ∧ (¬ STATE‘SubType (exp-v, tp))
               then error
               else skip;
          let env-s = PAT‘PatternMatch (CPAT‘P2P (pat), exp-v) in
          if env-s ≠ {}
          then let env ∈ env-s in
               (for all id ∈ dom env.id-m
                do let mk-SEM‘ValTp (val, -) = env.id-m (id) in
                   res-m := res-m†{id ↦ mk- (val, DEF‘RealAccess (access, INST))};
                STKM‘PushBlkEnv(env) ;
                res-m := res-m † InitGV (tl val-l);
                STKM‘PopBlkEnv() )
          else error));
        return res-m )
  else return {↦} ;
```

1.17.2 Translation of the Class Hierarchy

```
1004.0 TransHierarchy : () -o-> ()
TransHierarchy () ≜
  let classes = STATE‘GetClasses () in
  let clhchy = {name ↦ STATE‘GetSupers (name) | name ∈ dom classes} in
  let exp = {name | name ∈ dom classes · clhchy (name) = {}} in
  let clhchy1 = ExpandHierarchy (clhchy, exp) in
  if OkHierarchy (clhchy1)
  then STATE‘SetHchy(clhchy1)
  else ErrorEmptyOp("Circulardependency")
```

The next functions are auxiliary functions to the operation *TransHiearchy*.

The following function *ExpandHierarchy* expands the range of every entry in the hierarchy table *hchy'* such that for every class in the range, which does not inherit from other superclasses, its superclasses will be added to the range. The second argument *done* describes all the classes in the hiearchy which is fully expanded. The function returns the fully expanded hiearchy table.

The function works as follows: The function is recursive, if the the domain of the hierarchy table *hchy'* is the same as the set *done*, the table *hchy'* is fully updated, because the set *done* contains the names of the classes which have been fully expanded in the table *hchy'*. Otherwise, for each class which is not fully expanded, a hierarchy table is constructed which expands the hierarchy tables for these names, using the function *ExpandNextLevel*.

For the newly expanded table the newly fully expanded names is computed with the function *NewlyDone*, and the function itself *ExpandHierarchy* is called again.

functions

1005.0 $\mathit{ExpandHierarchy} : \mathit{GLOBAL}`\mathit{Hierarchy} \times \mathit{AS}`\mathit{Name}\text{-}\mathsf{set} \rightarrow \mathit{GLOBAL}`\mathit{Hierarchy}$

.1 $\mathit{ExpandHierarchy}\,(\mathit{hchy}', \mathit{done}) \triangleq$

.2 $\quad \mathsf{if\ dom}\ \mathit{hchy}' = \mathit{done}$

.3 $\quad \mathsf{then}\ \mathit{hchy}'$

.4 $\quad \mathsf{else\ let}\ \mathit{not}\text{-}\mathit{done} = \mathsf{dom}\ \mathit{hchy}' \setminus \mathit{done},$

.5 $\qquad\qquad \mathit{hchy}'' = \{\mathit{nm} \mapsto \mathit{ExpandNextLevel}\,(\mathit{hchy}'\,(\mathit{nm}), \mathit{hchy}', \{\}) \mid$

.6 $\qquad\qquad\qquad\qquad \mathit{nm} \in \mathit{not}\text{-}\mathit{done}\}\ \mathsf{in}$

.7 $\qquad \mathsf{let}\ \mathit{newly}\text{-}\mathit{done} = \mathit{NewlyDone}\,(\mathit{hchy}'', \mathit{done})\ \mathsf{in}$

.8 $\qquad \mathit{ExpandHierarchy}\,(\mathit{hchy}'' \dagger (\mathit{not}\text{-}\mathit{done} \lhd \mathit{hchy}'), \mathit{newly}\text{-}\mathit{done} \cup \mathit{done});$

The function *ExpandNextLevel* takes three arguments:

- to_exp: is the set of class names, which the function should expand such that the set describes all the class names which is in the dependency tree of the classes.
- hchy: is the hierarchy table build to far.
- in_hchy: is a set of class names, which already is in the dependency tree. This argument is included, in order to be able to check whether the dependency chain is cyclic.

1006.0 $\mathit{ExpandNextLevel} : \mathit{AS}`\mathit{Name}\text{-}\mathsf{set} \times \mathit{GLOBAL}`\mathit{Hierarchy} \times \mathit{AS}`\mathit{Name}\text{-}\mathsf{set} \rightarrow$

.1 $\qquad\qquad \mathit{AS}`\mathit{Name}\text{-}\mathsf{set}$

.2 $\mathit{ExpandNextLevel}\,(\mathit{to}\text{-}\mathit{exp}, \mathit{hchy}, \mathit{in}\text{-}\mathit{hchy}) \triangleq$

.3 $\quad \bigcup \{\mathit{ExpCl}\,(\mathit{nm}, \mathit{hchy}, \mathit{in}\text{-}\mathit{hchy}) \cup \{\mathit{nm}\} \mid \mathit{nm} \in \mathit{to}\text{-}\mathit{exp}\}$

The function *ExpCl* computes the superclasses of a class name. It takes three arguments:

- nm: the name of the class which super classes are to be computed.

- hchy': the hierarchy table.
- in_hchy: the set of names which already are dependency chain.

operations

1007.0 $\mathit{ExpCl} : \mathit{AS}`\mathit{Name} \times \mathit{GLOBAL}`\mathit{Hierarchy} \times \mathit{AS}`\mathit{Name}\text{-set} \xrightarrow{o} \mathit{AS}`\mathit{Name}\text{-set}$

.1 $\mathit{ExpCl}\,(nm, hchy', in\text{-}hchy) \triangleq$

.2 $(\text{if } nm \in \text{dom } hchy'$

.3 $\text{then } (\text{if } hchy'\,(nm) = \{\}$

.4 $\text{then return } in\text{-}hchy$

.5 $\text{else if } nm \in in\text{-}hchy$

.6 $\text{then } \mathit{RTERR}`\mathit{InitError}(\mathit{RTERR}`\mathit{CIRC\text{-}CL\text{-}DEPENDENCY}, nm.cid)$

.7 $\text{else let } nextlevel = \mathit{STATE}`\mathit{GetSupers}\,(nm) \text{ in}$

.8 $\text{return } \mathit{ExpandNextLevel}\,(nextlevel, hchy', in\text{-}hchy \cup \{nm\}) \cup$

.9 $\{nm\}\ )$

.10 $\text{else } \mathit{RTERR}`\mathit{InitError}(\mathit{RTERR}`\mathit{CLNM\text{-}NOT\text{-}DEFINED}, nm.cid)\,;$

.11 $\text{return } \{\}\ )$

The function *NewlyDone* computes the set of names which is fully expanded in the hierarchy table *hchy*. The second argument *done* is the set of names which currently are fully expanded in the hierarchy tree.

functions

1008.0 $\mathit{NewlyDone} : \mathit{GLOBAL}`\mathit{Hierarchy} \times \mathit{AS}`\mathit{Name}\text{-set} \to \mathit{AS}`\mathit{Name}\text{-set}$

.1 $\mathit{NewlyDone}\,(hchy, done) \triangleq$

.2 $\{nm \mid nm \in \text{dom } hchy \cdot$

.3 $\forall\, cl \in hchy\,(nm) \cdot cl \in done\};$

1009.0 $\mathit{OkHierarchy} : \mathit{GLOBAL}`\mathit{Hierarchy} \to \mathbb{B}$

.1 $\mathit{OkHierarchy}\,(clhchy) \triangleq$

.2 $\forall\, cl \in \text{dom } clhchy \cdot$

.3 $cl \notin clhchy\,(cl)$

1.17.3 Translation of Inheritance Structure

operations

1010.0 $\mathit{EvalInhStrct} : () \xrightarrow{o} ()$

.1 $\mathit{EvalInhStrct}\,() \triangleq$

.2 $(\text{dcl } inhstrct : \mathit{GLOBAL}`\mathit{InhStrct} := \{\mapsto\};$

.3 $\text{let } classes = \mathit{STATE}`\mathit{GetClasses}\,() \text{ in}$

.4 $\text{for all } clname \in \text{ dom } classes$

.5 $\quad$ do $inhstrct := inhstrct \dagger \{clname \mapsto OrderOfProcess\,(clname)\}$;
.6 $\quad STATE`SetInhStrct(inhstrct)\ )$

The next functions computes the order of type checking a class and its super classes.
functions

1011.0 $\quad OrderOfProcess : AS`Name \rightarrow AS`Name\text{-set}^*$
.1 $\quad OrderOfProcess\,(nm) \triangleq$
.2 $\quad$ (let $supers = STATE`GetSupers\,(nm)$ in
.3 $\quad OrderOfProcess\text{-}Aux\,([\{\}], supers))$;

1012.0 $\quad OrderOfProcess\text{-}Aux : (AS`Name\text{-set})^* \times AS`Name\text{-set} \rightarrow$
.1 $\quad AS`Name\text{-set}^*$
.2 $\quad OrderOfProcess\text{-}Aux\,(order, to\text{-}process) \triangleq$
.3 $\quad$ (if $to\text{-}process = \{\}$
.4 $\quad$ then $order$
.5 $\quad$ else let $new\text{-}order = [to\text{-}process] \curvearrowright [order\,(i) \setminus to\text{-}process \mid i \in \text{inds } order]$,
.6 $\quad supers = \bigcup \{STATE`GetSupers\,(sb) \mid sb \in to\text{-}process\}$ in
.7 $\quad OrderOfProcess\text{-}Aux\,(new\text{-}order, supers))$;

1.17.4 Translation of The Local Hierarchy

The operation *TransLocalHchy* creates the local hierarchy of a class *nm*.

1013.0 $\quad TransLocalHchy : AS`Name \rightarrow AS`Name \xrightarrow{m} AS`Name\text{-set}$
.1 $\quad TransLocalHchy\,(nm) \triangleq$
.2 $\quad$ let $supers = STATE`GetSupers\,(nm)$ in
.3 $\quad \{nm \mapsto supers\} \dagger$
.4 $\quad$ merge $\{TransLocalHchy\,(cl) \mid cl \in supers\}$

1.17.5 Hierarchy Look Up Auxiliary Functions/Operations

Constructors can be overloaded and this is taken into account in the CreateConstructor operation. In order to deal with the overloading a mapping from sequences of types to pairs of instruction code and access information is returned.
operations

```
1014.0 CreateConstructor : AS`Name × AS`InstAssignDef* × AS`Name* × AS`Name -m->
AS`OpDef -o->
                AS`Type* -m-> (STKM`DebugCmd × AS`Access)

CreateConstructor (curcls, instvars, supercls, opm) △
  (dcl constrmap : AS`Type* -m-> (STKM`DebugCmd × AS`Access) := {↦},
       sp : STKM`SubProgram := [];
   for cls in supercls
   do sp := sp ⌒ [mk-INSTRTP`PUSHCLNMCUROBJ (cls, cls),
            mk-INSTRTP`INITCLASS (cls, 0),
            mk-INSTRTP`POPCLNMCUROBJ ()] ⌒
           [];
   for mk-AS`InstAssignDef (mk-AS`AssignDef (nm, tp, Iinit, -), access,
                            static, -)
   in instvars
   do if Iinit ≠ nil ∧ ¬ static
      then sp := CMPL`SetContext (Iinit.cid, false) ⌒
                 [mk-INSTRTP`CONTEXT (Iinit.cid, false)] ⌒
                 sp ⌒
                 CEXPR`E2I (Iinit) ⌒
                 [mk-INSTRTP`DTC (tp), mk-INSTRTP`BINDINSTVAR (nm)]
      elseif static
      then sp := sp⌒[mk-INSTRTP`LOOKUPSTATIC (AUX`ConstructDoubleName (curcls, nm)),
                 mk-INSTRTP`BINDINSTVAR (nm)];
   constrmap := {[] ↦ mk- (mk-STKM`DebugCmd
                            (
                             [mk-INSTRTP`POP (1)] ⌒
                             sp ⌒
                             [mk-INSTRTP`EOCL ()]),
                          PUBLIC_AS)};
   for all opdef ∈ rng opm
   do if opdef.constr
      then let tpl = ExtOpDom (opdef) in
           (constrmap := constrmap †
                          {tpl ↦ mk-
                             (
                              mk-STKM`DebugCmd
                                (
                                 sp ⌒
                                 [mk-INSTRTP`LOOKUP (opdef.nm)] ⌒
                                 [mk-INSTRTP`SWAP ()] ⌒
                                 [mk-INSTRTP`APPLY ()] ⌒
                                 (if tpl = []
                                  then [mk-INSTRTP`POP (1)]
                                  else []) ⌒
                                 [mk-INSTRTP`EOCL ()]),
                              opdef.access)});
   return constrmap );
```

1015.0 $\mathit{ExtOpDom} : AS`\mathit{OpDef} \xrightarrow{o} AS`\mathit{Type}^*$
.1 $\mathit{ExtOpDom}\,(\mathit{opdef}) \triangleq$
.2 cases true:
.3 (is-$AS`\mathit{ExplOpDef}\,(\mathit{opdef})) \to$ return $\mathit{opdef.tp.opdom}$,
.4 (is-$AS`\mathit{ImplOpDef}\,(\mathit{opdef})$),
.5 (is-$AS`\mathit{ExtExplOpDef}\,(\mathit{opdef})) \to$
.6 let $\mathit{ptp\text{-}l} = \mathit{opdef.partps}$ in
.7 (dcl $\mathit{tp\text{-}l} : AS`\mathit{Type}^* := []$;
.8 for mk-$AS`\mathit{PatTypePair}\,(\mathit{pats}, \mathit{tp}, \text{-})$ in $\mathit{ptp\text{-}l}$
.9 do $\mathit{tp\text{-}l} := \mathit{tp\text{-}l} \curvearrowright [\mathit{tp} \mid i \in$ inds $\mathit{pats}]$;
.10 return $\mathit{tp\text{-}l}$)
.11 end;

1.17.6 Translation of Synchronisation

1016.0 $\mathit{TransSyncs} : AS`\mathit{Document} \xrightarrow{o} ()$
.1 $\mathit{TransSyncs}\,(\mathit{cs}) \triangleq$
.2 let $\mathit{order} = \mathit{GetPermissionOrder}\,(\mathit{cs})$,
.3 $\mathit{class\text{-}m} = \{\mathit{class.nm} \mapsto \mathit{class.defs} \mid \mathit{class} \in$ elems $\mathit{cs}\}$ in
.4 for clnm in order
.5 do $(\mathit{CMPL`SetClMod}(\mathit{clnm})$;
.6 $\mathit{TransSyncsForOneClass}(\mathit{clnm}, \mathit{class\text{-}m}\,(\mathit{clnm}))$);

The operation GetPermissionOrder returns a topologic sort of all the clases with respect to inheritance. Given the following example:

```
   A  D
  / \/
 B   C
    / \
   F   G
```

The result will might be [A,D,B,C,F,G].

The following variables are used:

res The result of the operation

dep_m map from class name to it immediate parents. For example:
{ A —-¿ {}, D —-¿ {}, B —-¿{A}, C —-¿ {A,D}, . . . }

no_link Set of all classes with no un-handled super classes.

rem_m equal to dep_m with the classes from no_link removed. That is classes that still have un-handled super classes new_m equals to rem_m with the classes from no_link removed from the values of the maps.

1017.0 $GetPermissionOrder : AS`Document \xrightarrow{o} AS`Name^*$

.1 $GetPermissionOrder\,(cs) \triangleq$

.2 (dcl $res : AS`Name^* := [\,]$,

.3 $dep\text{-}m : AS`Name \xrightarrow{m} AS`Name\text{-}\mathsf{set} := \{class.nm \mapsto \mathsf{elems}\ class.supercls \mid class \in \mathsf{elems}\ cs\}$;

.4 while $dep\text{-}m \neq \{\mapsto\}$

.5 do let $no\text{-}link = \{cl\text{-}nm \mid cl\text{-}nm \in \mathsf{dom}\ dep\text{-}m \cdot dep\text{-}m\,(cl\text{-}nm) = \{\}\}$,

.6 $rem\text{-}m = \{cl\text{-}nm \mapsto dep\text{-}m\,(cl\text{-}nm) \mid cl\text{-}nm \in \mathsf{dom}\ dep\text{-}m \cdot dep\text{-}m\,(cl\text{-}nm) \neq \{\}\}$,

.7 $new\text{-}m = \{cl\text{-}nm \mapsto dep\text{-}m\,(cl\text{-}nm) \setminus no\text{-}link \mid cl\text{-}nm \in \mathsf{dom}\ rem\text{-}m\}$ in

.8 $(res := res \frown set2seq\,(no\text{-}link)$;

.9 $dep\text{-}m := new\text{-}m)$;

.10 return res)

functions

1018.0 $set2seq : AS`Name\text{-}\mathsf{set} \to AS`Name^*$

.1 $set2seq\,(elms) \triangleq$

.2 if $elms = \{\}$

.3 then $[\,]$

.4 else let $elm \in elms$ in

.5 $[elm] \frown set2seq\,(elms \setminus \{elm\})$

operations

1019.0 $TransSyncsForOneClass : AS`Name \times [AS`Definitions] \xrightarrow{o} ()$

.1 $TransSyncsForOneClass\,(clnm, defs) \triangleq$

.2 if $defs = \mathsf{nil}$

.3 then skip

.4 else let mk-$AS`Definitions\,(-,-,-,opm,-,-,syncs,thread,-) = defs$,

.5 $superops = STATE`GetAllOpsNmsSupers\,(clnm) \setminus \mathsf{dom}\ STATE`GetAllOps\,(clnm)$ in

.6 $(TransThreadDef\,(clnm, thread)$;

.7 for $sync$ in $syncs$

.8 do $TransSyncDef\,(clnm, sync, \mathsf{dom}\ opm)$;

.9 for all $opnm \in superops$

.10 do $STATE`SetPermission(clnm, opnm, MergePermissionWithSupers\,(clnm, opnm, [\,])))$;

1020.0 $GetInhThread : AS`Name \xrightarrow{o} [STKM`SubProgram \times [\mathbb{N}]]$

.1 $GetInhThread\,(clnm) \triangleq$

.2 let $td = STATE`GetThreadDef\,(clnm)$ in

.3 if $td \notin \{\mathsf{nil}, \mathsf{mk\text{-}}(\mathsf{nil}, \mathsf{nil})\}$

.4 then return td

.5 else let $supers = STATE`GetInhCon\,(clnm)$ in
.6 let $super\text{-}threads = \{STATE`GetThreadDef\,(sclnm) \mid sclnm \in supers \cdot$
.7 $STATE`GetThreadDef\,(sclnm) \notin \{$nil , mk- (nil , nil)$\}\}$ in
.8 if card $super\text{-}threads > 1$
.9 then $(RTERR`InitError(RTERR`MULT\text{-}THREAD\text{-}INH, clnm.cid)$;
.10 return nil)
.11 elseif card $super\text{-}threads = 0$
.12 then return nil
.13 else let $\{t\} = super\text{-}threads$ in
.14 return t ;

1021.0 $TransThreadDef : AS`Name \times [AS`ThreadDef] \xrightarrow{o} ()$
.1 $TransThreadDef\,(clnm, Def) \triangleq$
.2 if $Def =$ nil
.3 then $STATE`SetThreadField(clnm, GetInhThread\,(clnm))$
.4 elseif is-$AS`PerObl\,(Def)$
.5 then $STATE`SetPerThreadDef(clnm, Def)$
.6 else $STATE`SetThreadDef(clnm, CSTMT`S2I\,(Def))$;

1022.0 $TransSyncDef : AS`Name \times AS`SyncDef \times AS`Name$-set $\xrightarrow{o} ()$
.1 $TransSyncDef\,(nm, Def, opnm\text{-}s) \triangleq$
.2 if is-$AS`Permission\,(Def)$
.3 then $TransPermission(nm, Def, opnm\text{-}s)$
.4 else $TransMutex(nm, Def, opnm\text{-}s)$;

Note that when translating a permission predicate, we ensure that no time increment instructions are generated. This is because when executing a permission predicate time will be incremented elsewhere by the task switching constant.

1023.0 $TransPermission : AS`Name \times AS`Permission \times AS`Name$-set $\xrightarrow{o} ()$
.1 $TransPermission\,(nm,$ mk-$AS`Permission\,(spec, guard, \text{-}), opnm\text{-}s) \triangleq$
.2 let $old\text{-}compiling = TIME`GetCompilingTime\,()$ in
.3 $(TIME`SetCompilingTime($false$)$;
.4 let $guard\text{-}code = CEXPR`E2I\,(guard)$ in
.5 $InstallPermission(nm, spec, guard\text{-}code, opnm\text{-}s)$;
.6 $TIME`SetCompilingTime(old\text{-}compiling)\,)$;

1024.0 $InstallPermission : AS`Name \times AS`Name \times STKM`SubProgram \times AS`Name$-set $\xrightarrow{o} ()$
.1 $InstallPermission\,(clnm, opnm, code, local\text{-}ops) \triangleq$
.2 let $nm =$ mk-$AS`Name\,([$hd $clnm.ids,$ hd $opnm.ids], opnm.cid)$,
.3 $existing\text{-}code = STATE`LookUpPermis\,(nm)$,

```
      res-code =
          if existing-code ≠ nil
          then let no-push-code = tl (existing-code (1, ..., len existing-code − 1)) in
               MergePermission (code, no-push-code)
          elseif opnm ∈ local-ops
          then code
          else MergePermissionWithSupers (clnm, opnm, code),
      push-code =
          [mk-INSTRTP‘PUSHCLNMCUROBJ (clnm, clnm)] ↷
          res-code ↷
          [mk-INSTRTP‘POPCLNMCUROBJ ()] in
  STATE‘SetPermission(clnm, opnm, push-code) ;
```

```
1025.0  MergePermission : STKM‘SubProgram × STKM‘SubProgram →o STKM‘SubProgram
  MergePermission (prog1, prog2) △
    if prog1 = []
    then return prog2
    elseif prog2 = []
    then return prog1
    else return CEXPR‘ConcIfThenElse (prog1, prog2, [mk-INSTRTP‘PUSH (mk-SEM‘BOOL (false))])
;
```

```
1026.0  MergePermissionWithSupers : AS‘Name × AS‘Name × STKM‘SubProgram →o STKM‘SubProgram
  MergePermissionWithSupers (clnm, opnm, prog) △
    let supers = STATE‘GetSupers (clnm) in
    (dcl res : STKM‘SubProgram := prog,
         superFound : B := false;
     for all supernm ∈ supers
     do let super-code = STATE‘LookUpPermis (mk-AS‘Name ([hd supernm.ids, hd opnm.ids],
                                                            opnm.cid)) in
        if super-code ≠ nil
        then if superFound
             then RTERR‘Error(RTERR‘OP-DEF-IN-MULTSUPERS, nil , nil , "")
             else (superFound := true;
                   res := MergePermission (res, super-code));
     return res );
```

```
1027.0  TransMutex : AS‘Name × AS‘Mutex × AS‘Name-set →o ()
  TransMutex (clnm, mk-AS‘Mutex (ops, cid), local-ops) △
    let old-compiling = TIME‘GetCompilingTime () in
    (TIME‘SetCompilingTime(false) ;
     let all-ops = if ops = nil
                   then STATE‘GetAllOpsNmsSupers (clnm)
                   else elems ops,
```

```
        pred = mk-AS`BinaryExpr (mk-AS`ActiveExpr (set2seq (all-ops), CI`NilContextId),
                                 EQ, mk-AS`RealLit (0, CI`NilContextId), CI`NilContextId),
        code = CEXPR`E2I (pred) in
    for all opnm ∈ all-ops
    do InstallPermission(clnm, opnm, code, local-ops) ;
    TIME`SetCompilingTime(old-compiling) )

end CLASS
```

Test Suite : rtinfo.ast
Module : CLASS

Name	**#Calls**	**Coverage**
CLASS‘ExpCl	**undefined**	**undefined**
CLASS‘InitGV	**undefined**	**undefined**
CLASS‘set2seq	**undefined**	**undefined**
CLASS‘ExtOpDom	**undefined**	**undefined**
CLASS‘GenInsMap	**undefined**	**undefined**
CLASS‘NewlyDone	**undefined**	**undefined**
CLASS‘TransMutex	**undefined**	**undefined**
CLASS‘TransSyncs	**undefined**	**undefined**
CLASS‘OkHierarchy	**undefined**	**undefined**
CLASS‘EvalInhStrct	**undefined**	**undefined**
CLASS‘GetInhThread	**undefined**	**undefined**
CLASS‘TransSyncDef	**undefined**	**undefined**
CLASS‘OrderOfProcess	**undefined**	**undefined**
CLASS‘TransHierarchy	**undefined**	**undefined**
CLASS‘TransLocalHchy	**undefined**	**undefined**
CLASS‘TransThreadDef	**undefined**	**undefined**
CLASS‘ExpandHierarchy	**undefined**	**undefined**
CLASS‘ExpandNextLevel	**undefined**	**undefined**
CLASS‘MergePermission	**undefined**	**undefined**
CLASS‘TransPermission	**undefined**	**undefined**
CLASS‘CreateConstructor	**undefined**	**undefined**
CLASS‘InstallPermission	**undefined**	**undefined**
CLASS‘GetPermissionOrder	**undefined**	**undefined**
CLASS‘OrderOfProcess-Aux	**undefined**	**undefined**
CLASS‘TransSyncsForOneClass	**undefined**	**undefined**
CLASS‘MergePermissionWithSupers	**undefined**	**undefined**
Total Coverage		**0%**

1.18 Definitions

The module DEF contains alle functions and operations related to definitions translation.
module *DEF*

imports

1028.0 from AS all ,

1029.0 from CI all ,

1030.0 from AUX all ,

1031.0 from PAT all ,

1032.0 from POS all ,

1033.0 from REP all ,

1034.0 from SEM all ,

1035.0 from $CMPL$ all ,

1036.0 from $CPAT$ all ,

1037.0 from $FREE$ all ,

1038.0 from $STKM$ all ,

1039.0 from $CLASS$ all ,

1040.0 from $INSTR$ all ,

1041.0 from $RTERR$ all ,

1042.0 from $STATE$ all ,

1043.0 from $GLOBAL$ all ,

1044.0 from $MANGLE$ all ,

1045.0 from $SCHDTP$ all ,

1046.0 from $INSTRTP$ all ,

1047.0 from $TIMEMAP$ all ,

1048.0 from $TIMEPARSER$ all

exports all

definitions
types

1049.0 $FnTuple = (AS`Name \xrightarrow{m} (SEM`CompExplFN \times AS`Access)) \times (AS`Name \xrightarrow{m} (SEM`ImplFN \times AS`Access)) \times (AS`Name \xrightarrow{m} (SEM`ExplPOLY \times AS`Access)) \times (AS`Name \xrightarrow{m} (SEM`ImplPOLY \times AS`Access));$

1050.0 $OpTuple = (AS`Name \xrightarrow{m} (SEM`ExplOP \times AS`Access)) \times (AS`Name \xrightarrow{m} (SEM`ImplOP \times AS`Access))$

1.18.1 Definitions Translation

operations

1051.0 $ReadClasses : AS`Class \xrightarrow{o} AS`Name \xrightarrow{m} GLOBAL`SigmaClass$

.1 $ReadClasses\,(\text{mk-}AS`Class\,(nm, supercls, defs, useslib, \text{-})) \triangleq$

.2 $(\text{dcl } sigmacl : GLOBAL`SigmaClass := EmptySigmaClass\,();$

```
CMPL`SetClMod(nm) ;
CMPL`ResetProgramTable(nm) ;
sigmacl.inhcon := TransInheritance (supercls);
if defs = nil
then return {nm ↦ sigmacl} ;
let mk-AS`Definitions (typem, valuem, fnm, opm, as-instvars,
                       timevars, syncs, threads, -) =
        defs in
(let mk- (instvars, instinv, instvars-tp) = TransInstVars (as-instvars) in
 (sigmacl.instvars := instvars;
  sigmacl.instvars-tp := instvars-tp;
  sigmacl.inst-inv := instinv;
  sigmacl.inst-init-val := {↦});
 sigmacl.vls-def := valuem;
 sigmacl.vls-init := {↦};
 let mk- (efn, ifn, epf, ipf) = TransFnMap (nm, fnm) in
 (sigmacl.explfns := efn;
  sigmacl.implfns := ifn;
  sigmacl.explpolys := epf;
  sigmacl.implpolys := ipf);
 let overloaded = TransOverloaded (opm, fnm),
     mk- (eop, iop) = TransOpMap (nm, opm) in
 (sigmacl.explops := eop;
  sigmacl.implops := iop;
  sigmacl.overloaded := overloaded);
 let prepostfns = CreateOperationPrePostFns (nm, opm, nil ) in
 (sigmacl.explfns := sigmacl.explfns †
                     CreateInvs (nm, typem) †
                     prepostfns;
  sigmacl.localtps := UpdateTypeDefs (typem, CMPL`GetClMod ());
  sigmacl.recsel := TransLocalRecSel (typem));
 sigmacl.all-fns-ops-polys := MergeFnsOpsPolys (sigmacl);
 sigmacl.localhchy := {↦};
 sigmacl.statics := ExtractStaticMembers (fnm, opm, sigmacl);
 sigmacl.defaultcons := STATE`GetDefaultCons (nm));
return {nm ↦ sigmacl} )
```

functions

1052.0 $EmptySigmaClass : () \to GLOBAL`SigmaClass$

.1 $EmptySigmaClass\ () \triangleq$

.2 $\textsf{mk-}GLOBAL`SigmaClass\ (\{\},$

.3 $[], \{\mapsto\}, [], \{\mapsto\},$

.4 $[], \{\mapsto\},$

.5 $\{\mapsto\},$

.6 $\{\mapsto\},$

.7 $\{\mapsto\},$

.8 $\{\mapsto\},$

.9 $\{\mapsto\},$

.10 $\{\mapsto\},$

.11 $\{\mapsto\},$

.12 $\{\mapsto\}, \{\mapsto\}, \{\mapsto\},$

.13 $\textsf{false},$

.14 $\{\mapsto\},$

.15 $\textsf{false},$

.16 $\{\mapsto\},$

.17 $\{\mapsto\},$

.18 $\textsf{nil}\ ,$

.19 $\{\mapsto\})$

operations

1053.0 $UpdateConstructors : AS`Name \times (AS`Name^*) \times (AS`Name \xrightarrow{m} AS`OpDef) \times AS`InstAssignDef^* \xrightarrow{o}$

.1 $AS`Type^* \xrightarrow{m} (STKM`DebugCmd \times AS`Access)$

.2 $UpdateConstructors\ (cls, supercls, opm, instvars) \triangleq$

.3 $\textsf{let}\ superlist = \textsf{if}\ supercls = []$

.4 $\textsf{then}\ []$

.5 $\textsf{else}\ STATE`AllSuperList\ (1, supercls)\ \textsf{in}$

.6 $CLASS`CreateConstructor(cls, instvars, superlist, opm)$

The operation *ReadClasses* translates one class into the state. The fields *inst_init_val* (the initial values of instance variables), *vls_init* (the values of values) are all assigned to an empty value. These values cannot be computed before all the classes in the specification has been read. This is done by the operation *InitSigma*. The flag *isinit* is set to false, the flag describes the initialisation of the values and instance variables of the class.

functions

1054.0 $MergeFnsOpsPolys : GLOBAL`SigmaClass \to GLOBAL`All\text{-}Fns\text{-}Ops\text{-}Polys$

.1 $MergeFnsOpsPolys\ (s) \triangleq$

.2 $\textsf{merge}\ \{s.explfns, s.implfns,$

.3 $s.explops, s.implops,$

.4 $s.explpolys, s.implpolys\};$

1055.0 $UpdateTypeDefs : (AS`Name \xrightarrow{m} AS`TypeDef) \times AS`Name \rightarrow$
.1 $AS`Name \xrightarrow{m} AS`TypeDef$
.2 $UpdateTypeDefs\,(typem, curcls) \triangleq$
.3 $\{nm \mapsto \mu\,(typem\,(nm), shape \mapsto UpdateType\,(typem\,(nm).shape, curcls),$
.4 $access \mapsto RealAccess\,(typem\,(nm).access, \mathrm{TP})) \mid$
.5 $nm \in \mathsf{dom}\ typem\};$

```
1056.0 UpdateType : AS`Type × AS`Name → AS`Type
UpdateType (type, curcls) △
  cases type :
    mk-AS`TypeName (nm, ci) →
        if STATE`AClass (nm)
        then type
        else if len nm.ids = 1
            then mk-AS`TypeName
                    (
                     mk-AS`Name (curcls.ids ⁀ [hd nm.ids], ci), ci)
            else type,
    mk-AS`CompositeType (nm, fields, ci) →
        let new-fls = [μ (fields (i),

                          type ↦ UpdateType (fields (i).type, curcls)) |
                         i ∈ inds fields] in
        mk-AS`CompositeType (nm, new-fls, ci),
    mk-AS`UnionType (typeset, ci) →
        let new-ts = [UpdateType (typeset (i), curcls) | i ∈ inds typeset] in
        mk-AS`UnionType (new-ts, ci),
    mk-AS`ProductType (typeseq, ci) →
        let new-ts = [UpdateType (typeseq (i), curcls) | i ∈ inds typeseq] in
        mk-AS`ProductType (new-ts, ci),
    mk-AS`OptionalType (type′, ci) →
        mk-AS`OptionalType (UpdateType (type′, curcls), ci),
    mk-AS`SetType (type′, ci) →
        mk-AS`SetType (UpdateType (type′, curcls), ci),
    mk-AS`BracketedType (type′, ci) →
        mk-AS`BracketedType (UpdateType (type′, curcls), ci),
    mk-AS`Seq0Type (type′, ci) →
        mk-AS`Seq0Type (UpdateType (type′, curcls), ci),
    mk-AS`Seq1Type (type′, ci) →
        mk-AS`Seq1Type (UpdateType (type′, curcls), ci),
    mk-AS`GeneralMapType (dt, rt, ci) →
        mk-AS`GeneralMapType (UpdateType (dt, curcls),
                              UpdateType (rt, curcls), ci),
    mk-AS`InjectiveMapType (dt, rt, ci) →
        mk-AS`InjectiveMapType (UpdateType (dt, curcls),
                                UpdateType (rt, curcls), ci),
    mk-AS`PartialFnType (dt, rt, ci) →
        let new-dt = [UpdateType (dt (i), curcls) | i ∈ inds dt] in
        mk-AS`PartialFnType (new-dt, UpdateType (rt, curcls), ci),
    mk-AS`TotalFnType (dt, rt, ci) →
        let new-dt = [UpdateType (dt (i), curcls) | i ∈ inds dt] in
        mk-AS`TotalFnType (new-dt, UpdateType (rt, curcls), ci),
    others → type
  end;
```

1.18.2 Reading Inheritance

The functions in this section describes the translation of the inheritance structure in the *SigmaClass* data type.

The function *TransInheritance* computes the superclasses of a class.

1057.0 $TransInheritance : AS`Name^* \to GLOBAL`InhCon$

.1 $TransInheritance\,(inh\text{-}l) \triangleq$

.2 $\quad \mathsf{elems}\ inh\text{-}l;$

1.18.3 Reading Instance Variables

The functions in this section describes the translation of instance variabels into the *SigmaClass* data type. The main function is the function *TrasnInstVars*.

1058.0 $TransInstVars : AS`InstanceVarDef^* \to$

.1 $\qquad AS`InstAssignDef^* \times AS`InstanceInv^* \times AS`Name \xrightarrow{m} AS`Type$

.2 $TransInstVars\,(ivar\text{-}l) \triangleq$

.3 $\quad \mathsf{let}\ instvars = [ivar\text{-}l\,(i) \mid i \in \mathsf{inds}\ ivar\text{-}l \cdot$

.4 $\qquad \mathsf{is}\text{-}AS`InstAssignDef\,(ivar\text{-}l\,(i))],$

.5 $\qquad instinv = [ivar\text{-}l\,(i) \mid i \in \mathsf{inds}\ ivar\text{-}l \cdot$

.6 $\qquad \mathsf{is}\text{-}AS`InstanceInv\,(ivar\text{-}l\,(i))],$

.7 $\qquad instvars\text{-}tp = \{asgndef.ad.var \mapsto asgndef.ad.tp \mid$

.8 $\qquad asgndef \in \mathsf{elems}\ ivar\text{-}l \cdot$

.9 $\qquad \mathsf{is}\text{-}AS`InstAssignDef\,(asgndef)\}\ \mathsf{in}$

.10 $\quad \mathsf{mk}\text{-}\,(instvars, instinv, instvars\text{-}tp)$

1.18.4 xtracting Static Member Declarations

Static members inside classes must be available at any time (even if there does not exists any instances of a class). The `ExtractStaticMembers` operation extract the statically declared function and operations from a class. This means that more references to such semantic entities will be present at a given time.
operations

1059.0 $ExtractStaticMembers : AS`Name \xrightarrow{m} AS`FnDef \times AS`Name \xrightarrow{m} AS`OpDef \times GLOBAL`SigmaClass \xrightarrow{o}$

.1 $\qquad AS`Name \xrightarrow{m} (SEM`VAL \times AS`Access)$

.2 $ExtractStaticMembers\,(fnm, opm, sigma) \triangleq$

.3 $\quad (\mathsf{dcl}\ statmap : AS`Name \xrightarrow{m} (SEM`VAL \times AS`Access) := \{\mapsto\};$

.4 $\quad \mathsf{for\ all}\ fnnm \in \mathsf{dom}\ fnm$

.5 do if *fnm* (*fnnm*).*static*
.6 then (*statmap*(*fnnm*) := *LookUpLocalFn* (*fnnm*, *sigma*);
.7 if *fnm* (*fnnm*).*fnpre* $\neq$ nil
.8 then let *prenm* = *AUX*‘*PreName* (*fnnm*) in
.9 *statmap*(*prenm*) := *LookUpLocalFn* (*prenm*, *sigma*);
.10 if *fnm* (*fnnm*).*fnpost* $\neq$ nil
.11 then let *postnm* = *AUX*‘*PostName* (*fnnm*) in
.12 *statmap*(*postnm*) := *LookUpLocalFn* (*postnm*, *sigma*));
.13 for all *opnm* $\in$ dom *opm*
.14 do if *opm* (*opnm*).*static*
.15 then *statmap*(*opnm*) := if *opnm* $\in$ dom *sigma*.*explops*
.16 then *sigma*.*explops* (*opnm*)
.17 else *sigma*.*implops* (*opnm*);
.18 return *statmap*);

1060.0 *LookUpLocalFn* : *AS*‘*Name* $\times$ *GLOBAL*‘*SigmaClass* $\xrightarrow{o}$ *SEM*‘*VAL* $\times$ *AS*‘*Access*
.1 *LookUpLocalFn* (*fnnm*, *sigma*) $\triangleq$
.2 return if *fnnm* $\in$ dom *sigma*.*explfns*
.3 then *sigma*.*explfns* (*fnnm*)
.4 elseif *fnnm* $\in$ dom *sigma*.*implfns*
.5 then *sigma*.*implfns* (*fnnm*)
.6 elseif *fnnm* $\in$ dom *sigma*.*explpolys*
.7 then *sigma*.*explpolys* (*fnnm*)
.8 else *sigma*.*implpolys* (*fnnm*)

1.18.5 Reading Types

The function *TransLocalRecSel* computes all the Local Record Selector Type information.
functions

1061.0 *TransLocalRecSel* : *AS*‘*Name* $\xrightarrow{m}$ *AS*‘*TypeDef* $\rightarrow$
.1 *AS*‘*Name* $\xrightarrow{m}$ (*GLOBAL*‘*RecSel* $\times$ *AS*‘*Access*)
.2 *TransLocalRecSel* (*tpdefs*) $\triangleq$
.3 *TransTP* ({mk- (*tp*.*shape*, *RealAccess* (*tp*.*access*, TP)) |
.4 *tp* $\in$ rng *tpdefs*})

operations

1062.0 *TransFnMap* : *AS*‘*Name* $\times$ *AS*‘*Name* $\xrightarrow{m}$ *AS*‘*FnDef* $\xrightarrow{o}$ *FnTuple*
.1 *TransFnMap* (*mod-id*, *fnm*) $\triangleq$
.2 (dcl *efn* : *AS*‘*Name* $\xrightarrow{m}$ (*SEM*‘*CompExplFN* $\times$ *AS*‘*Access*) := $\{\mapsto\}$,
.3 *ifn* : *AS*‘*Name* $\xrightarrow{m}$ (*SEM*‘*ImplFN* $\times$ *AS*‘*Access*) := $\{\mapsto\}$,
.4 *epf* : *AS*‘*Name* $\xrightarrow{m}$ (*SEM*‘*ExplPOLY* $\times$ *AS*‘*Access*) := $\{\mapsto\}$,
.5 *ipf* : *AS*‘*Name* $\xrightarrow{m}$ (*SEM*‘*ImplPOLY* $\times$ *AS*‘*Access*) := $\{\mapsto\}$;

```
.6      for all id ∈ dom fnm
.7      do let FNval = TransFN (mod-id, fnm (id)) in
.8          if is-SEM`CompExplFN (FNval)
.9          then efn := efn †
.10                     {id ↦ mk- (FNval, RealAccess (fnm (id).access, FN))} †
.11                     CreateExplPrePostFns (mod-id, fnm (id))
.12         elseif is-SEM`ImplFN (FNval)
.13         then (ifn := ifn †
.14                     {id ↦ mk- (FNval, RealAccess (fnm (id).access, FN))};
.15                efn := efn † CreateImplPrePostFns (mod-id, fnm (id)))
.16         elseif is-SEM`ExplPOLY (FNval)
.17         then epf := epf †
.18                     {id ↦ mk- (FNval, RealAccess (fnm (id).access, FN))} †
.19                     CreateExplPolyPrePostFns (mod-id, fnm (id))
.20         else (ipf := ipf †
.21                     {id ↦ mk- (FNval, RealAccess (fnm (id).access, FN))};
.22               epf := epf † CreateImplPolyPrePostFns (mod-id, fnm (id)));
.23     return mk- (efn, ifn, epf, ipf) );
```

This operation translates all function definitions into semantic function values. It also creates the pre and/or post predicate functions for the function definitions.

```
1063.0  TransFN : AS`Name × AS`FnDef -o-> SEM`FN | SEM`POLY
.1      TransFN (mod-id, fn) △
.2        let instr = CMPL`CompileFnOpDef (fn) in
.3        if is-AS`ExplFnDef (fn)
.4        then let mk-AS`ExplFnDef (nm, ttp, tp, parms-l, -, -, -, -, -, -) = fn in
.5             let pi-l = [CPAT`PL2PL (parms-l (j)) | j ∈ inds parms-l] in
.6             if len ttp = 0
.7             then return SEM`CompFN
.8                              (
.9                               mk-SEM`ExplFN ( TransType (tp), pi-l, instr,
.10                                                AUX`MkEmptyBlkEnv (READ_ONLY), {↦
},
.11                                                nm, mod-id, nil , nil ))
.12            else return mk-SEM`ExplPOLY (ttp,
.13                                         INSTR`UpdateTypeInfo ( TransType (tp),
.14                                                                CMPL`GetClMod ()),
.15                                         pi-l, instr,
.16                                         AUX`MkEmptyBlkEnv (READ_ONLY),
.17                                         nm, mod-id, nil , nil )
.18       elseif is-AS`ExtExplFnDef (fn)
.19       then let mk-AS`ExtExplFnDef (nm, ttp, parml, resnmtps, -, -, -, -, -, -) = fn in
.20            let mk- (fndom, parms) = ImplicitTypeParams (parml),
.21                fnrng = ImplicitResType (resnmtps) in
.22            let tp = mk-AS`TotalFnType (fndom, fnrng, CI`NilContextId),
```

.23 $\quad\quad pi\text{-}l = CPAT`PL2PL\,(parms)$ in
.24 $\quad$ if len $ttp = 0$
.25 $\quad$ then return $SEM`CompFN$
.26 $\quad\quad\quad$ (
.27 $\quad\quad\quad$ mk-$SEM`ExplFN\,(TransType\,(tp), [pi\text{-}l], instr,$
.28 $\quad\quad\quad\quad AUX`MkEmptyBlkEnv\,(\text{READ\_ONLY}), \{\mapsto$
$\},$
.29 $\quad\quad\quad\quad nm, mod\text{-}id, resnmtps,$ nil $))$
.30 $\quad$ else return mk-$SEM`ExplPOLY\,(ttp,$
.31 $\quad\quad\quad\quad INSTR`UpdateTypeInfo\,(TransType\,(tp),$
.32 $\quad\quad\quad\quad\quad CMPL`GetClMod\,()),$
.33 $\quad\quad\quad\quad [pi\text{-}l], instr,$
.34 $\quad\quad\quad\quad AUX`MkEmptyBlkEnv\,(\text{READ\_ONLY}), nm,$
.35 $\quad\quad\quad\quad mod\text{-}id, resnmtps,$ nil)
.36 else let mk-$AS`ImplFnDef\,(\text{-}, ttp, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}) = fn$ in
.37 $\quad$ if len $ttp = 0$
.38 $\quad$ then return mk-$SEM`ImplFN\,()$
.39 $\quad$ else return mk-$SEM`ImplPOLY\,()$;

This operation translates a single function definition into a semantic function value. Note that the initial closure environment is empty, and, in case of explicit non-polymorphic functions, the type instantiation map is also empty.

1.18.6 Creating Pre and Post Functions

1064.0 $CreateExplPrePostFns : AS`Name \times (AS`ExplFnDef \mid AS`ExtExplFnDef) \xrightarrow{o}$
.1 $\quad\quad\quad AS`Name \xrightarrow{m} (SEM`CompExplFN \times AS`Access)$
.2 $CreateExplPrePostFns\,(mod\text{-}id, fndef) \triangleq$
.3 $\quad$ if is-$AS`ExplFnDef\,(fndef)$
.4 $\quad$ then $DoCreateExplPrePostFns(mod\text{-}id, fndef)$
.5 $\quad$ else $DoCreateExtExplPrePostFns(mod\text{-}id, fndef)$;

1065.0 $DoCreateExplPrePostFns : AS`Name \times AS`ExplFnDef \xrightarrow{o}$
.1 $\quad\quad\quad AS`Name \xrightarrow{m} (SEM`CompExplFN \times AS`Access)$
.2 $DoCreateExplPrePostFns\,(mod\text{-}id,$ mk-$AS`ExplFnDef\,(name, \text{-}, tp, parms, \text{-}, Pre\text{-}e,$
.3 $\quad\quad\quad Post\text{-}e, access, static, \text{-})) \triangleq$
.4 $\quad$ (dcl $res\text{-}m : AS`Name \xrightarrow{m} (SEM`CompExplFN \times AS`Access) := \{\mapsto\};$
.5 $\quad$ if $Pre\text{-}e =$ nil
.6 $\quad$ then skip
.7 $\quad$ else let $nid = CI`NilContextId$ in
.8 $\quad\quad$ let $nm\text{-}pre = AUX`PreName\,(name),$
.9 $\quad\quad\quad bt =$ mk-$AS`BasicType\,(\text{BOOLEAN}, nid),$

```
.10             pi-l = CPAT`PL2PL(hd parms) in
.11         let fn-pre = SEM`CompFN
.12                       (
.13                        mk-SEM`ExplFN (mk-AS`TotalFnType(tp.fndom, bt, nid),
.14                                       [pi-l],
.15                                       CMPL`CompilePrePostExpr(Pre-e),
.16                                       AUX`MkEmptyBlkEnv(READ_ONLY),
.17                                       {|->}, nm-pre, mod-id, nil , nil )) in
.18         res-m := res-m † {nm-pre |-> mk-(fn-pre, RealAccess(access, FN))};
.19     if Post-e = nil
.20     then skip
.21     else let mk-AS`Name(-, cid) = name,
.22             nm-post = AUX`PostName(name),
.23             tp-post = CreateFunctionPostType(tp.fndom, [tp.fnrng]),
.24             parm-post = CPAT`PL2PL(hd parms) ^
.25                       [mk-STKM`PatternName(mk-AS`Name(["RESULT"], cid), cid)] in
.26         let fn-post = SEM`CompFN
.27                       (
.28                        mk-SEM`ExplFN (tp-post,
.29                                       [parm-post],
.30                                       CMPL`CompilePrePostExpr(Post-e),
.31                                       AUX`MkEmptyBlkEnv(READ_ONLY),
.32                                       {|->},
.33                                       nm-post,
.34                                       mod-id, nil , nil )) in
.35         res-m := res-m † {nm-post |-> mk-(fn-post, RealAccess(access, FN))};
.36     return res-m );


1066.0 DoCreateExtExplPrePostFns : AS`Name × AS`ExtExplFnDef -o->
.1                                  AS`Name -m-> (SEM`CompExplFN × AS`Access)
.2     DoCreateExtExplPrePostFns(mod-id, mk-AS`ExtExplFnDef(name, -, parml, resnmtps, -,
.3                                          Pre-e, Post-e, access, static, -)) △
.4     let mk-(fndom, parms) = ImplicitTypeParams(parml),
.5         mk-(resnms, restps) = ImplicitResNameTypes(resnmtps) in
.6     (dcl res-m : AS`Name -m-> (SEM`CompExplFN × AS`Access) := {|->};
.7      if Pre-e = nil
.8      then skip
.9      else let nid = CI`NilContextId in
.10         let nm-pre = AUX`PreName(name),
```

```
            pi-l = CPAT`PL2PL (parms) in
        let fn-pre = SEM`CompFN
                        (
                         mk-SEM`ExplFN (mk-AS`TotalFnType (fndom,
                                                           mk-AS`BasicType (BOOLEAN, nid), nid),
                                        [pi-l],
                                        CMPL`CompilePrePostExpr (Pre-e),
                                        AUX`MkEmptyBlkEnv (READ_ONLY),
                                        {|->}, nm-pre, mod-id, nil , nil )) in
        res-m := res-m † {nm-pre |-> mk- (fn-pre, RealAccess (access, FN))};
    if Post-e = nil
    then skip
    else let p-res = CreatePostParms (resnms) in
        let nm-post = AUX`PostName (name),
            tp-post = CreateFunctionPostType (fndom, restps),
            parm-post = CPAT`PL2PL (parms) ^ CPAT`PL2PL (p-res) in
        let fn-post = SEM`CompFN
                        (
                         mk-SEM`ExplFN (tp-post,
                                        [parm-post],
                                        CMPL`CompilePrePostExpr (Post-e),
                                        AUX`MkEmptyBlkEnv (READ_ONLY),
                                        {|->},
                                        nm-post,
                                        mod-id, nil , nil )) in
        res-m := res-m † {nm-post |-> mk- (fn-post, RealAccess (access, FN))};
    return res-m )
```

functions

```
1067.0  CreatePostParms : AS`Name* -> AS`Pattern*

        CreatePostParms (resnms) ≜
          if len resnms = 0
          then []
          elseif len resnms = 1
          then let nm = hd resnms in
                 [mk-AS`PatternName (nm, nm.cid)]
          else [mk-AS`TuplePattern ([mk-AS`PatternName (resnms (i), resnms (i).cid) |
                                         i ∈ inds resnms],
                                    (hd resnms).cid)]
```

Toghether these two operations return the semantic function values for the pre and/or post predicate functions for a non-polymorphic explicit or extended explicit function definition. Both the closure environment and the type instantiation map are empty.

operations

```
CreateImplPrePostFns : AS`Name × AS`ImplFnDef -o->
                               AS`Name -m-> (SEM`CompExplFN × AS`Access)
CreateImplPrePostFns (mod-id, mk-AS`ImplFnDef (name, -, partps, resnmtps, Pre-e,
                                               Post-e, access, static, -)) △
  let mk- (fn-tp, fn-parm) = ImplicitTypeParams (partps),
      mk- (res-nms, fnrng) = ImplicitResNameTypes (resnmtps) in
  (dcl res-m : AS`Name -m-> (SEM`CompExplFN × AS`Access) := {↦};
   if Pre-e = nil
   then skip
   else let nid = CI`NilContextId in
        let nm-pre = AUX`PreName (name),
            pi-l = CPAT`PL2PL (fn-parm) in
        let fn-pre = SEM`CompFN
                          (
                           mk-SEM`ExplFN
                                 (
                                  mk-AS`TotalFnType (fn-tp, mk-AS`BasicType (BOOLEAN, nid), nid),
                                  [pi-l],
                                  CMPL`CompilePrePostExpr (Pre-e),
                                  AUX`MkEmptyBlkEnv (READ_ONLY),
                                  {↦},
                                  nm-pre,
                                  mod-id, nil , nil )) in
        res-m := res-m † {nm-pre ↦ mk- (fn-pre, RealAccess (access, FN))};
   let res-p = CreatePostParms (res-nms) in
   let nm-post = AUX`PostName (name),
       tp-post = CreateFunctionPostType (fn-tp, fnrng),
       pi-l = CPAT`PL2PL (fn-parm) ⁀ CPAT`PL2PL (res-p) in
   let fn-post = SEM`CompFN
                      (
                       mk-SEM`ExplFN (tp-post,
                                      [pi-l],
                                      CMPL`CompilePrePostExpr (Post-e),
                                      AUX`MkEmptyBlkEnv (READ_ONLY),
                                      {↦},
                                      nm-post,
                                      mod-id, nil , nil )) in
   res-m := res-m † {nm-post ↦ mk- (fn-post, RealAccess (access, FN))};
   return res-m );
```

This operation creates the semantic function values for the pre and/or post predicate functions for a non-polymorphic implicit function definition.

```
1069.0 CreateExplPolyPrePostFns : AS`Name × (AS`ExplFnDef | AS`ExtExplFnDef) -o->
    .1                          AS`Name -m-> (SEM`ExplPOLY × AS`Access)
    .2 CreateExplPolyPrePostFns (mod-id, fndef) ≜
    .3   if is-AS`ExplFnDef (fndef)
    .4   then DoCreateExplPolyPrePostFns(mod-id, fndef)
    .5   else DoCreateExtExplPolyPrePostFns(mod-id, fndef) ;

1070.0 DoCreateExplPolyPrePostFns : AS`Name × AS`ExplFnDef -o->
    .1                          AS`Name -m-> (SEM`ExplPOLY × AS`Access)
    .2 DoCreateExplPolyPrePostFns (mod-id, mk-AS`ExplFnDef (name, ttp, tp, parms, -, Pre-e,
    .3                                                      Post-e, access, static, -)) ≜
    .4   (dcl res-m : AS`Name -m-> (SEM`ExplPOLY × AS`Access) := {↦};
    .5    if Pre-e = nil
    .6    then skip
    .7    else let nid = CI`NilContextId in
    .8         let nm-pre = AUX`PreName (name),
    .9             pi-l = [CPAT`PL2PL (parms (j)) | j ∈ inds parms] in
    .10        let fn-pre = mk-SEM`ExplPOLY (ttp,
    .11                                     mk-AS`TotalFnType (tp.fndom,
    .12                                            mk-AS`BasicType (BOOLEAN, nid), nid),
    .13                                     pi-l,
    .14                                     CMPL`CompilePrePostExpr (Pre-e),
    .15                                     AUX`MkEmptyBlkEnv (READ_ONLY),
    .16                                     nm-pre,
    .17                                     mod-id, nil , nil ) in
    .18        res-m := res-m † {nm-pre ↦ mk- (fn-pre, RealAccess (access, FN))};
    .19   if Post-e = nil
    .20   then skip
    .21   else let nid = CI`NilContextId in
    .22        let nm-post = AUX`PostName (name),
    .23            tp-post = CreateFunctionPostType (tp.fndom, [tp.fnrng]),
    .24            parm-post = CPAT`PL2PL (hd parms) ⁀
    .25                    [mk-STKM`PatternName (mk-AS`Name (["RESULT"], nid), nid)] in
    .26        let fn-post = mk-SEM`ExplPOLY (ttp,
    .27                                      tp-post,
    .28                                      [parm-post],
    .29                                      CMPL`CompilePrePostExpr (Post-e),
    .30                                      AUX`MkEmptyBlkEnv (READ_ONLY),
    .31                                      nm-post,
    .32                                      mod-id, nil , nil ) in
    .33        res-m := res-m † {nm-post ↦ mk- (fn-post, RealAccess (access, FN))};
    .34   return res-m );
```

```
1071.0 DoCreateExtExplPolyPrePostFns : AS`Name × AS`ExtExplFnDef o→
                                        AS`Name -m-> (SEM`ExplPOLY × AS`Access)
DoCreateExtExplPolyPrePostFns (mod-id, mk-AS`ExtExplFnDef (name, ttp, parml, resnmtps,
                                                           -, Pre-e, Post-e, access,
                                                           static, -)) △
  let mk- (fndom, parms) = ImplicitTypeParams (parml),
      mk- (resnms, restps) = ImplicitResNameTypes (resnmtps) in
  (dcl res-m : AS`Name -m-> (SEM`ExplPOLY × AS`Access) := {↦};
   if Pre-e = nil
   then skip
   else let nid = CI`NilContextId in
        let nm-pre = AUX`PreName (name),
            pi-l = CPAT`PL2PL (parms) in
        let fn-pre = mk-SEM`ExplPOLY (ttp,
                                      mk-AS`TotalFnType (fndom,
                                                         mk-AS`BasicType (BOOLEAN, nid), nid),
                                      [pi-l],
                                      CMPL`CompilePrePostExpr (Pre-e),
                                      AUX`MkEmptyBlkEnv (READ_ONLY),
                                      nm-pre,
                                      mod-id, nil , nil ) in
        res-m := res-m † {nm-pre ↦ mk- (fn-pre, RealAccess (access, FN))};
   if Post-e = nil
   then skip
   else let res-p = CreatePostParms (resnms) in
        let nm-post = AUX`PostName (name),
            tp-post = CreateFunctionPostType (fndom, restps),
            parm-post = CPAT`PL2PL (parms) ⁀ CPAT`PL2PL (res-p) in
        let fn-post = mk-SEM`ExplPOLY (ttp,
                                       tp-post,
                                       [parm-post],
                                       CMPL`CompilePrePostExpr (Post-e),
                                       AUX`MkEmptyBlkEnv (READ_ONLY),
                                       nm-post,
                                       mod-id, nil , nil ) in
        res-m := res-m † {nm-post ↦ mk- (fn-post, RealAccess (access, FN))};
   return res-m );
```

This operation creates the semantic function values for the pre and/or post predicate functions for an explicit polymorphic function definitions.

```
1072.0 CreateImplPolyPrePostFns : AS`Name × AS`ImplFnDef o→
                                   AS`Name -m-> (SEM`ExplPOLY × AS`Access)
CreateImplPolyPrePostFns (mod-id, mk-AS`ImplFnDef (name, tpp, partps, resnmtps,
                                                   Pre-e, Post-e, access, static, -)) △
  let mk- (fn-tp, fn-parm) = ImplicitTypeParams (partps),
```

```
      mk- (res-nms, fnrng) = ImplicitResNameTypes (resnmtps) in
  (dcl res-m : AS`Name -m-> (SEM`ExplPOLY × AS`Access) := {↦};
   if Pre-e = nil
   then skip
   else let nid = CI`NilContextId in
        let nm-pre = AUX`PreName (name),
            pi-l = CPAT`PL2PL (fn-parm) in
        let fn-pre = mk-SEM`ExplPOLY
                          (
                           tpp,
                           mk-AS`TotalFnType (fn-tp, mk-AS`BasicType (BOOLEAN, nid), nid),
                           [pi-l],
                           CMPL`CompilePrePostExpr (Pre-e),
                           AUX`MkEmptyBlkEnv (READ_ONLY),
                           nm-pre,
                           mod-id, nil , nil ) in
         res-m := res-m † {nm-pre ↦ mk- (fn-pre, RealAccess (access, FN))};
   let res-p = CreatePostParms (res-nms) in
   let nm-post = AUX`PostName (name),
       tp-post = CreateFunctionPostType (fn-tp, fnrng),
       parm-post = CPAT`PL2PL (fn-parm) ⁀ CPAT`PL2PL (res-p) in
   let fn-post = mk-SEM`ExplPOLY (tpp,
                                  tp-post,
                                  [parm-post],
                                  CMPL`CompilePrePostExpr (Post-e),
                                  AUX`MkEmptyBlkEnv (READ_ONLY),
                                  nm-post,
                                  mod-id, nil , nil ) in
   res-m := res-m † {nm-post ↦ mk- (fn-post, RealAccess (access, FN))};
   return res-m );
```

This operation returns the semantic function values for the pre and/or post predicate functions for an implicit polymorphic function definition.

```
1073.0 ImplicitTypeParams : AS`ParameterTypes -o-> AS`DiscretionaryType × AS`Parameters
  ImplicitTypeParams (partps) △
    if len partps = 0
    then return mk- ([], [])
    else (dcl tp-l : AS`Type* := [],
              parm-l : AS`Pattern* := [];
          for mk-AS`PatTypePair (pat-l, tp, -) in partps
          do for pat in pat-l
             do (tp-l := tp-l ⁀ [tp];
                 parm-l := parm-l ⁀ [pat]);
          return mk- (tp-l, parm-l) );
```

This operation converts the pattern/type pairs of an implicit function or operation definition into a type and a sequence. The sequence contains all the patterns defined, and the type is the domain type of the function or operation.

1074.0 $\mathit{ImplicitResNameTypes} : \mathit{AS`NameType}^{*} \xrightarrow{o} \mathit{AS`Name}^{*} \times \mathit{AS`Type}^{*}$
$\mathit{ImplicitResNameTypes}\,(\mathit{resnmtps}) \triangleq$
  $\mathsf{if}\ \mathit{resnmtps} = [\,]$
  $\mathsf{then\ return\ mk\text{-}}\,([\,], [\,])$
  $\mathsf{else}\ (\mathsf{dcl}\ \mathit{tp\text{-}l} : \mathit{AS`Type}^{*} := [\,],$
      $\mathit{nm\text{-}l} : \mathit{AS`Name}^{*} := [\,];$
    $\mathsf{for\ mk\text{-}}\mathit{AS`NameType}\,(\mathit{nm}, \mathit{tp}, \text{-})\ \mathsf{in}\ \mathit{resnmtps}$
    $\mathsf{do}\ (\mathit{tp\text{-}l} := \mathit{tp\text{-}l} \curvearrowright [\mathit{tp}];$
      $\mathit{nm\text{-}l} := \mathit{nm\text{-}l} \curvearrowright [\mathit{nm}]);$
    $\mathsf{return\ mk\text{-}}\,(\mathit{nm\text{-}l}, \mathit{tp\text{-}l})\ );$

This operation converts the result name/types of an implicit or extended explicit function/operation into a sequence of name and a type. If there is only one name/type this type is returned, if there is more than one type these are combined in an AS‘ProductType.

1075.0 $\mathit{ImplicitResType} : \mathit{AS`NameType}^{*} \xrightarrow{o} [\mathit{AS`Type}]$
$\mathit{ImplicitResType}\,(\mathit{resnmtps}) \triangleq$
  $\mathsf{if}\ \mathit{resnmtps} = [\,]$
  $\mathsf{then\ return\ nil}$
  $\mathsf{else}\ (\mathsf{dcl}\ \mathit{tp\text{-}l} : \mathit{AS`Type}^{*} := [\,];$
    $\mathsf{for\ mk\text{-}}\mathit{AS`NameType}\,(\text{-}, \mathit{tp}, \text{-})\ \mathsf{in}\ \mathit{resnmtps}$
    $\mathsf{do}\ \mathit{tp\text{-}l} := \mathit{tp\text{-}l} \curvearrowright [\mathit{tp}];$
    $\mathsf{if\ len}\ \mathit{tp\text{-}l} = 1$
    $\mathsf{then\ let}\ \mathit{tp} = \mathsf{hd}\ \mathit{tp\text{-}l}\ \mathsf{in}$
      $\mathsf{return}\ \mathit{tp}$
    $\mathsf{else\ return\ mk\text{-}}\mathit{AS`ProductType}\,(\mathit{tp\text{-}l}, \mathit{CI`NilContextId})\ );$

This operation converts the result types of an implicit or extended explicit function/operation into a type: if the sequence has one element into this type, if it has more than one element into a product type.

1076.0 $\mathit{CreateFunctionPostType} : \mathit{AS`DiscretionaryType} \times \mathit{AS`DiscretionaryType} \xrightarrow{o} \mathit{AS`Type}$
$\mathit{CreateFunctionPostType}\,(\mathit{tpdom}, \mathit{tprng}) \triangleq$
  $\mathsf{let}\ \mathit{nid} = \mathit{CI`NilContextId},$
    $\mathit{realrng} = \mathsf{if\ len}\ \mathit{tprng} \leq 1$
      $\mathsf{then}\ \mathit{tprng}$
      $\mathsf{else}\ [\mathsf{mk\text{-}}\mathit{AS`ProductType}\,(\mathit{tprng}, (\mathsf{hd}\ \mathit{tprng}).\mathit{cid})]\ \mathsf{in}$
  $\mathsf{return\ mk\text{-}}\mathit{AS`TotalFnType}\,(\mathit{tpdom} \curvearrowright \mathit{realrng}, \mathsf{mk\text{-}}\mathit{AS`BasicType}\,(\mathrm{BOOLEAN}, \mathit{nid}), \mathit{nid});$

This operation returns the type of a post predicate function in a function definition.

1077.0
```
CreateOperationPreType : AS`DiscretionaryType × [AS`Name] →(o) AS`Type
CreateOperationPreType (tpdom, st-id) △
  let nid = CI`NilContextId in
  if tpdom = []
  then if st-id = nil
       then return mk-AS`TotalFnType ([], mk-AS`BasicType (BOOLEAN, nid), nid)
       else return mk-AS`TotalFnType ([mk-AS`TypeName (st-id, st-id.cid)],
                                      mk-AS`BasicType (BOOLEAN, nid), nid)
  else if st-id = nil
      then return mk-AS`TotalFnType (tpdom, mk-AS`BasicType (BOOLEAN, nid), nid)
      else return mk-AS`TotalFnType (tpdom ⁀ [mk-AS`TypeName (st-id, st-id.cid)],
                                     mk-AS`BasicType (BOOLEAN, nid), nid) ;
```

This operation returns the type of a pre predicate function in an operation definition.

1078.0
```
CreateOperationPostType : AS`DiscretionaryType×AS`DiscretionaryType×[AS`Name] →(o)
                          AS`Type
CreateOperationPostType (tpdom, tprng, st-id) △
  let nid = CI`NilContextId,
      realtprng = if tprng = []
                  then []
                  elseif len tprng = 1
                  then tprng
                  else [mk-AS`ProductType (tprng, (hd tprng).cid)] in
  if tpdom = []
  then if st-id = nil
       then return mk-AS`TotalFnType (realtprng, mk-AS`BasicType
                                                  (
                                                   BOOLEAN, nid),
                                      nid)
       else let st-tp = mk-AS`TypeName (st-id, st-id.cid) in
            return mk-AS`TotalFnType (realtprng ⁀ [st-tp, st-tp],
                                      mk-AS`BasicType (BOOLEAN, nid), nid)
  else if st-id = nil
      then let new-tp = if realtprng = []
                        then tpdom
                        else tpdom ⁀ realtprng in
           return mk-AS`TotalFnType (new-tp, mk-AS`BasicType (BOOLEAN, nid), nid)
      else let st-tp = mk-AS`TypeName (st-id, st-id.cid) in
           let new-tp = if realtprng = []
                        then tpdom ⁀ [st-tp, st-tp]
                        else tpdom ⁀ realtprng ⁀ [st-tp, st-tp] in
           return mk-AS`TotalFnType (new-tp, mk-AS`BasicType (BOOLEAN, nid), nid)
;
```

This operation returns the type of the post predicate function for an operation definition.

1079.0 $\mathit{TransOverloaded} : (\mathit{AS}`\mathit{Name} \xrightarrow{m} \mathit{AS}`\mathit{OpDef}) \times (\mathit{AS}`\mathit{Name} \xrightarrow{m} \mathit{AS}`\mathit{FnDef}) \xrightarrow{o}$
.1 $\qquad \mathit{AS}`\mathit{Name} \xrightarrow{m} \mathit{GLOBAL}`\mathit{Overloaded}$
.2 $\mathit{TransOverloaded}\,(\mathit{opm}, \mathit{fnm}) \triangleq$
.3 $\quad$ (dcl $\mathit{over} : \mathit{AS}`\mathit{Name} \xrightarrow{m} \mathit{GLOBAL}`\mathit{Overloaded} := \{\mapsto\}$;
.4 $\quad$ for all $\mathit{id} \in$ dom $\mathit{opm} \cup$ dom fnm
.5 $\quad$ do if $\mathit{MANGLE}`\mathit{IsMangled}\,(\mathit{id})$
.6 $\qquad$ then let mk- $(\mathit{realid}, \mathit{arit}, \mathit{tp\text{-}l}) = \mathit{MANGLE}`\mathit{UnMangle}\,(\mathit{id})$,
.7 $\qquad\qquad \mathit{overload} =$ if $\mathit{realid} \in$ dom over
.8 $\qquad\qquad\qquad$ then $\mathit{over}\,(\mathit{realid}) \dagger$
.9 $\qquad\qquad\qquad\quad$ if $\mathit{arit} \in$ dom $\mathit{over}\,(\mathit{realid})$
.10 $\qquad\qquad\qquad\quad$ then $\{\mathit{arit} \mapsto \mathit{over}\,(\mathit{realid})\,(\mathit{arit}) \dagger$
.11 $\qquad\qquad\qquad\qquad \{\mathit{id} \mapsto \mathit{tp\text{-}l}\}\}$
.12 $\qquad\qquad\qquad\quad$ else $\{\mathit{arit} \mapsto \{\mathit{id} \mapsto \mathit{tp\text{-}l}\}\}$
.13 $\qquad\qquad\qquad$ else $\{\mathit{arit} \mapsto \{\mathit{id} \mapsto \mathit{tp\text{-}l}\}\}$ in
.14 $\qquad\quad \mathit{over} := \mathit{over} \dagger \{\mathit{realid} \mapsto \mathit{overload}\}$;
.15 $\quad$ return over);

1080.0 $\mathit{TransOpMap} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{Name} \xrightarrow{m} \mathit{AS}`\mathit{OpDef} \xrightarrow{o} \mathit{OpTuple}$
.1 $\mathit{TransOpMap}\,(\mathit{mod\text{-}id}, \mathit{opm}) \triangleq$
.2 $\quad$ (dcl $\mathit{eop} : \mathit{AS}`\mathit{Name} \xrightarrow{m} (\mathit{SEM}`\mathit{ExplOP} \times \mathit{AS}`\mathit{Access}) := \{\mapsto\}$,
.3 $\qquad \mathit{iop} : \mathit{AS}`\mathit{Name} \xrightarrow{m} (\mathit{SEM}`\mathit{ImplOP} \times \mathit{AS}`\mathit{Access}) := \{\mapsto\}$;
.4 $\quad$ for all $\mathit{id} \in$ dom opm
.5 $\quad$ do let $\mathit{OPval} = \mathit{TransOP}\,(\mathit{mod\text{-}id}, \mathit{opm}\,(\mathit{id}))$ in
.6 $\qquad$ if is-$\mathit{SEM}`\mathit{ExplOP}\,(\mathit{OPval})$
.7 $\qquad$ then $\mathit{eop} := \mathit{eop} \dagger \{\mathit{id} \mapsto$ mk- $(\mathit{OPval}, \mathit{RealAccess}\,(\mathit{opm}\,(\mathit{id}).\mathit{access}, \mathrm{OP}))\}$
.8 $\qquad$ else $\mathit{iop} := \mathit{iop} \dagger \{\mathit{id} \mapsto$ mk- $(\mathit{OPval}, \mathit{RealAccess}\,(\mathit{opm}\,(\mathit{id}).\mathit{access}, \mathrm{OP}))\}$;
.9 $\quad$ return mk- $(\mathit{eop}, \mathit{iop})$);

The operation *TransOpMap* creates for all the operation definitions the corresponding semantic operation values.

1081.0 $\mathit{TransOP} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{OpDef} \xrightarrow{o} \mathit{SEM}`\mathit{OP}$
.1 $\mathit{TransOP}\,(\mathit{mod\text{-}id}, \mathit{op}) \triangleq$
.2 $\quad$ if is-$\mathit{AS}`\mathit{ExplOpDef}\,(\mathit{op})$
.3 $\quad$ then let mk-$\mathit{AS}`\mathit{ExplOpDef}\,(\mathit{nm}, \mathit{tp}, \mathit{parms}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}) = \mathit{op}$ in
.4 $\qquad$ let $\mathit{pi\text{-}l} = \mathit{CPAT}`\mathit{PL2PL}\,(\mathit{parms})$ in
.5 $\qquad$ return mk-$\mathit{SEM}`\mathit{ExplOP}\,(\mathit{TransType}\,(\mathit{tp}), \mathit{pi\text{-}l}, \mathit{CMPL}`\mathit{CompileFnOpDef}\,(\mathit{op})$,
.6 $\qquad\qquad \mathit{nm}, \mathit{mod\text{-}id}$, nil , nil)
.7 $\quad$ elseif is-$\mathit{AS}`\mathit{ExtExplOpDef}\,(\mathit{op})$
.8 $\quad$ then let mk-$\mathit{AS}`\mathit{ExtExplOpDef}\,(\mathit{nm}, \mathit{parml}, \mathit{resnmtps}, \text{-}, \text{-}$,
.9 $\qquad\qquad \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}, \text{-}) =$
.10 $\qquad\qquad \mathit{op}$ in
.11 $\qquad$ let mk- $(\mathit{opdom}, \mathit{parms}) = \mathit{ImplicitTypeParams}\,(\mathit{parml})$,

```
.12            oprng = ImplicitResType (resnmtps) in
.13        let tp = mk-AS`OpType (opdom, oprng, CI`NilContextId),
.14            pi-l = CPAT`PL2PL (parms) in
.15        return mk-SEM`ExplOP (TransType (tp), pi-l, CMPL`CompileFnOpDef (op), nm,
.16                              mod-id, resnmtps, nil )
.17    else return mk-SEM`ImplOP () ;
```

This operation translates an operation definition into a semantic operation value.

```
1082.0 CreateOperationPrePostFns : AS`Name × (AS`Name -m-> AS`OpDef) × [AS`StateDef] -o->
    .1                             AS`Name -m-> (SEM`CompExplFN × AS`Access)
    .2 CreateOperationPrePostFns (mod-id, opm, gst) ≜
    .3   (dcl res-m : AS`Name -m-> (SEM`CompExplFN × AS`Access) := {↦};
    .4    for all id ∈ dom opm
    .5    do let op-def = opm (id) in
    .6       if is-AS`ExplOpDef (op-def) ∨ is-AS`ExtExplOpDef (op-def)
    .7       then res-m := res-m † CreateExplOpFns (mod-id, op-def, gst)
    .8       else res-m := res-m † CreateImplOpFns (mod-id, op-def, gst);
    .9    return res-m );
```

This operation creates for all operation definitions the semantic function values for the pre and/or post predicate functions.

```
1083.0 CreateOperationPreParms : AS`Parameters × [AS`StateDef] -o-> AS`ParametersList
    .1 CreateOperationPreParms (parm, gst) ≜
    .2   if gst = nil
    .3   then return [parm]
    .4   else let sigma-parm = CreateSigmaPattern (gst.tp, false) in
    .5          return [parm ⁀ [sigma-parm]] ;
```

This operation returns the parameters for the pre predicate function of an operation definition.

```
1084.0 CreateExplOperationPostParms : AS`Parameters × AS`DiscretionaryType × [AS`StateDef] -o->
    .1                                AS`ParametersList
    .2 CreateExplOperationPostParms (parm, resnmtps, gst) ≜
    .3   let nid = CI`NilContextId in
    .4   let res-parm = if resnmtps = []
    .5                  then []
    .6                  else [mk-AS`PatternName (mk-AS`Name (["RESULT"], nid), nid)] in
    .7   if gst = nil
    .8   then return [parm ⁀ res-parm]
    .9   else let sigma-old = CreateSigmaPattern (gst.tp, true),
```

.10 $\qquad\qquad sigma\text{-}res = CreateSigmaPattern\,(gst.tp, \mathsf{false})$ in
.11 $\qquad$ return $[parm \curvearrowright res\text{-}parm \curvearrowright [sigma\text{-}old, sigma\text{-}res]]$;

This operation returns the parameters for the post predicate function of an explicit operation definition.

1085.0 $CreateExtExplOperationPostParms : AS`Parameters \times AS`Name^* \times [AS`StateDef] \xrightarrow{o}$
.1 $\qquad\qquad AS`ParametersList$
.2 $CreateExtExplOperationPostParms\,(parm, resnms, gst) \triangleq$
.3 let $res\text{-}parm = CreatePostParms\,(resnms)$ in
.4 if $gst = \mathsf{nil}$
.5 then return $[parm \curvearrowright res\text{-}parm]$
.6 else let $sigma\text{-}old = CreateSigmaPattern\,(gst.tp, \mathsf{true})$,
.7 $\qquad\qquad sigma\text{-}res = CreateSigmaPattern\,(gst.tp, \mathsf{false})$ in
.8 $\qquad$ return $[parm \curvearrowright res\text{-}parm \curvearrowright [sigma\text{-}old, sigma\text{-}res]]$;

This operation returns the parameters for the post predicate function of an explicit operation definition.

1086.0 $CreateImplOperationPostParms : AS`Parameters \times AS`Name^* \times [AS`StateDef] \xrightarrow{o}$
.1 $\qquad\qquad AS`ParametersList$
.2 $CreateImplOperationPostParms\,(parm, resnms, gst) \triangleq$
.3 let $res\text{-}parm = CreatePostParms\,(resnms)$ in
.4 if $gst = \mathsf{nil}$
.5 then return $[parm \curvearrowright res\text{-}parm]$
.6 else let $sigma\text{-}old = CreateSigmaPattern\,(gst.tp, \mathsf{true})$,
.7 $\qquad\qquad sigma\text{-}res = CreateSigmaPattern\,(gst.tp, \mathsf{false})$ in
.8 $\qquad$ return $[parm \curvearrowright res\text{-}parm \curvearrowright [sigma\text{-}old, sigma\text{-}res]]$;

This operation returns the parameters for the post predicate function of an explicit operation definition.

1087.0 $CreateSigmaPattern : AS`CompositeType \times \mathbb{B} \xrightarrow{o} AS`RecordPattern$
.1 $CreateSigmaPattern\,(\mathsf{mk\text{-}}AS`CompositeType\,(tag, fields, cid), old\text{-}names) \triangleq$
.2 (dcl $rec\text{-}l : AS`Pattern^* := []$;
.3 for mk-$AS`Field\,(sel, \text{-}, \text{-}, cid2)$ in $fields$
.4 do if $sel = \mathsf{nil}$
.5 $\qquad$ then $rec\text{-}l := rec\text{-}l \curvearrowright [\mathsf{mk\text{-}}AS`PatternName\,(\mathsf{nil}, cid2)]$
.6 $\qquad$ else let mk-$AS`Name\,(sel\text{-}id, cid3) = sel$ in
.7 $\qquad\qquad$ let $sel\text{-}id' =$ if $old\text{-}names$
.8 $\qquad\qquad\qquad\qquad$ then $sel\text{-}id\,(1) \curvearrowright$ "~"
.9 $\qquad\qquad\qquad\qquad$ else $sel\text{-}id\,(1)$ in
.10 $\qquad\qquad rec\text{-}l := rec\text{-}l \curvearrowright [\mathsf{mk\text{-}}AS`PatternName\,(\mathsf{mk\text{-}}AS`Name\,([sel\text{-}id'], cid3), cid3)]$;
.11 return mk-$AS`RecordPattern\,(tag, rec\text{-}l, cid)$);

This operation creates a record pattern corresponding to the state type. If the record pattern denotes the old state pattern, all pattern identifiers are appended with the hook symbol. This way, we create unique pattern identifiers for each of the old state components.

```
1088.0 CreateExplOpFns : AS`Name × (AS`ExplOpDef | AS`ExtExplOpDef) × [AS`StateDef] -o->
    .1                   AS`Name -m-> (SEM`CompExplFN × AS`Access)
    .2 CreateExplOpFns (mod-id, opdef, gst) ≜
    .3   if is-AS`ExplOpDef (opdef)
    .4   then DoCreateExplOpFns(mod-id, opdef, gst)
    .5   else DoCreateExtExplOpFns(mod-id, opdef, gst) ;
```

```
1089.0 DoCreateExplOpFns : AS`Name × AS`ExplOpDef × [AS`StateDef] -o->
    .1                     AS`Name -m-> (SEM`CompExplFN × AS`Access)
    .2 DoCreateExplOpFns (mod-id, mk-AS`ExplOpDef (nm, tp, parms, -, Pre-e,
    .3                                   Post-e, access, static, constr, -), gst) ≜
    .4   let st-id = if gst = nil
    .5               then nil
    .6               else gst.tp.name in
    .7   (dcl res-m : AS`Name -m-> (SEM`CompExplFN × AS`Access) := {↦};
    .8    if Pre-e = nil
    .9    then skip
    .10   else let nm-pre = AUX`PreName (nm),
    .11            pl = CreateOperationPreParms (parms, gst) in
    .12        let fn-pre = SEM`CompFN
    .13                     (
    .14                      mk-SEM`ExplFN (TransType (CreateOperationPreType (tp.opdom, st-id)),
    .15                                     [CPAT`PL2PL (pl (i)) | i ∈ inds pl],
    .16                                     CMPL`CompilePrePostExpr (Pre-e),
    .17                                     AUX`MkEmptyBlkEnv (READ_ONLY),
    .18                                     {↦},
    .19                                     nm-pre,
    .20                                     mod-id, nil , nil )) in
    .21        res-m := res-m † {nm-pre ↦ mk- (fn-pre, RealAccess (access, OP))};
    .22   return res-m );
```

```
1090.0 DoCreateExtExplOpFns : AS`Name × AS`ExtExplOpDef × [AS`StateDef] -o->
    .1                        AS`Name -m-> (SEM`CompExplFN × AS`Access)
    .2 DoCreateExtExplOpFns (mod-id, mk-AS`ExtExplOpDef (nm, partps, resnmtps, -, -, Pre-e,
    .3                                         Post-e, -, access, static, constr,
    .4                                         -), gst) ≜
    .5   let mk- (opdom, parms) = ImplicitTypeParams (partps),
```

```
.6        mk-(resnms, restps) = ImplicitResNameTypes(resnmtps) in
.7      let st-id = if gst = nil
.8                  then nil
.9                  else gst.tp.name in
.10     (dcl res-m : AS`Name -m-> (SEM`CompExplFN × AS`Access) := {↦};
.11      if Pre-e = nil
.12      then skip
.13      else let nm-pre = AUX`PreName(nm),
.14               pl = CreateOperationPreParms(parms, gst) in
.15           let fn-pre = SEM`CompFN
.16                          (
.17                           mk-SEM`ExplFN(TransType(CreateOperationPreType(opdom, st-id)),
.18                                         [CPAT`PL2PL(pl(i)) | i ∈ inds pl],
.19                                         CMPL`CompilePrePostExpr(Pre-e),
.20                                         AUX`MkEmptyBlkEnv(READ_ONLY),
.21                                         {↦},
.22                                         nm-pre,
.23                                         mod-id, nil , nil )) in
.24           res-m := res-m † {nm-pre ↦ mk-(fn-pre, RealAccess(access, OP))};
.25      return res-m );
```

Togehter these operations create the semantic function value for the pre and/or post predicate functions for an explicit operation definition. The closure environment and the type instantiation map in the semantic function values are both empty. In case of a post predicate function, we replace all occurences of old names with normal names, but only if a state is defined.

```
1091.0  CreateImplOpFns : AS`Name × AS`ImplOpDef × [AS`StateDef] -o->
    .1                    AS`Name -m-> (SEM`CompExplFN × AS`Access)
    .2  CreateImplOpFns(mod-id, mk-AS`ImplOpDef(name, partps, resnmtps, -, Pre-e,
    .3                                    Post-e, -, access, static, constr, -), gst) △
    .4    let mk-(fn-tp, fn-parm) = ImplicitTypeParams(partps),
    .5        mk-(res-nms, res-tps) = ImplicitResNameTypes(resnmtps),
    .6        st-id = if gst = nil
    .7                then nil
    .8                else gst.tp.name in
    .9    (dcl res-m : AS`Name -m-> (SEM`CompExplFN × AS`Access) := {↦};
    .10    if Pre-e = nil
    .11    then skip
    .12    else let nm-pre = AUX`PreName(name),
```

.13 $\quad\quad pi\text{-}l = CreateOperationPreParms\,(fn\text{-}parm, gst)$ in
.14 $\quad$ let $fn\text{-}pre = SEM`CompFN$
.15 $\quad\quad\quad ($
.16 $\quad\quad\quad$ mk-$SEM`ExplFN\,(TransType\,(CreateOperationPreType\,(fn\text{-}tp, st\text{-}id)),$
.17 $\quad\quad\quad\quad [CPAT`PL2PL\,(pi\text{-}l\,(i)) \mid i \in$ inds $pi\text{-}l],$
.18 $\quad\quad\quad\quad CMPL`CompilePrePostExpr\,(Pre\text{-}e),$
.19 $\quad\quad\quad\quad AUX`MkEmptyBlkEnv\,(\mathrm{READ\_ONLY}),$
.20 $\quad\quad\quad\quad \{\mapsto\},$
.21 $\quad\quad\quad\quad nm\text{-}pre,$
.22 $\quad\quad\quad\quad mod\text{-}id,$ nil , nil)) in
.23 $\quad res\text{-}m := res\text{-}m \dagger \{nm\text{-}pre \mapsto$ mk- $(fn\text{-}pre, RealAccess\,(access, \mathrm{OP}))\};$
.24 return $res\text{-}m$);

This operation creates the semantic function values for the pre and/or post predicate function for an implicit operation definition.

1.18.7 Translation of Record Selector Information

1092.0 $TransTP : (AS`Type \times AS`Access)$-set $\xrightarrow{o}$
.1 $\quad AS`Name \xrightarrow{m} ((\mathbb{N} \times (AS`Name \xrightarrow{m} \mathbb{N}) \times AS`Type^{*}) \times AS`Access)$
.2 $TransTP\,(typeset) \triangleq$
.3 if $typeset = \{\}$
.4 then return $\{\mapsto\}$

.5 else let mk- $(type, access) \in typeset$ in
.6 let $tmp =$
.7 cases $type$:
.8 mk-$AS`CompositeType\,(nm, fields, \text{-}) \rightarrow$
.9 $\{nm \mapsto$ mk- $(TCompT\,(fields), access)\} \uplus$
.10 $TransTP\,(\{$mk- $(fields\,(i).type, access) \mid$
.11 $i \in$ inds $fields\}),$
.12 mk-$AS`UnionType\,(typeset, \text{-}) \rightarrow$
.13 $TransTP\,(\{$mk- $(t, access) \mid t \in$ elems $typeset\}),$
.14 mk-$AS`ProductType\,(typeseq, \text{-}) \rightarrow$
.15 $TransTP\,(\{$mk- $(t, access) \mid t \in$ elems $typeseq\}),$
.16 mk-$AS`OptionalType\,(type', \text{-}),$
.17 mk-$AS`SetType\,(type', \text{-}),$
.18 mk-$AS`BracketedType\,(type', \text{-}),$
.19 mk-$AS`Seq0Type\,(type', \text{-}),$
.20 mk-$AS`Seq1Type\,(type', \text{-}) \rightarrow$
.21 $TransTP\,(\{$mk- $(type', access)\}),$
.22 mk-$AS`GeneralMapType\,(dt, rt, \text{-}),$
.23 mk-$AS`InjectiveMapType\,(dt, rt, \text{-}) \rightarrow$
.24 $TransTP\,(\{$mk- $(dt, access),$ mk- $(rt, access)\}),$
.25 mk-$AS`PartialFnType\,(dt, rt, \text{-}),$
.26 mk-$AS`TotalFnType\,(dt, rt, \text{-}) \rightarrow$
.27 let $dtm =$ if $dt = []$
.28 then $\{\mapsto\}$
.29 else $TransTP\,(\{$mk- $(t, access) \mid t \in$ elems $dt\}),$
.30 $rtm = TransTP\,(\{$mk- $(rt, access)\})$ in
.31 $dtm \uplus rtm,$
.32 others $\rightarrow \{\mapsto\}$
.33 end in
.34 let $inc\text{-}access = \{nm \mapsto tmp\,(nm) \mid nm \in$ dom $tmp\}$ in
.35 return $inc\text{-}access \uplus TransTP\,(typeset \setminus \{$mk- $(type, access)\})$;

This operation analyses a set of types and creates for all the composite types the record selectors. For all non-composite types, $TransTP$ is called recursively.

1093.0 $TransType : AS`Type \xrightarrow{o} AS`Type$
.1 $TransType\,(tp) \triangleq$
.2 return tp;

1094.0 $TCompT : AS`Field^* \xrightarrow{o} \mathbb{N} \times (AS`Name \xrightarrow{m} \mathbb{N}) \times AS`Type^*$
.1 $TCompT\,(fields) \triangleq$
.2 if len $[fields\,(i) \mid i \in$ inds $fields \cdot fields\,(i).sel \neq$ nil $] \neq$
.3 card $\{fields\,(i).sel \mid i \in$ inds $fields \cdot fields\,(i).sel \neq$ nil $\}$
.4 then $(RTERR`InitError(RTERR`IDENTICAL\text{-}FIELDS,$ (hd $fields).cid)$;
.5 return mk- $(0, \{\mapsto\}, [])$)

```
.6     else return mk- (len fields,
                         {fields (i).sel ↦ i |
                              i ∈ inds fields · fields (i).sel ≠ nil },
                         [fields (i).type | i ∈ inds fields]) ;
```

This operation creates the record selector for the fields of a composite type. The record selector consists of three fields. The first field contains the number of fields in the composite type, the second field is a map from field identifier to field position, and the last field is a sequence with the type of each field in the composite type.

1.18.8 Translation of Invariants

```
1095.0 CreateInvs : AS`Name × AS`Name -m-> AS`TypeDef -o->
                   AS`Name -m-> (SEM`CompExplFN × AS`Access)
  .2   CreateInvs (mod-id, tm) △
         let nid = CI`NilContextId in
         (dcl tmp : AS`Name -m-> (SEM`CompExplFN × AS`Access) := {↦};
          for all name ∈ dom tm
          do let mk-AS`TypeDef (-, shape, Inv, access, -) = tm (name) in
             if Inv ≠ nil
             then let mk-AS`Invariant (pattern, expr, -) = Inv,
                         mk-AS`Name (id, cid) = name in
                    tmp := tmp †
                          {mk-AS`Name (["inv-" ⁀ id (1)], cid) ↦
                           mk- (TransFN (mod-id,
                                         mk-AS`ExplFnDef
                                           (
                                            name,
                                            [],
                                            mk-AS`TotalFnType
                                              (
                                               [shape],
                                               mk-AS`BasicType (BOOLEAN, nid), nid),
                                            [[pattern]],
                                            mk-AS`FnBody (expr, CI`NilContextId),
                                            nil ,
                                            nil ,
                                            Default-Tp,
                                            false,
                                            nid)),
                                RealAccess (access, TP))};
          return tmp )
```

values

```
1096.0 Default-Op : AS`Access = PRIVATE_AS;
```

1097.0 $\textit{Default-Fn} : \textit{AS}`\textit{Access} = \mathrm{PRIVATE\_AS};$

1098.0 $\textit{Default-Val} : \textit{AS}`\textit{Access} = \mathrm{PRIVATE\_AS};$

1099.0 $\textit{Default-Tp} : \textit{AS}`\textit{Access} = \mathrm{PRIVATE\_AS};$

1100.0 $\textit{Default-Inst} : \textit{AS}`\textit{Access} = \mathrm{PRIVATE\_AS}$

functions

1101.0 $\textit{RealAccess} : \textit{AS}`\textit{Access} \times (\mathrm{OP} \mid \mathrm{FN} \mid \mathrm{VAL} \mid \mathrm{TP} \mid \mathrm{INST}) \to \textit{AS}`\textit{Access}$
.1 $\textit{RealAccess}\,(\textit{access}, \textit{kind}) \triangleq$
.2 if $\textit{access} = \mathrm{DEFAULT\_AS}$
.3 then cases $\textit{kind}$:
.4 $\mathrm{OP} \to \textit{Default-Op}$,
.5 $\mathrm{FN} \to \textit{Default-Fn}$,
.6 $\mathrm{VAL} \to \textit{Default-Val}$,
.7 $\mathrm{TP} \to \textit{Default-Tp}$,
.8 $\mathrm{INST} \to \textit{Default-Inst}$
.9 end
.10 elseif $\textit{access} = \mathrm{NOT\_INITIALISED\_AS}$
.11 then $\mathrm{PRIVATE\_AS}$
.12 else $\textit{access}$

end $\textit{DEF}$

The operation *CreateInvs* returns the semantic function values for the invariant predicate functions for the type definitions in a definitions block.

Test Suite : rtinfo.ast
Module : DEF

Name	**#Calls**	**Coverage**
DEF‘TCompT	**undefined**	**undefined**
DEF‘TransFN	**undefined**	**undefined**
DEF‘TransOP	**undefined**	**undefined**
DEF‘TransTP	**undefined**	**undefined**
DEF‘TransType	**undefined**	**undefined**
DEF‘CreateInvs	**undefined**	**undefined**
DEF‘RealAccess	**undefined**	**undefined**
DEF‘TransFnMap	**undefined**	**undefined**
DEF‘TransOpMap	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
DEF‘UpdateType	**undefined**	**undefined**
DEF‘ReadClasses	**undefined**	**undefined**
DEF‘LookUpLocalFn	**undefined**	**undefined**
DEF‘TransInstVars	**undefined**	**undefined**
DEF‘UpdateTypeDefs	**undefined**	**undefined**
DEF‘CreateExplOpFns	**undefined**	**undefined**
DEF‘CreateImplOpFns	**undefined**	**undefined**
DEF‘CreatePostParms	**undefined**	**undefined**
DEF‘EmptySigmaClass	**undefined**	**undefined**
DEF‘ImplicitResType	**undefined**	**undefined**
DEF‘TransOverloaded	**undefined**	**undefined**
DEF‘MergeFnsOpsPolys	**undefined**	**undefined**
DEF‘TransInheritance	**undefined**	**undefined**
DEF‘TransLocalRecSel	**undefined**	**undefined**
DEF‘DoCreateExplOpFns	**undefined**	**undefined**
DEF‘CreateSigmaPattern	**undefined**	**undefined**
DEF‘ImplicitTypeParams	**undefined**	**undefined**
DEF‘UpdateConstructors	**undefined**	**undefined**
DEF‘CreateExplPrePostFns	**undefined**	**undefined**
DEF‘CreateImplPrePostFns	**undefined**	**undefined**
DEF‘DoCreateExtExplOpFns	**undefined**	**undefined**
DEF‘ExtractStaticMembers	**undefined**	**undefined**
DEF‘ImplicitResNameTypes	**undefined**	**undefined**
DEF‘CreateFunctionPostType	**undefined**	**undefined**
DEF‘CreateOperationPreType	**undefined**	**undefined**
DEF‘DoCreateExplPrePostFns	**undefined**	**undefined**
DEF‘CreateOperationPostType	**undefined**	**undefined**
DEF‘CreateOperationPreParms	**undefined**	**undefined**
DEF‘CreateExplPolyPrePostFns	**undefined**	**undefined**
DEF‘CreateImplPolyPrePostFns	**undefined**	**undefined**
DEF‘CreateOperationPrePostFns	**undefined**	**undefined**
DEF‘DoCreateExtExplPrePostFns	**undefined**	**undefined**
DEF‘DoCreateExplPolyPrePostFns	**undefined**	**undefined**
DEF‘CreateExplOperationPostParms	**undefined**	**undefined**
DEF‘CreateImplOperationPostParms	**undefined**	**undefined**
DEF‘DoCreateExtExplPolyPrePostFns	**undefined**	**undefined**
DEF‘CreateExtExplOperationPostParms	**undefined**	**undefined**
Total Coverage		**0%**

1.19 Expressions

The module EXPR contains functions and operations related to the evaluation of expressions.
module *EXPR*

imports

1102.0 from AS all ,

1103.0 from CI all ,

1104.0 from AUX

1105.0 functions $ExtractId : AS`Name \to AS`Id$ renamed $ExtractId$;
.1 $ExtractName : AS`Name \to AS`Name$;
.2 $ConstructName : AS`Id \times CI`ContextId \to AS`Name$

1106.0 operations $IsInt : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsInt$;
.1 $IsNat : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsNat$;
.2 $IsRat : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsRat$;
.3 $IsReal : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsReal$;
.4 $Ceiling : \mathbb{R} \xrightarrow{o} \mathbb{Z}$ renamed $Ceiling$;
.5 $ErrorOp$: char* $\xrightarrow{o}$
.6 $\mathbb{B} \mid SEM`VAL \mid AS`Name$-set $\mid$
.7 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL]) \mid AS`Name \mid$
.8 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL] \times [SEM`VAL] \times [AS`Name] \times [AS`Access]) \mid$
.9 $SEM`BlkEnv$-set $\mid (\mathbb{B} \times [GLOBAL`Type] \times [AS`Invariant] \times [AS`Name]) \mid$

.10 $GLOBAL`RecSel$ renamed $ErrorOp$;
.11 $IsNatOne : SEM`VAL \xrightarrow{o} \mathbb{B}$ renamed $IsNatOne$;
.12 $IsRecSel : AS`Name \xrightarrow{o} \mathbb{B} \times [AS`Name \xrightarrow{m} \mathbb{N}]$;
.13 $MkBlkEnv : AS`Name \times SEM`VAL \times [AS`Type] \times SEM`Permission \xrightarrow{o}$
.14 $SEM`BlkEnv$ renamed $MkBlkEnv$;
.15 $SetToSeq : SEM`VAL$-set $\xrightarrow{o} SEM`VAL^*$ renamed $SetToSeq$;
.16 $IsTypeDef : AS`Name \xrightarrow{o}$
.17 $\mathbb{B} \times [GLOBAL`Type] \times [AS`Invariant] \times [AS`Name] \times [AS`Access]$;
.18 $ValSetToSeq : SEM`VAL$-set $\xrightarrow{o} SEM`VAL^*$ renamed $ValSetToSeq$;
.19 $LookUpRecSel : AS`Name \xrightarrow{o} [GLOBAL`RecSel]$ renamed $LookUpRecSel$;
.20 $CombineBlkEnv : SEM`BlkEnv \times SEM`BlkEnv \xrightarrow{o} SEM`BlkEnv$ renamed $CombineBlkEnv$;
.21 $MkEmptyBlkEnv : (SEM`Permission) \xrightarrow{o} SEM`BlkEnv$ renamed $MkEmptyBlkEnv$;
.22 $ExtractTagName : AS`Name \xrightarrow{o} [AS`Name] \times \mathbb{B}$,

1107.0 from PAT

1108.0 types $PARTITION$

1109.0 operations $GetExpr : AS`Pattern \xrightarrow{o} AS`Expr$ renamed $GetExpr$;
.1 $MatchLists : STKM`Pattern^* \times SEM`VAL^* \xrightarrow{o} SEM`BlkEnv$-set renamed $MatchLists$;
.2 $SelPattern : AS`Bind \xrightarrow{o} AS`Pattern$ renamed $SelPattern$;
.3 $ConstructFN : AS`FnDef \xrightarrow{o} SEM`BlkEnv \times AS`Name \xrightarrow{m} (AS`Expr \mid$
NOTYETSPEC |
.4 SUBRESP) renamed $ConstructFN$;
.5 $PatternMatch : STKM`Pattern \times SEM`VAL \xrightarrow{o} SEM`BlkEnv$-set renamed $PatternMatch$;
.6 $DoCarePattern : AS`PatternBind \times AS`Name \xrightarrow{o} AS`PatternBind$ renamed $DoCarePattern$;
.7 $EvalMultBindSeq : STKM`Pattern^* \times SEM`VAL^* \times PAT`PARTITION \xrightarrow{o}$

.8 $SEM`BlkEnv$-set $\mid (SEM`BlkEnv$-set$)$-set renamed $EvalMultBindSeq$, 1110.0 from POS all ,

1111.0 from REP all ,

1112.0 from SEM all ,

1113.0 from $FREE$

1114.0 operations $FreeInExpr{:}AS`Expr \times AS`Name\text{-set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$ renamed $FreeInExpr$;
.1 $IdentInPattern{:}AS`PatternBind \xrightarrow{o} AS`Name\text{-set}$ renamed $IdentInPattern$;
.2 $FreeMapToBlkEnv{:}AS`Name \xrightarrow{m} SEM`VAL \xrightarrow{o} SEM`BlkEnv$ renamed $FreeMapToBlkEnv$
,

1115.0 from $STKM$ all ,

1116.0 from $RTERR$ all ,

1117.0 from $STATE$ all ,

1118.0 from $GLOBAL$ all ,

1119.0 from $SCHDTP$ all ,

1120.0 from $INSTRTP$ all ,

1121.0 from $TIMEMAP$ all ,

1122.0 from $SETTINGS$

1123.0 operations $DTC : () \xrightarrow{o} \mathbb{B}$ renamed DTC ,

1124.0 from $TIMEPARSER$ all

exports all

definitions

1.19.1 Self Expression

operations

1125.0 $EvalSelfExpr : AS`SelfExpr \xrightarrow{o} SEM`OBJ\text{-}Ref$
.1 $EvalSelfExpr$ (-) $\triangleq$
.2 return $STKM`GetCurObjRef$ ();

The operation *EvalSelfExpr* returns the semantic value of the current object.

1126.0 $EvalSetRangeExpr : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$
.1 $EvalSetRangeExpr\ (lb\text{-}v, ub\text{-}v) \triangleq$
.2 if is-$SEM`NUM\ (lb\text{-}v)$
.3 then if is-$SEM`NUM\ (ub\text{-}v)$
.4 then let mk-$SEM`NUM\ (lb) = lb\text{-}v$,
.5 mk-$SEM`NUM\ (ub) = ub\text{-}v$ in
.6 return mk-$SEM`SET$ ({mk-$SEM`NUM\ (e)$ |
.7 $e \in \{Ceiling\ (lb), \ldots,$ floor $ub\}\})$
.8 else $RTERR`ErrorVal(RTERR`UPPER\text{-}BOUND\text{-}NOT\text{-}A\text{-}NUMBER, ub\text{-}v,$ nil , [])

.9 $\quad \mathsf{else}\ RTERR`ErrorVal(RTERR`LOWER\text{-}BOUND\text{-}NOT\text{-}A\text{-}NUMBER, lb\text{-}v, \mathsf{nil}\ , [\,]);$

This operation returns the set of semantic number values, ranging from a lower to an upper bound.

1127.0 $\quad EvalSubSequenceExpr : SEM`VAL \times SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $\quad EvalSubSequenceExpr\ (seq\text{-}v, from\text{-}v, to\text{-}v) \triangleq$
.2 $\quad\quad \mathsf{if}\ \mathsf{is}\text{-}SEM`SEQ\ (seq\text{-}v) \wedge \mathsf{is}\text{-}SEM`NUM\ (from\text{-}v) \wedge \mathsf{is}\text{-}SEM`NUM\ (to\text{-}v)$
.3 $\quad\quad \mathsf{then\ let\ mk}\text{-}SEM`NUM\ (From) = from\text{-}v,$
.4 $\quad\quad\quad\quad \mathsf{mk}\text{-}SEM`NUM\ (To) = to\text{-}v,$
.5 $\quad\quad\quad\quad \mathsf{mk}\text{-}SEM`SEQ\ (Seq) = seq\text{-}v\ \mathsf{in}$
.6 $\quad\quad\quad \mathsf{return\ mk}\text{-}SEM`SEQ\ ([Seq\ (i) \mid i \in \mathsf{inds}\ Seq \cdot From \leq i \wedge i \leq To])$
.7 $\quad\quad \mathsf{elseif}\ \neg\, \mathsf{is}\text{-}SEM`SEQ\ (seq\text{-}v)$
.8 $\quad\quad \mathsf{then}\ RTERR`ErrorVal(RTERR`SEQ\text{-}EXPECTED, seq\text{-}v, \mathsf{nil}\ , [\,])$
.9 $\quad\quad \mathsf{elseif}\ \neg\, \mathsf{is}\text{-}SEM`NUM\ (from\text{-}v)$
.10 $\quad\quad \mathsf{then}\ RTERR`ErrorVal(RTERR`INT\text{-}EXPECTED, from\text{-}v, \mathsf{nil}\ , [\,])$
.11 $\quad\quad \mathsf{else}\ RTERR`ErrorVal(RTERR`INT\text{-}EXPECTED, to\text{-}v, \mathsf{nil}\ , [\,])\ ;$

This operation returns the semantic sequence value that contains all the elements from the input sequence with indices in a given range.

1128.0 $\quad EvalSeqModifyMapOverrideExpr : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $\quad EvalSeqModifyMapOverrideExpr\ (seqmap\text{-}v, map\text{-}v) \triangleq$
.2 $\quad\quad \mathsf{if}\ \neg\, \mathsf{is}\text{-}SEM`MAP\ (map\text{-}v)$
.3 $\quad\quad \mathsf{then}\ RTERR`ErrorVal(RTERR`MAP\text{-}EXPECTED, \mathsf{nil}\ , \mathsf{nil}\ , [\,])$
.4 $\quad\quad \mathsf{elseif}\ \mathsf{is}\text{-}SEM`MAP\ (seqmap\text{-}v)$
.5 $\quad\quad \mathsf{then}\ EvalMapOverrideExpr(seqmap\text{-}v, map\text{-}v)$
.6 $\quad\quad \mathsf{elseif}\ \mathsf{is}\text{-}SEM`SEQ\ (seqmap\text{-}v)$
.7 $\quad\quad \mathsf{then}\ EvalSeqModifyExpr(seqmap\text{-}v, map\text{-}v)$
.8 $\quad\quad \mathsf{else}\ RTERR`ErrorVal(RTERR`MAP\text{-}OR\text{-}SEQ\text{-}EXPECTED, \mathsf{nil}\ , \mathsf{nil}\ , [\,])\ ;$

This operation calls one of the operation *EvalMapOverrideExpr* or *EvalSeqModifyExpr*, depending on the value of the first argument.

1129.0 $\quad EvalMapOverrideExpr : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $\quad EvalMapOverrideExpr\ (\mathsf{mk}\text{-}SEM`MAP\ (op1), \mathsf{mk}\text{-}SEM`MAP\ (op2)) \triangleq$
.2 $\quad\quad \mathsf{return\ mk}\text{-}SEM`MAP\ (op1 \dagger op2);$

The result of this operation is a semantic map value, that contains all the maplets of the second map, and those maplets from the first map, for which the domain element is not contained in the domain set of the second map.

```
1130.0 EvalSeqModifyExpr : SEM`VAL × SEM`VAL -o-> SEM`VAL
EvalSeqModifyExpr (seq-v, modifier-sv) △
  (dcl tmp-s : (SEM`VAL × SEM`VAL)-set := {},
       res-lv : SEM`VAL* := [];
   let mk-SEM`SEQ (seq-lv) = seq-v,
       mk-SEM`MAP (modifiers) = modifier-sv in
   (res-lv := seq-lv;
    for all d-v ∈ dom modifiers
    do let r-v = modifiers (d-v) in
       if IsInt (d-v)
       then let mk-SEM`NUM (d) = d-v in
            if (d ≥ 1 ∧ d ≤ len seq-lv) ∧
               (∀ mk- (d1, r1) ∈ tmp-s · if d1 = d-v
                                         then r1 = r-v
                                         else true)
            then (tmp-s := tmp-s ∪ {mk- (d-v, r-v)};
                  res-lv := res-lv † {d ↦ r-v})
            else RTERR`ErrorVal(RTERR`ILLEGAL-INDICES, nil , nil , [])
       else RTERR`ErrorVal(RTERR`INT-EXPECTED, nil , nil , []) ;
    return mk-SEM`SEQ (res-lv) ));
```

The result semantic sequence value of this operation is equal to the old input sequence value, but for every index from the domain of the modifier map, the corresponding element in the sequence value is replaced with the range value from the map. Checks are made to be sure that the domain elements are all natural numbers, and that the map is injective.

1.19.2 Record Expressions

The operation *EvalRecordConstructorExpr* evaluated record constructor expressions. Initially, it is investigated by the operation *AUX`ExtractTagName* if the tag *tag* is in the current scope, and if it is the tag name is computed. The tag name *tagname* is the tag name qualified with the class name where type is defined.

The semantics values of the *fields* are then computed, and it is checked if the size of the record corresponds to the length of the fields. If dynamic type check the check is done in two ways depending if the type has a type definition, or not. The latter can be the case in case of an let expression of the following form:

```
let e: compose A of a:int, b nat end = mk_A(-1,2)
in ....
```

1131.0 $EvalRecordConstructorExpr : AS`Name \times SEM`VAL^* \xrightarrow{o} SEM`VAL$
.1 $EvalRecordConstructorExpr\,(tag, fields) \triangleq$
.2 let mk- $(tagname, isinscope\text{-}tag) = AUX`ExtractTagName\,(tag)$ in
.3 if $\neg\, isinscope\text{-}tag$
.4 then $RTERR`ErrorVal(RTERR`TAG\text{-}UNKNOWN,$ nil , nil , [])
.5 else let mk- $(size, \text{-}, type\text{-}l) = LookUpRecSel\,(tagname)$ in
.6 if $\neg$ len $fields = size$
.7 then $RTERR`ErrorVal(RTERR`RECORD\text{-}SIZE\text{-}WRONG,$ nil , nil , [])
.8 else let $res\text{-}v = ConstructSEMRecFields\,(tagname, fields)$ in
.9 if $\neg\, DTC\,()$
.10 then return $res\text{-}v$
.11 else let mk- $(isit, \text{-}, \text{-}, \text{-}, \text{-}) = AUX`IsTypeDef\,(tagname)$ in
.12 if $isit$
.13 then if $STATE`SubType\,(res\text{-}v,$
.14 mk-$AS`TypeName\,(tagname, tagname.cid))$
.15 then return $res\text{-}v$
.16 else $RTERR`ErrorVal(RTERR`TYPE\text{-}INCOMP,$
.17 $res\text{-}v,$ nil , [])
.18 else if $\forall\, i \in$ inds $type\text{-}l \cdot$
.19 let mk-$SEM`REC\,(\text{-}, v, v\text{-}dc) = res\text{-}v$ in
.20 (if $i \in$ dom v
.21 then $STATE`SubType\,(v\,(i), type\text{-}l\,(i))$
.22 else $STATE`SubType\,(v\text{-}dc\,(i), type\text{-}l\,(i)))$
.23 then return $res\text{-}v$
.24 else $RTERR`ErrorVal(RTERR`TYPE\text{-}INCOMP,$
.25 $res\text{-}v,$ nil , []) ;

The operation *ConstructSEMRecFields* is an auxiliary operation that computes a record semantic value.

1132.0 $ConstructSEMRecFields : AS`Name \times SEM`VAL^* \xrightarrow{o} SEM`VAL$
.1 $ConstructSEMRecFields\,(sem\text{-}tag, val\text{-}l) \triangleq$
.2 (dcl $v : \mathbb{N} \xrightarrow{m} SEM`VAL,$
.3 $v\text{-}dc : \mathbb{N} \xrightarrow{m} SEM`VAL;$
.4 let mk- $(isit, type, \text{-}, \text{-}, \text{-}) = AUX`IsTypeDef\,(sem\text{-}tag)$ in
.5 if $isit$
.6 then let mk-$AS`CompositeType\,(\text{-}, fl, \text{-}) = type$ in
.7 $(v := \{i \mapsto val\text{-}l\,(i) \mid i \in$ inds $fl \cdot \neg\, fl\,(i).dc\};$
.8 $v\text{-}dc := \{i \mapsto val\text{-}l\,(i) \mid i \in$ inds $fl \cdot fl\,(i).dc\})$
.9 else let mk- $(ok, \text{-}) = AUX`IsRecSel\,(sem\text{-}tag)$ in
.10 if ok
.11 then $(v := \{i \mapsto val\text{-}l\,(i) \mid i \in$ inds $val\text{-}l\};$
.12 $v\text{-}dc := \{\mapsto\})$
.13 else $RTERR`ErrorVal(RTERR`TYPE\text{-}NOT\text{-}IN\text{-}SCOPE,$ nil , nil , []) ;
.14 return mk-$SEM`REC\,(sem\text{-}tag, v, v\text{-}dc)$);

The operatin *EvalRecordModifierExpr* evaluates record modification expressions. The semantic value of the record *rec-e* is first computed. It is computed if the tag *tag* of the record is within the current scope, and the full tag name *tagname* is computed. The full *tag name* is the tag name qualified with the class name where the record type is defined. For each of the modifications, we change the original record value. Checks are made to be sure that the field identifiers in the modification records are defined for the input record value. Also, we use dynamic type checking on the new values.

```
EvalRecordModifierExpr : SEM`VAL × (AS`Name*) × (SEM`VAL*) -o->
                                  SEM`VAL

EvalRecordModifierExpr (rec-v, fid-l, val-l) △
  (if ¬ is-SEM`REC (rec-v)
   then RTERR`ErrorVal(RTERR`REC-EXPECTED, rec-v, nil , [])
   else let mk-SEM`REC (tag, v, v-dc) = rec-v in
        let mk- (tagname, isinscope-tag) = AUX`ExtractTagName (tag) in
        if ¬ isinscope-tag
        then RTERR`ErrorVal(RTERR`RECORD-TAG-UNKNOWN, rec-v, nil , [])
        else let mk- (-, pos, type-l) = LookUpRecSel (tagname) in
             (dcl tmp-v : ℕ -m-> SEM`VAL := v,
                  tmp-v-dc : ℕ -m-> SEM`VAL := v-dc;
              for i = 1 to len fid-l
              do let fid = fid-l (i),
                     new-v = val-l (i) in
                 if fid ∉ dom pos
                 then RTERR`ErrorVal(RTERR`RECORD-FIELD-ID-UNKNOWN, nil , nil , [])
                 else if pos (fid) ∈ dom v
                      then tmp-v := tmp-v † {pos (fid) ↦ new-v}
                      else tmp-v-dc := tmp-v-dc † {pos (fid) ↦ new-v};
              let res-v = mk-SEM`REC (tagname, tmp-v, tmp-v-dc) in
              if DTC ()
              then let mk- (isit, td, -, -, -) = AUX`IsTypeDef (tagname) in
                   if isit
                   then if STATE`SubType (res-v, td)
                        then return res-v
                        else RTERR`ErrorVal(RTERR`TYPE-INCOMP, nil , nil , [])
                   else if ∀ i ∈ inds type-l ·
                                if i ∈ dom tmp-v
                                then STATE`SubType (tmp-v (i), type-l (i))
                                else STATE`SubType (tmp-v-dc (i), type-l (i))
                        then return res-v
                        else RTERR`ErrorVal(RTERR`TYPE-INCOMP, nil , nil , [])
              else return res-v ));
```

1134.0 $EvalSeqApply : SEM`SEQ \times SEM`VAL^* \xrightarrow{o} SEM`VAL$

.1 $EvalSeqApply$ (mk-$SEM`SEQ\,(seq\text{-}v), arg\text{-}lv) \triangleq$
.2 if len $arg\text{-}lv = 1$
.3 then if $IsNat\,(arg\text{-}lv\,(1))$
.4 then let mk-$SEM`NUM\,(arg) = arg\text{-}lv\,(1)$ in
.5 if $arg \in$ inds $seq\text{-}v$
.6 then return $seq\text{-}v\,(arg)$
.7 else error
.8 else error
.9 else error;

The evaluation of a sequence application returns the value from the input sequence at the given index.

1135.0 $EvalMapApply : SEM`MAP \times SEM`VAL^* \xrightarrow{o} SEM`VAL$

.1 $EvalMapApply$ (mk-$SEM`MAP\,(map\text{-}v), arg\text{-}lv) \triangleq$
.2 if len $arg\text{-}lv = 1$
.3 then if $arg\text{-}lv\,(1) \in$ dom $map\text{-}v$
.4 then return $map\text{-}v\,(arg\text{-}lv\,(1))$
.5 else error
.6 else error;

This operation returns the range value from the input map for a given domain value.

1136.0 $EvalFieldSelectExpr : SEM`VAL \times (AS`Name \mid AS`FctTypeInstExpr) \xrightarrow{o}$
.1 $SEM`VAL$

.2 $EvalFieldSelectExpr\,(record\text{-}v, field) \triangleq$
.3 cases true:
.4 (is-$SEM`REC\,(record\text{-}v)) \rightarrow EvalFieldRecordSelect(record\text{-}v, field)$,
.5 (is-$SEM`OBJ\text{-}Ref\,(record\text{-}v)) \rightarrow EvalFieldOBJRefSelect(record\text{-}v,$
.6 $field)$,
.7 others $\rightarrow RTERR`ErrorVal(RTERR`OBJ\text{-}RECORD\text{-}EXP, record\text{-}v,$ nil , [])
.8 end;

The operation *EvalFieldOBJRefSelect* evaluates the field select expression of the field *field* of the semantic object reference *obj_ref*.

1137.0 $EvalFieldOBJRefSelect : SEM`OBJ\text{-}Ref \times (AS`Name \mid AS`FctTypeInstExpr) \xrightarrow{o}$
.1 $SEM`VAL$

.2 $EvalFieldOBJRefSelect\,(objref, field) \triangleq$
.3 (dcl $res\text{-}v : SEM`VAL$;

```
.4     STKM`PushEmptyEnv() ;
.5     STKM`PushCurObj(objref, STATE`GetClFromObjRef (objref), STKM`GetCurCl ());
.6     res-v := if is-AS`Name (field)
.7              then STATE`LookUp (field)
.8              else let mk-AS`FctTypeInstExpr (polynm, inst, -) = field in
.9                   let polyval = STATE`LookUp (polynm) in
.10                  STATE`EvalFctTypeInstExpr (polyval, inst);
.11    STKM`PopEnvL() ;
.12    if is-SEM`ExplOP (res-v) ∨
.13      is-SEM`CompExplFN (res-v) ∨
.14      is-SEM`ExplPOLY (res-v) ∨
.15      is-SEM`OverOPFN (res-v)
.16    then res-v.objref := objref;
.17    STKM`PopCurObj() ;
.18    return res-v );
```

The operation *EvalFieldRecordSelect* evaluates the field selection *field* of the semantic record value *record_v*.

```
1138.0 EvalFieldRecordSelect : SEM`REC × AS`Name -o-> SEM`VAL
.1     EvalFieldRecordSelect (record-v, field) △
.2       let mk-SEM`REC (tag, v, v-dc) = record-v in
.3       let mk- (tagname, isinscope-tag) = AUX`ExtractTagName (tag) in
.4       if isinscope-tag
.5       then let mk- (-, pos, -) = LookUpRecSel (tagname) in
.6            if field ∈ dom pos
.7            then if pos (field) ∈ dom v
.8                 then return v (pos (field))
.9                 else return v-dc (pos (field))
.10           else RTERR`ErrorVal(RTERR`RECORD-FIELD-ID-UNKNOWN, nil , nil , [])
.11      else error;
```

This operation returns the value for the given field identifier of the input record value. If the type structure is not available, an error is generated.

```
1139.0 ConvertPolyToFn : SEM`ExplPOLY × AS`Type* -o-> SEM`CompExplFN
.1     ConvertPolyToFn (mk-SEM`ExplPOLY (tpp, tp, parms, instr, env,
.2                          fnName, modName, resvars, objref), type-l) △
.3       if len tpp = len type-l
.4       then let fndom = [SubstType (tp.fndom (i), tpp, type-l) |
.5                              i ∈ inds tp.fndom],
.6                fnrng = SubstType (tp.fnrng, tpp, type-l),
```

```
            tm = {tpp (i) ↦ type-l (i) | i ∈ inds tpp} in
        let newtp = if is-AS‘PartialFnType (tp)
                    then mk-AS‘PartialFnType (fndom, fnrng, CI‘NilContextId)
                    else mk-AS‘TotalFnType (fndom, fnrng, CI‘NilContextId) in
        return SEM‘CompFN
                  (
                  mk-SEM‘ExplFN (newtp, parms, instr, env, tm,
                                 fnName, modName, resvars, objref))
    else error;
```

This operation returns a semantic function value, in which all type variable identifiers in the type of the function are replaced with normal types.

```
SubstType : AS‘Type × AS‘TypeVarList × AS‘Type* ⥲
            AS‘Type
SubstType (tp, tv-l, tp-l) ≜
  cases true:
    (is-AS‘BasicType (tp)),
    (is-AS‘QuoteType (tp)),
    (is-AS‘TypeName (tp)) → return tp,
    (is-AS‘BracketedType (tp)) → let mk-AS‘BracketedType (btp, cid) = tp in
                                 return mk-AS‘BracketedType (SubstType (btp, tv-l, tp-l), cid),
    (is-AS‘CompositeType (tp)) → let mk-AS‘CompositeType (id, fields, cid) = tp in
                                 (dcl newf : AS‘Field* := [];
                                  for mk-AS‘Field (sel, ftp, dc, cid2) in fields
                                  do newf := newf ⌒ [mk-AS‘Field (sel,
                                                                  SubstType (ftp,
                                                                             tv-l,
                                                                             tp-l),
                                                                  dc, cid2)];
                                  return mk-AS‘CompositeType (id, newf, cid) ),
    (is-AS‘UnionType (tp)) → let mk-AS‘UnionType (utp-l, cid) = tp in
                             return mk-AS‘UnionType ([SubstType (utp-l (i),
                                                                 tv-l,
                                                                 tp-l) |
                                                      i ∈ inds utp-l],
                                                     cid),
    (is-AS‘ProductType (tp)) → let mk-AS‘ProductType (ptp-l, cid) = tp in
                               return mk-AS‘ProductType ([SubstType (ptp-l (i),
                                                                     tv-l,
                                                                     tp-l) |
                                                          i ∈ inds ptp-l],
                                                         cid),
    (is-AS‘OptionalType (tp)) → let mk-AS‘OptionalType (otp, cid) = tp in
                                return mk-AS‘OptionalType (SubstType (otp, tv-l, tp-l), cid),
    (is-AS‘SetType (tp)) → let mk-AS‘SetType (stp, cid) = tp in
                           return mk-AS‘SetType (SubstType (stp, tv-l, tp-l), cid),
```

```
(is-AS`Seq0Type (tp)) → let mk-AS`Seq0Type (stp, cid) = tp in
                        return mk-AS`Seq0Type (SubstType (stp, tv-l, tp-l), cid),
(is-AS`Seq1Type (tp)) → let mk-AS`Seq1Type (stp, cid) = tp in
                        return mk-AS`Seq1Type (SubstType (stp, tv-l, tp-l), cid),
(is-AS`GeneralMapType (tp)) → let mk-AS`GeneralMapType (dtp, rtp, cid) = tp in
                              return mk-AS`GeneralMapType (SubstType (dtp,
                                                                      tv-l,
                                                                      tp-l),
                                                           SubstType (rtp,
                                                                      tv-l,
                                                                      tp-l),
                                                           cid),
(is-AS`InjectiveMapType (tp)) → let mk-AS`InjectiveMapType (dtp, rtp, cid) = tp in
                                return mk-AS`InjectiveMapType (SubstType (dtp,
                                                                          tv-l,
                                                                          tp-l),
                                                               SubstType (rtp,
                                                                          tv-l,
                                                                          tp-l),
                                                               cid),
(is-AS`PartialFnType (tp)) → let mk-AS`PartialFnType (dtp, rtp, cid) = tp in
                             return mk-AS`PartialFnType ([SubstType (dtp (i),
                                                                     tv-l,
                                                                     tp-l) |
                                                          i ∈ inds dtp],
                                                         SubstType (rtp,
                                                                    tv-l,
                                                                    tp-l),
                                                         cid),
(is-AS`TotalFnType (tp)) → let mk-AS`TotalFnType (dtp, rtp, cid) = tp in
                           return mk-AS`TotalFnType ([SubstType (dtp (i),
                                                                 tv-l,
                                                                 tp-l) |
                                                      i ∈ inds dtp],
                                                     SubstType (rtp,
                                                                tv-l,
                                                                tp-l),
                                                     cid),
(is-AS`TypeVar (tp)) → let mk-AS`TypeVar (id,-) = tp in
                       (dcl ttv-l : AS`TypeVarList := tv-l,
                            ttp-l : AS`Type* := tp-l;
                        while ttv-l ≠ []
                        do (let mk-AS`TypeVar (tv-id,-) = hd ttv-l in
                              if id = tv-id
                              then return hd ttp-l
                              else skip;
                              ttv-l := tl ttv-l;
                              ttp-l := tl ttp-l);
                        error)
end;
```

This operation performs the actual substitution of the type variable identifiers with normal types.

1.19.3 Lambda Expression

Deleted because it is no longer necessary. (PGL)

1.19.4 Is Expression

This operation returns the semantic value true, if the evaluated input expression is of the same type as the input type.If the value is a record value, the structure of the record type must be available.

1.19.5 Names

Hej implementoer (Jesper/Hanne).

Funktionerne IsFunction IsPoly og IsOp er flyttet til LookOpFctPoly operationen som nu befinder sig i STATE modulet..

Hej implementoer (Jesper/Hanne): The operations IsInObjScope and IsValue are moved to module state!!!!!

1.19.6 Unary and Binary Expressions

```
1141.0  EvalMapInverseExpr : SEM`VAL -o-> SEM`VAL
    .1  EvalMapInverseExpr (val-v) △
    .2    if is-SEM`MAP (val-v)
    .3    then let mk-SEM`MAP (map-v) = val-v in
    .4           if card dom map-v = card rng map-v
    .5           then return mk-SEM`MAP ({map-v (d-v) ↦ d-v | d-v ∈ dom map-v})
    .6           else error
    .7    else error;
```

This operation returns the inverse of the input map value, i.e. for all maplets in the map, domain and range are switched. To be able to do this, the map must be injective, i.e. the cardinality of the domain and range sets must be equal.

1142.0 $EvalNumUnaryExpr : AS`UnaryOp \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalNumUnaryExpr\,(opr, op\text{-}v) \triangleq$
.2 if is-$SEM`NUM\,(op\text{-}v)$
.3 then cases opr:
.4 NUMPLUS,
.5 NUMMINUS,
.6 NUMABS $\rightarrow EvalPlusMinusAbs(opr, op\text{-}v)$,
.7 FLOOR $\rightarrow EvalFloor(op\text{-}v)$
.8 end
.9 else error;

1143.0 $EvalPlusMinusAbs : AS`UnaryOp \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalPlusMinusAbs\,(opr,$ mk-$SEM`NUM\,(tmp)) \triangleq$
.2 cases opr:
.3 NUMPLUS $\rightarrow$ return mk-$SEM`NUM\,(+\,tmp)$,
.4 NUMMINUS $\rightarrow$ return mk-$SEM`NUM\,(-\,tmp)$,
.5 NUMABS $\rightarrow$ if $tmp < 0$
.6 then return mk-$SEM`NUM\,(-\,tmp)$
.7 else return mk-$SEM`NUM\,(tmp)$
.8 end;

1144.0 $EvalFloor : SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalFloor\,($mk-$SEM`NUM\,(n)) \triangleq$
.2 return mk-$SEM`NUM$ (floor n);

1145.0 $EvalNumBinaryExpr : SEM`VAL \times AS`BinaryOp \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalNumBinaryExpr\,(op1\text{-}v, opr, op2\text{-}v) \triangleq$
.2 if is-$SEM`NUM\,(op1\text{-}v)$
.3 then if is-$SEM`NUM\,(op2\text{-}v)$
.4 then cases opr:
.5 NUMPLUS,
.6 NUMMINUS,
.7 NUMMULT,
.8 NUMDIV,
.9 NUMGE,
.10 NUMGT,
.11 NUMLT,
.12 NUMLE $\rightarrow EvalNumBinOp(op1\text{-}v, opr, op2\text{-}v)$,
.13 INTDIV,
.14 NUMREM $\rightarrow$ if $IsInt\,(op1\text{-}v) \wedge IsInt\,(op2\text{-}v)$
.15 then cases opr:
.16 INTDIV $\rightarrow EvalIntDiv(op1\text{-}v, op2\text{-}v)$,
.17 NUMREM $\rightarrow EvalNumRem(op1\text{-}v, op2\text{-}v)$
.18 end

.19 else error,
.20 NUMMOD $\rightarrow$ if $IsInt\,(op1\text{-}v) \wedge IsInt\,(op2\text{-}v)$
.21 then $EvalNumMod(op1\text{-}v, op2\text{-}v)$
.22 else error
.23 end
.24 else $RTERR`ErrorVal(RTERR`NUM\text{-}EXPECTED, op2\text{-}v,$ nil , $[])$
.25 else $RTERR`ErrorVal(RTERR`NUM\text{-}EXPECTED, op1\text{-}v,$ nil , $[])$;

1146.0 $EvalNumBinOp : SEM`VAL \times AS`BinaryOp \times SEM`VAL \xrightarrow{o} SEM`VAL$
.1 $EvalNumBinOp\,($mk-$SEM`NUM\,(n1), opr,$ mk-$SEM`NUM\,(n2)) \triangleq$
.2 cases opr:
.3 NUMPLUS $\rightarrow$ return mk-$SEM`NUM\,(n1 + n2)$,
.4 NUMMINUS $\rightarrow$ return mk-$SEM`NUM\,(n1 - n2)$,
.5 NUMMULT $\rightarrow$ return mk-$SEM`NUM\,(n1 \times n2)$,
.6 NUMDIV $\rightarrow$ (if $n2 = 0$
.7 then $RTERR`ErrorVal(RTERR`DIVISION\text{-}WITH\text{-}ZERO,$ nil , nil , $[])$;
.8 return mk-$SEM`NUM\,(n1/n2)$),
.9 NUMEXP $\rightarrow$ if is-$\mathbb{R}\,(n2) \wedge$
.10 $((n1 > 0) \vee (n1 = 0 \wedge n2 > 0) \vee (n1 < 0 \wedge$ is-$\mathbb{Z}\,(n2)))$
.11 then return mk-$SEM`NUM\,(n1 \uparrow n2)$
.12 else error,
.13 NUMGE $\rightarrow$ return mk-$SEM`BOOL\,(n1 \geq n2)$,
.14 NUMGT $\rightarrow$ return mk-$SEM`BOOL\,(n1 > n2)$,
.15 NUMLE $\rightarrow$ return mk-$SEM`BOOL\,(n1 \leq n2)$,
.16 NUMLT $\rightarrow$ return mk-$SEM`BOOL\,(n1 < n2)$,
.17 others $\rightarrow$ error
.18 end;

1147.0 $EvalIntDiv : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$
.1 $EvalIntDiv\,($mk-$SEM`NUM\,(n1),$ mk-$SEM`NUM\,(n2)) \triangleq$
.2 if is-$\mathbb{Z}\,(n1) \wedge$ is-$\mathbb{Z}\,(n2)$
.3 then if $n2 \neq 0$
.4 then return mk-$SEM`NUM\,(n1$ div $n2)$
.5 else error
.6 else error;

1148.0 $EvalNumRem : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$
.1 $EvalNumRem\,($mk-$SEM`NUM\,(n1),$ mk-$SEM`NUM\,(n2)) \triangleq$
.2 if is-$\mathbb{Z}\,(n1) \wedge$ is-$\mathbb{Z}\,(n2)$
.3 then if $n2 \neq 0$
.4 then return mk-$SEM`NUM\,(n1$ rem $n2)$
.5 else error
.6 else error;

1149.0 $EvalNumMod : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalNumMod\,(\mathsf{mk}\text{-}SEM`NUM\,(n1), \mathsf{mk}\text{-}SEM`NUM\,(n2)) \triangleq$

.2   if is-$\mathbb{Z}\,(n1) \wedge$ is-$\mathbb{Z}\,(n2)$

.3   then if $n2 \neq 0$

.4   then return mk-$SEM`NUM\,(n1 - n2 \times \mathsf{floor}\ (n1/n2))$

.5   else error

.6   else error;

1150.0 $EvalLogUnaryExpr : AS`UnaryOp \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalLogUnaryExpr\,(opr, op\text{-}v) \triangleq$

.2   if is-$SEM`BOOL\,(op\text{-}v)$

.3   then let mk-$SEM`BOOL\,(b) = op\text{-}v$ in

.4   cases opr:

.5 $\mathrm{NOT} \rightarrow$ return mk-$SEM`BOOL\,(\neg\, b)$

.6   end

.7   else error;

1151.0 $EvalEqNeBinaryExpr : SEM`VAL \times AS`BinaryOp \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalEqNeBinaryExpr\,(op1\text{-}v, op, op2\text{-}v) \triangleq$

.2   if $\mathrm{NE} = op$

.3   then return mk-$SEM`BOOL\,(op1\text{-}v \neq op2\text{-}v)$

.4   elseif $\mathrm{EQ} = op$

.5   then return mk-$SEM`BOOL\,(op1\text{-}v = op2\text{-}v)$

.6   elseif is-$SEM`BOOL\,(op1\text{-}v) \wedge$ is-$SEM`BOOL\,(op2\text{-}v)$

.7   then return mk-$SEM`BOOL\,(op1\text{-}v.v \;\Leftrightarrow\; op2\text{-}v.v)$

.8   elseif $\neg$ is-$SEM`BOOL\,(op1\text{-}v)$

.9   then $RTERR`ErrorVal(RTERR`TWO\text{-}BOOL\text{-}EXPECTED, op1\text{-}v, \mathsf{nil}\ , [])$

.10   else $RTERR`ErrorVal(RTERR`TWO\text{-}BOOL\text{-}EXPECTED, op2\text{-}v, \mathsf{nil}\ , [])$

.11 pre $op \in \{\mathrm{NE}, \mathrm{EQ}, \mathrm{EQUIV}\}$;

1152.0 $EvalSetUnaryExpr : AS`UnaryOp \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalSetUnaryExpr\,(opr, op\text{-}v) \triangleq$

.2   if is-$SEM`SET\,(op\text{-}v)$

.3   then let mk-$SEM`SET\,(op\text{-}sv) = op\text{-}v$ in

.4   cases opr:

.5 $\mathrm{SETCARD} \rightarrow$ return mk-$SEM`NUM\,(\mathsf{card}\ op\text{-}sv)$,

.6 $\mathrm{SETPOWER} \rightarrow$ if card $op\text{-}sv \geq 16$

.7   then $RTERR`ErrorVal(RTERR`SET\text{-}TOO\text{-}BIG, op\text{-}v, \mathsf{nil}\ , [])$

.8   else let $ps\text{-}sv = \{\mathsf{mk}\text{-}SEM`SET\ (sub) \mid$

.9 $sub \in \mathcal{F}\ op\text{-}sv\}$ in

.10   return mk-$SEM`SET\,(ps\text{-}sv)$,

```
      SETDISTRUNION → if ∀ elm-v ∈ op-sv · is-SEM‘SET (elm-v)
                      then let tmp-ssv = {let mk-SEM‘SET (elm-sv) =
                                                  elm-v in
                                              elm-sv |
                                                  elm-v ∈ op-sv} in
                           return mk-SEM‘SET (⋃ tmp-ssv)
                      else error,
      SETDISTRINTERSECT → if card op-sv > 0 ∧
                               ∀ elm-v ∈ op-sv · is-SEM‘SET (elm-v)
                          then let tmp-ssv = {let mk-SEM‘SET (elm-sv) =

                                                      elm-v in
                                                  elm-sv |
                                                      elm-v ∈ op-sv} in
                               return mk-SEM‘SET (⋂ tmp-ssv)
                          else error,
      others → error
    end
  else RTERR‘ErrorVal(RTERR‘SET-EXPECTED, op-v, nil , []) ;
```

1153.0 $EvalSetBinaryExpr : SEM‘VAL \times AS‘BinaryOp \times SEM‘VAL \xrightarrow{o} SEM‘VAL$

```
EvalSetBinaryExpr (op1-v, opr, op2-v) △
  cases opr:
    NOTINSET → if is-SEM‘SET (op2-v)
               then EvalNotInSet(op1-v, op2-v)
               else error,
    INSET → if is-SEM‘SET (op2-v)
            then EvalInSet(op1-v, op2-v)
            else error,
    SETUNION → if is-SEM‘SET (op1-v) ∧ is-SEM‘SET (op2-v)
               then EvalSetUnion(op1-v, op2-v)
               else error,
    SETINTERSECT → if is-SEM‘SET (op1-v) ∧ is-SEM‘SET (op2-v)
                   then EvalSetIntersect(op1-v, op2-v)
                   else error,
    SETMINUS → if is-SEM‘SET (op1-v) ∧ is-SEM‘SET (op2-v)
               then EvalSetMinus(op1-v, op2-v)
               else error,
    PROPERSUBSET → if is-SEM‘SET (op1-v) ∧ is-SEM‘SET (op2-v)
                   then EvalProperSubSet(op1-v, op2-v)
                   else error,
    SUBSET → if is-SEM‘SET (op1-v) ∧ is-SEM‘SET (op2-v)
             then EvalSubSet(op1-v, op2-v)
             else error,
    others → error
  end;
```

1154.0 $EvalInSet : SEM\text{‘}VAL \times SEM\text{‘}VAL \xrightarrow{o} SEM\text{‘}VAL$
.1 $EvalInSet\,(op1\text{-}v, \mathsf{mk}\text{-}SEM\text{‘}SET\,(op2\text{-}sv)) \triangleq$
.2 $\quad \mathsf{let}\ b = (op1\text{-}v \in op2\text{-}sv)\ \mathsf{in}$
.3 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM\text{‘}BOOL\,(b);$

1155.0 $EvalNotInSet : SEM\text{‘}VAL \times SEM\text{‘}VAL \xrightarrow{o} SEM\text{‘}VAL$
.1 $EvalNotInSet\,(op1\text{-}v, \mathsf{mk}\text{-}SEM\text{‘}SET\,(op2\text{-}sv)) \triangleq$
.2 $\quad \mathsf{let}\ b = (op1\text{-}v \notin op2\text{-}sv)\ \mathsf{in}$
.3 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM\text{‘}BOOL\,(b);$

1156.0 $EvalSetUnion : SEM\text{‘}VAL \times SEM\text{‘}VAL \xrightarrow{o} SEM\text{‘}VAL$
.1 $EvalSetUnion\,(\mathsf{mk}\text{-}SEM\text{‘}SET\,(op1\text{-}sv), \mathsf{mk}\text{-}SEM\text{‘}SET\,(op2\text{-}sv)) \triangleq$
.2 $\quad \mathsf{let}\ set\text{-}sv = op1\text{-}sv \cup op2\text{-}sv\ \mathsf{in}$
.3 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM\text{‘}SET\,(set\text{-}sv);$

1157.0 $EvalSetIntersect : SEM\text{‘}VAL \times SEM\text{‘}VAL \xrightarrow{o} SEM\text{‘}VAL$
.1 $EvalSetIntersect\,(\mathsf{mk}\text{-}SEM\text{‘}SET\,(op1\text{-}sv), \mathsf{mk}\text{-}SEM\text{‘}SET\,(op2\text{-}sv)) \triangleq$
.2 $\quad \mathsf{let}\ set\text{-}sv = op1\text{-}sv \cap op2\text{-}sv\ \mathsf{in}$
.3 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM\text{‘}SET\,(set\text{-}sv);$

1158.0 $EvalSetMinus : SEM\text{‘}VAL \times SEM\text{‘}VAL \xrightarrow{o} SEM\text{‘}VAL$
.1 $EvalSetMinus\,(\mathsf{mk}\text{-}SEM\text{‘}SET\,(op1\text{-}sv), \mathsf{mk}\text{-}SEM\text{‘}SET\,(op2\text{-}sv)) \triangleq$
.2 $\quad \mathsf{let}\ set\text{-}sv = op1\text{-}sv \setminus op2\text{-}sv\ \mathsf{in}$
.3 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM\text{‘}SET\,(set\text{-}sv);$

1159.0 $EvalSubSet : SEM\text{‘}VAL \times SEM\text{‘}VAL \xrightarrow{o} SEM\text{‘}VAL$
.1 $EvalSubSet\,(\mathsf{mk}\text{-}SEM\text{‘}SET\,(op1\text{-}sv), \mathsf{mk}\text{-}SEM\text{‘}SET\,(op2\text{-}sv)) \triangleq$
.2 $\quad \mathsf{let}\ b = op1\text{-}sv \subseteq op2\text{-}sv\ \mathsf{in}$
.3 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM\text{‘}BOOL\,(b);$

1160.0 $EvalProperSubSet : SEM\text{‘}VAL \times SEM\text{‘}VAL \xrightarrow{o} SEM\text{‘}VAL$
.1 $EvalProperSubSet\,(\mathsf{mk}\text{-}SEM\text{‘}SET\,(op1\text{-}sv), \mathsf{mk}\text{-}SEM\text{‘}SET\,(op2\text{-}sv)) \triangleq$
.2 $\quad \mathsf{let}\ b = op1\text{-}sv \subset op2\text{-}sv\ \mathsf{in}$
.3 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM\text{‘}BOOL\,(b);$

```
1161.0 EvalSeqUnaryExpr : AS‘UnaryOp × SEM‘VAL ⥲ SEM‘VAL
    .1 EvalSeqUnaryExpr (opr, op-v) ≜
    .2   if is-SEM‘SEQ (op-v)
    .3   then let mk-SEM‘SEQ (op-lv) = op-v in
    .4          cases opr:
    .5            SEQHEAD → if op-lv ≠ []
    .6                        then return hd op-lv
    .7                        else error,
    .8            SEQTAIL → if op-lv ≠ []
    .9                       then return mk-SEM‘SEQ (tl op-lv)
   .10                       else error,
   .11            SEQLEN → return mk-SEM‘NUM (len op-lv),
   .12            SEQELEMS → return mk-SEM‘SET (elems op-lv),
   .13            SEQINDICES → return mk-SEM‘SET ({mk-SEM‘NUM (i) | i ∈ inds op-lv}),
   .14            SEQDISTRCONC → if ∀ elm-v ∈ elems op-lv · is-SEM‘SEQ (elm-v)
   .15                            then let seq-llv = [let mk-SEM‘SEQ (elm-lv) = op-lv (i) in
   .16                                                   elm-lv |
   .17                                                     i ∈ inds op-lv] in
   .18                                   return mk-SEM‘SEQ (conc seq-llv)
   .19                            else error,
   .20            others → error
   .21          end
   .22   else error;


1162.0 EvalSeqBinaryExpr : SEM‘VAL × AS‘BinaryOp × SEM‘VAL ⥲ SEM‘VAL
    .1 EvalSeqBinaryExpr (op1-v, opr, op2-v) ≜
    .2   cases opr:
    .3     SEQCONC → if is-SEM‘SEQ (op1-v) ∧ is-SEM‘SEQ (op2-v)
    .4               then let mk-SEM‘SEQ (op1-lv) = op1-v,
    .5                        mk-SEM‘SEQ (op2-lv) = op2-v in
    .6                      return mk-SEM‘SEQ (op1-lv ⁀ op2-lv)
    .7               else error,
    .8     others → error
    .9   end;


1163.0 EvalMapUnaryExpr : AS‘UnaryOp × SEM‘VAL ⥲ SEM‘VAL
    .1 EvalMapUnaryExpr (opr, op-v) ≜
    .2   cases opr:
    .3     MAPDOM → if is-SEM‘MAP (op-v)
    .4              then let mk-SEM‘MAP (map-v) = op-v in
    .5                     return mk-SEM‘SET (dom map-v)
    .6              else error,
    .7     MAPRNG → if is-SEM‘MAP (op-v)
    .8              then let mk-SEM‘MAP (map-v) = op-v in
    .9                     return mk-SEM‘SET (rng map-v)
   .10              else error,
```

.11 MAPDISTRMERGE $\rightarrow$ if is-$SEM`SET$ $(op\text{-}v)$
.12 then let mk-$SEM`SET$ $(op\text{-}sv) = op\text{-}v$ in
.13 if $\forall$ $elm\text{-}v \in op\text{-}sv \cdot$ is-$SEM`MAP$ $(elm\text{-}v)$
.14 then let $mapsops = \{$let mk-$SEM`MAP$ $(elm) = elm\text{-}v$ in
.15 elm |
.16 $elm\text{-}v \in op\text{-}sv\}$ in
.17 if $\forall$ $m1, m2 \in mapsops \cdot$
.18 $\forall$ $d \in$ (dom $m1 \cap$ dom $m2) \cdot$
.19 $m1\,(d) = m2\,(d)$
.20 then return mk-$SEM`MAP$ (merge $mapsops$)
.21 else error
.22 else error
.23 else error
.24 end;

1164.0 $EvalMapBinaryExpr : SEM`VAL \times AS`BinaryOp \times SEM`VAL \xrightarrow{o} SEM`VAL$
.1 $EvalMapBinaryExpr\,(op1\text{-}v, opr, op2\text{-}v) \triangleq$
.2 cases opr:
.3 MAPMERGE $\rightarrow$ if is-$SEM`MAP$ $(op1\text{-}v) \wedge$ is-$SEM`MAP$ $(op2\text{-}v)$
.4 then $EvalMapMerge(op1\text{-}v, op2\text{-}v)$
.5 else error,
.6 MAPDOMRESTTO,
.7 MAPDOMRESTBY $\rightarrow$ if is-$SEM`SET$ $(op1\text{-}v) \wedge$ is-$SEM`MAP$ $(op2\text{-}v)$
.8 then cases opr:
.9 MAPDOMRESTTO $\rightarrow$ $EvalMapDomRestTo(op1\text{-}v, op2\text{-}v)$,
.10 MAPDOMRESTBY $\rightarrow$ $EvalMapDomRestBy(op1\text{-}v, op2\text{-}v)$
.11 end
.12 else error,
.13 MAPRNGRESTTO,
.14 MAPRNGRESTBY $\rightarrow$ if is-$SEM`MAP$ $(op1\text{-}v) \wedge$ is-$SEM`SET$ $(op2\text{-}v)$
.15 then cases opr:
.16 MAPRNGRESTTO $\rightarrow$ $EvalMapRngRestTo(op1\text{-}v, op2\text{-}v)$,
.17 MAPRNGRESTBY $\rightarrow$ $EvalMapRngRestBy(op1\text{-}v, op2\text{-}v)$
.18 end
.19 else error,
.20 others $\rightarrow$ error
.21 end;

1165.0 $EvalMapMerge : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$
.1 $EvalMapMerge$ (mk-$SEM`MAP$ $(op1)$, mk-$SEM`MAP$ $(op2)) \triangleq$
.2 if $\forall$ $d \in$ (dom $op1 \cap$ dom $op2) \cdot op1\,(d) = op2\,(d)$
.3 then return mk-$SEM`MAP$ $(op1 \overset{m}{\cup} op2)$
.4 else error;

1166.0 $EvalMapDomRestTo : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalMapDomRestTo\ (\mathsf{mk\text{-}}SEM`SET\ (op1\text{-}sv), \mathsf{mk\text{-}}SEM`MAP\ (op2)) \triangleq$

.2 $\quad \mathsf{return\ mk\text{-}}SEM`MAP\ (op1\text{-}sv \lhd op2);$

1167.0 $EvalMapDomRestBy : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalMapDomRestBy\ (\mathsf{mk\text{-}}SEM`SET\ (op1\text{-}sv), \mathsf{mk\text{-}}SEM`MAP\ (op2)) \triangleq$

.2 $\quad \mathsf{return\ mk\text{-}}SEM`MAP\ (op1\text{-}sv \mathbin{-\!\!\!\lhd} op2);$

1168.0 $EvalMapRngRestTo : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalMapRngRestTo\ (\mathsf{mk\text{-}}SEM`MAP\ (op1), \mathsf{mk\text{-}}SEM`SET\ (op2\text{-}sv)) \triangleq$

.2 $\quad \mathsf{return\ mk\text{-}}SEM`MAP\ (op1 \rhd op2\text{-}sv);$

1169.0 $EvalMapRngRestBy : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalMapRngRestBy\ (\mathsf{mk\text{-}}SEM`MAP\ (op1), \mathsf{mk\text{-}}SEM`SET\ (op2\text{-}sv)) \triangleq$

.2 $\quad \mathsf{return\ mk\text{-}}SEM`MAP\ (op1 \mathbin{\rhd\!\!\!-} op2\text{-}sv);$

1170.0 $EvalComposeExpr : SEM`VAL \times SEM`VAL \xrightarrow{o} SEM`VAL$

.1 $EvalComposeExpr\ (op1\text{-}v, op2\text{-}v) \triangleq$

.2 $\quad \mathsf{if\ is\text{-}}SEM`CompExplFN\ (op1\text{-}v) \wedge \mathsf{is\text{-}}SEM`CompExplFN\ (op2\text{-}v)$

.3 $\quad \mathsf{then}\ EvalComposeFctExpr(op1\text{-}v, op2\text{-}v)$

.4 $\quad \mathsf{elseif\ is\text{-}}SEM`ImplFN\ (op1\text{-}v) \vee \mathsf{is\text{-}}SEM`ImplFN\ (op2\text{-}v)$

.5 $\quad \mathsf{then\ return\ mk\text{-}}SEM`ImplFN\ ()$

.6 $\quad \mathsf{elseif\ is\text{-}}SEM`MAP\ (op1\text{-}v) \wedge \mathsf{is\text{-}}SEM`MAP\ (op2\text{-}v)$

.7 $\quad \mathsf{then\ let\ mk\text{-}}SEM`MAP\ (map1\text{-}v) = op1\text{-}v,$

.8 $\qquad\qquad \mathsf{mk\text{-}}SEM`MAP\ (map2\text{-}v) = op2\text{-}v\ \mathsf{in}$

.9 $\qquad \mathsf{if\ rng}\ map2\text{-}v \subseteq \mathsf{dom}\ map1\text{-}v$

.10 $\qquad \mathsf{then\ return\ mk\text{-}}SEM`MAP\ (\{i \mapsto map1\text{-}v\ (map2\text{-}v\ (i)) \mid i \in \mathsf{dom}\ map2\text{-}v\})$

.11 $\qquad \mathsf{else\ error}$

.12 $\quad \mathsf{else\ error};$

If both of the evaluated input expressions are explicit semantic function values, the operation *EvalComposeFctExpr* is called. If either of the result values is an implicit semantic function value, an implicit semantic function value is returned. Otherwise, both values are map values, and the map composition is created, if the range of the seconds map value is a subset of the domain of the first set.

1171.0 $EvalComposeFctExpr : SEM`CompExplFN \times SEM`CompExplFN \xrightarrow{o} SEM`VAL$

.1 $EvalComposeFctExpr\ (\mathsf{mk\text{-}}SEM`CompExplFN\ (fvl1, \mathsf{nil}\ ), \mathsf{mk\text{-}}SEM`CompExplFN\ (fvl2, \mathsf{nil}\ )) \triangleq$

.2 $\quad \mathsf{return\ mk\text{-}}SEM`CompExplFN\ (fvl2 \frown fvl1, \mathsf{nil}\ );$

This operation returns the semantic function value for the function composition of two explicit functions. The domain type of the composit function is the domain type of the second function, and the range type is the range type of the first function. The body of the function is the following let-expression:

$$\mathsf{let}\ x = g(y)\ \mathsf{in}\ f(x)$$

where f represents the first function, and g the second function.

1172.0 $\mathit{EvalIterateExpr} : \mathit{SEM`VAL} \times \mathit{SEM`VAL} \xrightarrow{o} \mathit{SEM`VAL}$

$\mathit{EvalIterateExpr}\ (op1\text{-}v, op2\text{-}v) \triangleq$
$\quad \mathsf{if}\ \mathsf{is}\text{-}\mathit{SEM`CompExplFN}\ (op1\text{-}v) \wedge \mathit{IsNat}\ (op2\text{-}v)$
$\quad \mathsf{then}\ \mathit{IterateFct}(op1\text{-}v, op2\text{-}v)$
$\quad \mathsf{elseif}\ \mathsf{is}\text{-}\mathit{SEM`ImplFN}\ (op1\text{-}v) \wedge \mathit{IsNat}\ (op2\text{-}v)$
$\quad \mathsf{then\ return\ mk}\text{-}\mathit{SEM`ImplFN}\ ()$
$\quad \mathsf{elseif}\ \mathsf{is}\text{-}\mathit{SEM`MAP}\ (op1\text{-}v) \wedge \mathit{IsNat}\ (op2\text{-}v)$
$\quad \mathsf{then\ let\ mk}\text{-}\mathit{SEM`NUM}\ (n) = op2\text{-}v\ \mathsf{in}$
$\qquad\quad \mathit{IterateMap}(op1\text{-}v, n)$
$\quad \mathsf{elseif}\ \mathsf{is}\text{-}\mathit{SEM`NUM}\ (op1\text{-}v) \wedge \mathsf{is}\text{-}\mathit{SEM`NUM}\ (op2\text{-}v)$
$\quad \mathsf{then}\ \mathit{EvalNumBinOp}(op1\text{-}v, \mathrm{NUMEXP}, op2\text{-}v)$
$\quad \mathsf{else\ error;}$

This operation returns the iteration of an expression with a number. Depending on the result of the evaluation of the expression, we call the correct operation.

1173.0 $\mathit{IterateFct} : \mathit{SEM`CompExplFN} \times \mathit{SEM`VAL} \xrightarrow{o} \mathit{SEM`VAL}$

$\mathit{IterateFct}\ (fn\text{-}v, num) \triangleq$
$\quad \mathsf{let\ mk}\text{-}\mathit{SEM`NUM}\ (n) = num\ \mathsf{in}$
$\quad \mathsf{if}\ n = 1$
$\quad \mathsf{then\ return}\ fn\text{-}v$
$\quad \mathsf{elseif}\ n > 1$
$\quad \mathsf{then\ return\ mk}\text{-}\mathit{SEM`CompExplFN}\ (\mathsf{conc}\ [fn\text{-}v.fl \mid i \in \{1, \ldots, n\}], \mathsf{nil}\ )$
$\quad \mathsf{elseif}\ n = 0$
$\quad \mathsf{then\ return\ mk}\text{-}\mathit{SEM`CompExplFN}\ ([\mu\ (fn\text{-}v.fl\ (i), instr \mapsto \mathsf{nil}\ ) \mid$
$\qquad\qquad i \in \mathsf{inds}\ fn\text{-}v.fl],$
$\qquad\quad \mathsf{nil}\ )$
$\quad \mathsf{else\ error;}$

For function iteration, we return the identity function in case the number of iterations is 0. The identity function returns the input as result of the function. If the number of iterations is 1, we return the the input function as result. Otherwise, we use repeated function composition with the input function to create the proper function.

1174.0 $IterateMap : SEM`VAL \times \mathbb{N} \xrightarrow{o} SEM`VAL$

.1 $IterateMap\,(\mathsf{mk}\text{-}SEM`MAP\,(map\text{-}v), n) \triangleq$

.2 $\quad$ if $n \geq 1 \wedge (n \geq 2 \;\Rightarrow\; (\mathsf{rng}\ map\text{-}v \subseteq \mathsf{dom}\ map\text{-}v))$

.3 $\quad$ then let $\mathsf{mk}\text{-}SEM`MAP\,(tmp\text{-}v) = IterateMap\,(\mathsf{mk}\text{-}SEM`MAP\,(map\text{-}v), n-1)$ in

.4 $\quad\quad$ return $\mathsf{mk}\text{-}SEM`MAP\,(\{i \mapsto map\text{-}v\,(tmp\text{-}v\,(i)) \mid i \in \mathsf{dom}\ map\text{-}v\})$

.5 $\quad$ elseif $n = 0$

.6 $\quad$ then return $\mathsf{mk}\text{-}SEM`MAP\,(\{i \mapsto i \mid i \in \mathsf{dom}\ map\text{-}v\})$

.7 $\quad$ else error

end $EXPR$

If the number of iterations is equal to 1, we return the input map as result. In case the number of iterations is equal to 0, the identity map is returned, i.e. the domain of the map is equal to the domain of the input map, and each element from the domain maps to itself. Otherwise, map iteration is equal to repeated map composition with the input map.

Test Suite : rtinfo.ast
Module : EXPR

Name	#Calls	Coverage
EXPR`EvalFloor	**undefined**	**undefined**
EXPR`EvalInSet	**undefined**	**undefined**
EXPR`SubstType	**undefined**	**undefined**
EXPR`EvalIntDiv	**undefined**	**undefined**
EXPR`EvalNumMod	**undefined**	**undefined**
EXPR`EvalNumRem	**undefined**	**undefined**
EXPR`EvalSubSet	**undefined**	**undefined**
EXPR`IterateFct	**undefined**	**undefined**
EXPR`IterateMap	**undefined**	**undefined**
EXPR`EvalMapApply	**undefined**	**undefined**
EXPR`EvalMapMerge	**undefined**	**undefined**
EXPR`EvalNotInSet	**undefined**	**undefined**
EXPR`EvalNumBinOp	**undefined**	**undefined**
EXPR`EvalSelfExpr	**undefined**	**undefined**
EXPR`EvalSeqApply	**undefined**	**undefined**
EXPR`EvalSetMinus	**undefined**	**undefined**
EXPR`EvalSetUnion	**undefined**	**undefined**
EXPR`ConvertPolyToFn	**undefined**	**undefined**
EXPR`EvalComposeExpr	**undefined**	**undefined**
EXPR`EvalIterateExpr	**undefined**	**undefined**
EXPR`EvalLogUnaryExpr	**undefined**	**undefined**
EXPR`EvalMapDomRestBy	**undefined**	**undefined**
EXPR`EvalMapDomRestTo	**undefined**	**undefined**
EXPR`EvalMapRngRestBy	**undefined**	**undefined**
EXPR`EvalMapRngRestTo	**undefined**	**undefined**
EXPR`EvalMapUnaryExpr	**undefined**	**undefined**
EXPR`EvalNumUnaryExpr	**undefined**	**undefined**
EXPR`EvalPlusMinusAbs	**undefined**	**undefined**
EXPR`EvalProperSubSet	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
EXPR‘EvalSeqUnaryExpr	**undefined**	**undefined**
EXPR‘EvalSetIntersect	**undefined**	**undefined**
EXPR‘EvalSetRangeExpr	**undefined**	**undefined**
EXPR‘EvalSetUnaryExpr	**undefined**	**undefined**
EXPR‘EvalMapBinaryExpr	**undefined**	**undefined**
EXPR‘EvalNumBinaryExpr	**undefined**	**undefined**
EXPR‘EvalSeqBinaryExpr	**undefined**	**undefined**
EXPR‘EvalSeqModifyExpr	**undefined**	**undefined**
EXPR‘EvalSetBinaryExpr	**undefined**	**undefined**
EXPR‘EvalComposeFctExpr	**undefined**	**undefined**
EXPR‘EvalEqNeBinaryExpr	**undefined**	**undefined**
EXPR‘EvalMapInverseExpr	**undefined**	**undefined**
EXPR‘EvalFieldSelectExpr	**undefined**	**undefined**
EXPR‘EvalMapOverrideExpr	**undefined**	**undefined**
EXPR‘EvalSubSequenceExpr	**undefined**	**undefined**
EXPR‘ConstructSEMRecFields	**undefined**	**undefined**
EXPR‘EvalFieldOBJRefSelect	**undefined**	**undefined**
EXPR‘EvalFieldRecordSelect	**undefined**	**undefined**
EXPR‘EvalRecordModifierExpr	**undefined**	**undefined**
EXPR‘EvalRecordConstructorExpr	**undefined**	**undefined**
EXPR‘EvalSeqModifyMapOverrideExpr	**undefined**	**undefined**
Total Coverage		**0%**

1.20 Pattern and Binds

The module PAT contains all definition related to pattern and binds.
module PAT

imports

1175.0 from AS all ,

1176.0 from CI all ,

1177.0 from AUX

1178.0 functions $Permute : SEM`VAL^* \rightarrow SEM`VAL^*$-set renamed $Permute$

1179.0 operations $ErrorOp : \mathsf{char}^* \xrightarrow{o}$
.1 $\mathbb{B} \mid SEM`VAL \mid AS`Name\text{-}\mathsf{set} \mid$
.2 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL]) \mid AS`Name \mid$
.3 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL] \times [SEM`VAL] \times [AS`Name] \times [AS`Access]) \mid$
.4 $SEM`BlkEnv\text{-}\mathsf{set} \mid (\mathbb{B} \times [GLOBAL`Type] \times [AS`Invariant] \times [AS`Name]) \mid$

.5 $GLOBAL`RecSel$ renamed $ErrorOp$;
.6 $MkBlkEnv : AS`Name \times SEM`VAL \times [AS`Type] \times SEM`Permission \xrightarrow{o}$
.7 $SEM`BlkEnv$ renamed $MkBlkEnv$;
.8 $ValSetToSeq : SEM`VAL\text{-}\mathsf{set} \xrightarrow{o} SEM`VAL^*$ renamed $ValSetToSeq$;
.9 $CombineBlkEnv : SEM`BlkEnv \times SEM`BlkEnv \xrightarrow{o} SEM`BlkEnv$ renamed $CombineBlkEnv$;
.10 $MkEmptyBlkEnv : (SEM`Permission) \xrightarrow{o} SEM`BlkEnv$ renamed $MkEmptyBlkEnv$;
.11 $ExtractTagName : AS`Name \xrightarrow{o} [AS`Name] \times \mathbb{B}$;
.12 $DistribCombineBlkEnv : SEM`BlkEnv\text{-}\mathsf{set} \xrightarrow{o} SEM`BlkEnv\text{-}\mathsf{set}$ renamed $DistribCombineBlkEnv$;
.13 $SeqOfSetOf2SetOfSeqOf : (SEM`VAL \mid SEM`BlkEnv)\text{-}\mathsf{set}^* \xrightarrow{o}$
.14 $(SEM`VAL \mid SEM`BlkEnv)^*\text{-}\mathsf{set}$ renamed $SeqOfSetOf2SetOfSeqOf$,

1180.0 from DEF all ,
1181.0 from POS all ,
1182.0 from REP all ,
1183.0 from SEM all ,
1184.0 from $CMPL$ all ,
1185.0 from $CPAT$ all ,
1186.0 from $STKM$ all ,
1187.0 from $RTERR$ all ,
1188.0 from $STATE$ all ,
1189.0 from $GLOBAL$ all ,
1190.0 from $SCHDTP$ all ,
1191.0 from $INSTRTP$ all ,
1192.0 from $TIMEMAP$ all ,
1193.0 from $SETTINGS$
1194.0 operations $DTC : () \xrightarrow{o} \mathbb{B}$ renamed DTC ,
1195.0 from $TIMEPARSER$ all

exports

1196.0 types struct $PARTITION$

1197.0 operations $GetExpr : AS`Pattern \xrightarrow{o} AS`Expr$;
.1 $MatchLists : STKM`Pattern^* \times SEM`VAL^* \xrightarrow{o} SEM`BlkEnv\text{-}\mathsf{set}$;
.2 $SelPattern : AS`Bind \xrightarrow{o} AS`Pattern$;
.3 $ConstructFN : AS`FnDef \xrightarrow{o} SEM`BlkEnv \times AS`Name \xrightarrow{m} (AS`Expr \mid$
NOTYETSPEC |
.4 SUBRESP);
.5 $PatternMatch : STKM`Pattern \times SEM`VAL \xrightarrow{o} SEM`BlkEnv\text{-}\mathsf{set}$;
.6 $DoCarePattern : AS`PatternBind \times AS`Name \xrightarrow{o} AS`PatternBind$;
.7 $EvalMultBindSeq : STKM`Pattern^* \times SEM`VAL^* \times PARTITION \xrightarrow{o}$
.8 $SEM`BlkEnv\text{-}\mathsf{set} \mid (SEM`BlkEnv\text{-}\mathsf{set})\text{-}\mathsf{set}$

definitions
types

1198.0 $PARTITION = \mathrm{DO\_PARTITION} \mid \mathrm{DONT\_PARTITION}$

This type is used to control where looseness shall apear.

1.20.1 Pattern Matching

operations

1199.0 $PatternMatch : STKM`Pattern \times SEM`VAL \xrightarrow{o} SEM`BlkEnv$-set
.1 $PatternMatch\,(pat\text{-}p, val\text{-}v) \triangleq$
.2 cases true:
.3 (is-$STKM`PatternName\,(pat\text{-}p)) \rightarrow$
.4 let mk-$STKM`PatternName\,(id, \text{-}) = pat\text{-}p$ in
.5 if $id =$ nil
.6 then return $\{MkEmptyBlkEnv\,(\mathrm{READ\_ONLY})\}$
.7 else return $\{MkBlkEnv\,(id, val\text{-}v,$ nil $, \mathrm{READ\_ONLY})\}$,
.8 (is-$STKM`SetEnumPattern\,(pat\text{-}p)) \rightarrow MatchSetEnumPattern(pat\text{-}p, val\text{-}v)$,
.9 (is-$STKM`SetUnionPattern\,(pat\text{-}p)) \rightarrow MatchSetUnionPattern(pat\text{-}p, val\text{-}v)$,
.10 (is-$STKM`SeqEnumPattern\,(pat\text{-}p)) \rightarrow MatchSeqEnumPattern(pat\text{-}p, val\text{-}v)$,
.11 (is-$STKM`SeqConcPattern\,(pat\text{-}p)) \rightarrow MatchSeqConcPattern(pat\text{-}p, val\text{-}v)$,
.12 (is-$STKM`TuplePattern\,(pat\text{-}p)) \rightarrow MatchTuplePattern(pat\text{-}p, val\text{-}v)$,
.13 (is-$STKM`RecordPattern\,(pat\text{-}p)) \rightarrow MatchRecordPattern(pat\text{-}p, val\text{-}v)$,
.14 (is-$STKM`MatchVal\,(pat\text{-}p)) \rightarrow$
.15 if $pat\text{-}p.val = val\text{-}v$
.16 then return $\{MkEmptyBlkEnv\,(\mathrm{READ\_ONLY})\}$
.17 else return $\{\}$
.18 end;

The operation *PatternMatch* takes a pattern/bind and a semantic value as input, and returns the set with possible block environments. In each block environment the identifiers defined in the input pattern/bind are bound to the corresponding value from the input semantic value. An empty return set indicates no matching. A return set with an empty block environment means that the pattern matched, but that no binding were made.

1200.0 $MatchSetEnumPattern : STKM`SetEnumPattern \times SEM`VAL \xrightarrow{o} SEM`BlkEnv$-set
.1 $MatchSetEnumPattern\,($mk-$STKM`SetEnumPattern\,(elems\text{-}lp, \text{-}), val\text{-}v) \triangleq$
.2 if is-$SEM`SET\,(val\text{-}v)$
.3 then let mk-$SEM`SET\,(val\text{-}sv) = val\text{-}v$ in
.4 if card $val\text{-}sv =$ card elems $elems\text{-}lp$
.5 then let $perm\text{-}slv = Permute\,(ValSetToSeq\,(val\text{-}sv))$ in
.6 return $\bigcup\{MatchLists\,(elems\text{-}lp, tmp\text{-}lv) \mid$
.7 $tmp\text{-}lv \in perm\text{-}slv\}$
.8 else return $\{\}$

```
    else return {} ;
```

This operation returns the set with all possible binding environments. We first create the set of all permutations of the input semantic value. Then, we return the distributed union of all matchings of the input pattern with the elements from the permutation.

```
1201.0 MatchSetUnionPattern : STKM`SetUnionPattern × SEM`VAL -o-> SEM`BlkEnv-set
       MatchSetUnionPattern (mk-STKM`SetUnionPattern (lp-p, rp-p, -), val-v) ≜
         (dcl envres-sl : SEM`BlkEnv-set := {};
          if is-SEM`SET (val-v)
          then let mk-SEM`SET (val-sv) = val-v in
                 (for all mk- (setl-sv, setr-sv) ∈
                        {mk- (setl-sv, setr-sv) |
                              setl-sv, setr-sv ∈ F val-sv ·
                                   (setl-sv ∪ setr-sv = val-sv) ∧
                                   (setl-sv ∩ setr-sv = {}) ∧
                                   ((card val-sv > 1 ∧
                                     (¬ is-STKM`SetEnumPattern (lp-p) ∧
                                      ¬ is-STKM`SetEnumPattern (rp-p)))  ⇒
                                    (setl-sv ≠ {} ∧
                                     setr-sv ≠ {}))}
                  do let envl-s = PatternMatch (lp-p, mk-SEM`SET (setl-sv)),
                         envr-s = PatternMatch (rp-p, mk-SEM`SET (setr-sv)) in
                     if envl-s ≠ {} ∧ envr-s ≠ {}
                     then let tmpenv = ⋃ {DistribCombineBlkEnv ({tmp1, tmp2}) |
                                              tmp1 ∈ envl-s, tmp2 ∈ envr-s} in
                            envres-sl := envres-sl ∪ tmpenv;
                  return envres-sl )
          else return {} );
```

In the case of a set union pattern, we first create all pairs of set values, for which the union is equal to the original input set value, but are still disjoint. For each pair, we create two sets of binding environments. These are merged, and inserted into the resulting set of binding environments after duplicate entries are removed.

```
1202.0 MatchSeqEnumPattern : STKM`SeqEnumPattern × SEM`VAL -o-> SEM`BlkEnv-set
       MatchSeqEnumPattern (mk-STKM`SeqEnumPattern (els-lp, -), val-v) ≜
         if is-SEM`SEQ (val-v)
         then let mk-SEM`SEQ (val-lv) = val-v in
                MatchLists(els-lp, val-lv)
         else return {} ;
```

Here, we can directly create the set of binding environments, because the order in the sequence is important.

```
1203.0 MatchSeqConcPattern : STKM`SeqConcPattern × SEM`VAL →ᵒ SEM`BlkEnv-set
    .1 MatchSeqConcPattern (mk-STKM`SeqConcPattern (lp-p, rp-p, -), val-v) △
    .2   (dcl envres-sl : SEM`BlkEnv-set := {};
    .3    if is-SEM`SEQ (val-v)
    .4    then let mk-SEM`SEQ (val-lv) = val-v in
    .5          let tmp-s = if len val-lv > 1 ∧
    .6                         (¬ is-STKM`SeqEnumPattern (lp-p) ∧
    .7                          ¬ is-STKM`SeqEnumPattern (rp-p))
    .8                      then {mk- ([val-lv (j) | j ∈ {1, . . . , i}],
    .9                                 [val-lv (j) | j ∈ {i + 1, . . . , len val-lv}]) |
    .10                               i ∈ {1, . . . , len val-lv − 1}}
    .11                     else {mk- ([val-lv (j) | j ∈ {1, . . . , i}],
    .12                                [val-lv (j) | j ∈ {i + 1, . . . , len val-lv}]) |
    .13                              i ∈ if is-STKM`SeqEnumPattern (lp-p)
    .14                                  then {len lp-p.els}
    .15                                  else if is-STKM`SeqEnumPattern (rp-p)
    .16                                       then {len val-lv − len rp-p.els}
    .17                                       else (inds val-lv)} in
    .18         (for all mk- (seql-lv, seqr-lv) ∈ tmp-s
    .19          do let envl-sl = PatternMatch (lp-p, mk-SEM`SEQ (seql-lv)),
    .20                 envr-sl = PatternMatch (rp-p, mk-SEM`SEQ (seqr-lv)) in
    .21             if envl-sl ≠ {} ∧ envr-sl ≠ {}
    .22             then let tmp-sl = ⋃ {DistribCombineBlkEnv ({tmpl-l, tmpr-l}) |
    .23                                  tmpl-l ∈ envl-sl,
    .24                                  tmpr-l ∈ envr-sl} in
    .25                  envres-sl := envres-sl ∪ tmp-sl;
    .26          return envres-sl )
    .27    else return {} );
```

For sequence concatenation patterns, we first create the set of all possible pairs of sequences. Next, for each pair, the set of binding environments is created, and inserted into the result set. Duplicates are also stripped.

```
1204.0 MatchTuplePattern : STKM`TuplePattern × SEM`VAL →ᵒ SEM`BlkEnv-set
    .1 MatchTuplePattern (mk-STKM`TuplePattern (fields-lp, -), val-v) △
    .2   if is-SEM`TUPLE (val-v)
    .3   then let mk-SEM`TUPLE (val-lv) = val-v in
    .4          MatchLists(fields-lp, val-lv)
    .5   else return {} ;
```

We return the set of environments that is the result of the matching of the separate fields in the tuple value.

1205.0 $MatchRecordPattern : STKM`RecordPattern \times SEM`VAL \xrightarrow{o} SEM`BlkEnv\text{-set}$

$MatchRecordPattern\,(\text{mk-}STKM`RecordPattern\,(ptag, fields\text{-}lp, \text{-}), val\text{-}v) \triangleq$
$\quad \text{if is-}SEM`REC\,(val\text{-}v)$
$\quad \text{then let mk-}SEM`REC\,(vtag, v, v\text{-}dc) = val\text{-}v \text{ in}$
$\quad\quad \text{let mk-}\,(vtagname, v\text{-}isit) = AUX`ExtractTagName\,(vtag),$
$\quad\quad\quad \text{mk-}\,(ptagname, p\text{-}isit) = AUX`ExtractTagName\,(ptag) \text{ in}$
$\quad\quad \text{if } \neg\,(v\text{-}isit \wedge p\text{-}isit)$
$\quad\quad \text{then } ErrorOp(\text{"}Unknowntag * vtagname * or * ptagname * \text{"})$
$\quad\quad \text{else if } vtagname = ptagname$
$\quad\quad\quad \text{then if } v\text{-}dc = \{\mapsto\}$
$\quad\quad\quad\quad \text{then let } v\text{-}l = [v\,(i) \mid i \in \text{dom } v] \text{ in}$
$\quad\quad\quad\quad\quad MatchLists(fields\text{-}lp, v\text{-}l)$
$\quad\quad\quad\quad \text{else let } v\text{-}l = [\text{if } i \in \text{dom } v$
$\quad\quad\quad\quad\quad\quad\quad \text{then } v\,(i)$
$\quad\quad\quad\quad\quad\quad\quad \text{else } v\text{-}dc\,(i) \mid$
$\quad\quad\quad\quad\quad\quad\quad\quad i \in \text{dom } v \cup \text{dom } v\text{-}dc] \text{ in}$
$\quad\quad\quad\quad\quad (\text{if } DTC\,()$
$\quad\quad\quad\quad\quad \text{then for all } i \in \text{ dom } v\text{-}dc$
$\quad\quad\quad\quad\quad\quad \text{do let } pat = fields\text{-}lp\,(i) \text{ in}$
$\quad\quad\quad\quad\quad\quad\quad \text{if } \neg\,\text{is-}STKM`PatternName\,(pat)$
$\quad\quad\quad\quad\quad\quad\quad \text{then } RTERR`Error(RTERR`DC\text{-}NOT\text{-}PATTERN\text{-}NAME, \text{nil}, \text{nil}, []);$
$\quad\quad\quad\quad\quad MatchLists(fields\text{-}lp, v\text{-}l)\,)$
$\quad\quad\quad \text{else return } \{\}$
$\quad \text{else return } \{\}\ ;$

We return the set of environments that is the result of the matching of the separate fields in the record value.

1206.0 $MatchLists : STKM`Pattern^* \times SEM`VAL^* \xrightarrow{o} SEM`BlkEnv\text{-set}$

$MatchLists\,(els\text{-}lp, val\text{-}lv) \triangleq$
$\quad \text{if len } val\text{-}lv = \text{len } els\text{-}lp$
$\quad \text{then if } val\text{-}lv = []$
$\quad\quad \text{then return } \{MkEmptyBlkEnv\,(\text{READ\_ONLY})\}$
$\quad\quad \text{else let } tmp\text{-}ls = [PatternMatch\,(els\text{-}lp\,(i), val\text{-}lv\,(i)) \mid$
$\quad\quad\quad\quad\quad\quad i \in \text{inds } els\text{-}lp] \text{ in}$
$\quad\quad\quad \text{if } \{\} \notin \text{elems } tmp\text{-}ls$
$\quad\quad\quad \text{then let } perm\text{-}s = SeqOfSetOf2SetOfSeqOf\,(tmp\text{-}ls) \text{ in}$
$\quad\quad\quad\quad \text{return } \bigcup\,(\{DistribCombineBlkEnv\,(\text{elems } l) \mid l \in perm\text{-}s\})$
$\quad\quad\quad \text{else return } \{\}$
$\quad \text{else return } \{\}\ ;$

For each element from the input pattern sequence, we create a set of binding environments by matching the pattern with the corresponding element from the input value sequence. If all elements match, each set of environments from the sequence *tmp-ls* is merged into a set with one single environment.

```
1207.0 EvalMultBindSeq : STKM`Pattern* × SEM`VAL* × PARTITION -o->
                           SEM`BlkEnv-set | (SEM`BlkEnv-set)-set
EvalMultBindSeq (pat-lp, seq-lv, partition) ≜
  (dcl env-ls : (SEM`BlkEnv-set)* := [];
   if ∀ elm-v ∈ elems seq-lv · is-SEM`SET (elm-v)
   then let seq-lsv = [seq-lv (i).v | i ∈ inds seq-lv] in
        let tmp-slv = SeqOfSetOf2SetOfSeqOf (seq-lsv) in
        (for all val-lv ∈  tmp-slv
         do env-ls := env-ls ⁀ [MatchLists (pat-lp, val-lv)];
         if partition = DO_PARTITION
         then return Partition (env-ls)
         else return Collapse (env-ls) )
   else RTERR`ErrorVal(RTERR`SET-EXPECTED, nil , nil , []) );
```

This operation returns the set of all binding environments for multiple bindings.

This operation returns the set of binding environments for the pattern and all the elements of the set expression.

1.20.2 Constructing Block environments for functions

```
1208.0 ConstructFN : AS`FnDef -o-> SEM`BlkEnv×AS`Name -m-> (AS`Expr | NOTYETSPEC |
                      SUBRESP)
ConstructFN (fn) ≜
  (dcl benv : SEM`BlkEnv,
       b-m : AS`Name -m-> (AS`Expr | NOTYETSPEC) := {↦},
       clmodName : AS`Name := CMPL`GetClMod (),
       mp : AS`Name -m-> (SEM`FN | SEM`ExplPOLY);
   if is-AS`ExplFnDef (fn)
   then let mk-AS`ExplFnDef (nm, tpparms, -, -, body, fnpre, fnpost, access, static, -) = fn in
        (benv := MkBlkEnv (nm, ConstructExplFN (fn), nil , READ_ONLY);
         b-m := {nm ↦ body.body};
         if fnpre ≠ nil
         then b-m := b-m ⊎ {mk-AS`Name (["pre-" ⁀ nm.ids (1)], nm.cid) ↦
                          fnpre};
         if fnpost ≠ nil
         then b-m := b-m ⊎ {mk-AS`Name (["post-" ⁀ nm.ids (1)], nm.cid) ↦
                          fnpost};
         if tpparms = []
         then let m = DEF`CreateExplPrePostFns (clmodName, fn) in
              mp := {nm ↦ let mk- (v, -) = m (nm) in
                          v |
                              nm ∈ dom m}
```

```
            else let m = DEF`CreateExplPolyPrePostFns (clmodName, fn) in
                  mp := {nm ↦ let mk- (v, -) = m (nm) in
                              v |
                                  nm ∈ dom m};
            for all nm ∈ dom mp
            do benv := CombineBlkEnv (benv, MkBlkEnv (nm, mp (nm), nil , READ_ONLY)))
    elseif is-AS`ImplFnDef (fn)
    then let mk-AS`ImplFnDef (nm, tpparms, -, -, -, -, access, static, -) = fn in
          (benv := MkBlkEnv (nm, mk-SEM`ImplFN (), nil , READ_ONLY);
            if tpparms = []
            then let m = DEF`CreateImplPrePostFns (clmodName, fn) in
                   mp := {nm ↦ let mk- (v, -) = m (nm) in
                               v |
                                   nm ∈ dom m}
            else let m = DEF`CreateImplPolyPrePostFns (clmodName, fn) in
                  mp := {nm ↦ let mk- (v, -) = m (nm) in
                              v |
                                  nm ∈ dom m};
            let nm ∈ dom mp in
            benv := CombineBlkEnv (benv,
                                              MkBlkEnv (nm, mp (nm), nil , READ_ONLY)))
    else let mk-AS`ExtExplFnDef (nm, tpparms, -, -, body, fnpre, fnpost, access,
                                            static, -) =
                  fn in
          (benv := MkBlkEnv (nm, ConstructExtExplFN (fn), nil , READ_ONLY);
           b-m := {nm ↦ body.body};
           if fnpre ≠ nil
           then b-m := b-m ⊎ {mk-AS`Name (["pre-" ⁀ nm.ids (1)], nm.cid) ↦
                              fnpre};
           if fnpost ≠ nil
           then b-m := b-m ⊎ {mk-AS`Name (["post-" ⁀ nm.ids (1)], nm.cid) ↦
                              fnpost};
           if tpparms = []
           then let m = DEF`CreateExplPrePostFns (clmodName, fn) in
                  mp := {nm ↦ let mk- (v, -) = m (nm) in
                              v |
                                  nm ∈ dom m}
           else let m = DEF`CreateExplPolyPrePostFns (clmodName, fn) in
                 mp := {nm ↦ let mk- (v, -) = m (nm) in
                             v |
                                 nm ∈ dom m};
           let nm ∈ dom mp in
           benv := CombineBlkEnv (benv,
                                             MkBlkEnv (nm, mp (nm), nil , READ_ONLY)));
    return mk- (benv, b-m) );
```

This function returns a semantic function value for a function definition. The closure environment is created with an initial set of identifiers (*id-s*).

```
1209.0 ConstructExplFN : AS`ExplFnDef →(o) SEM`VAL
ConstructExplFN (fndef) △
  let mk-AS`ExplFnDef (nm, tpparms, tp, parms, -, -, -, access, static, -) = fndef in
  let clmodName = CMPL`GetClMod () in
  let p-l-l = [CPAT`PL2PL (parms (i)) | i ∈ inds parms],
      p-id = CMPL`CompileFnOpDef (fndef),
      env = mk-SEM`BlkEnv ({↦}, READ_ONLY) in
  if tpparms = []
  then return SEM`CompFN (mk-SEM`ExplFN (tp, p-l-l, p-id, env,
                                          {↦}, nm, clmodName, nil , nil ))
  else return mk-SEM`ExplPOLY (tpparms, tp, p-l-l, p-id,
                               env, nm, clmodName, nil , nil ) ;
```

```
1210.0 ConstructExtExplFN : AS`ExtExplFnDef →(o) SEM`VAL
ConstructExtExplFN (fndef) △
  let mk-AS`ExtExplFnDef (nm, tpparms, partps, resnmtps, -, -, -, access, static,
                          -) =
        fndef in
  let clmodName = CMPL`GetClMod (),
      mk- (fn-tp, -) = DEF`ImplicitTypeParams (partps),
      fnrng = DEF`ImplicitResType (resnmtps) in
  let p-l-l = [CPAT`PL2PL (partps (i).pats) | i ∈ inds partps],
      p-id = CMPL`CompileFnOpDef (fndef),
      env = mk-SEM`BlkEnv ({↦}, READ_ONLY),
      tp = mk-AS`TotalFnType (fn-tp, fnrng, CI`NilContextId) in
  if tpparms = []
  then return SEM`CompFN (mk-SEM`ExplFN (tp, p-l-l, p-id, env,
                                          {↦}, nm, clmodName, nil , nil ))
  else return mk-SEM`ExplPOLY (tpparms, tp, p-l-l, p-id,
                               env, nm, clmodName, nil , nil ) ;
```

We first create the closure environment for the body of the function, and then return a semantic (non-)polymorphic function value.

```
1211.0 SelPattern : AS`Bind →(o) AS`Pattern
SelPattern (bind) △
  return bind.pat;
```

This operation returns the pattern from a bind.

```
1212.0 GetExpr : AS`Pattern →o AS`Expr
    .1 GetExpr (pat) △
    .2   cases pat:
    .3     mk-AS`PatternName (id, cid) → if id = nil
    .4                                   then GetExpr(DoCarePattern (pat, mk-AS`Name (["1"], cid)))
    .5                                   else return id ,
    .6     mk-AS`MatchVal (val,-) → return val,
    .7     mk-AS`SetEnumPattern (els, cid) → let e-l = [GetExpr (els (i)) | i ∈ inds els] in
    .8                                       return mk-AS`SetEnumerationExpr (e-l, cid),
    .9     mk-AS`SetUnionPattern (lp, rp, cid) → return mk-AS`BinaryExpr (GetExpr (lp),
   .10                                                                    SETUNION,
   .11                                                                    GetExpr (rp), cid),
   .12     mk-AS`SeqEnumPattern (els, cid) → let e-l = [GetExpr (els (i)) | i ∈ inds els] in
   .13                                       return mk-AS`SeqEnumerationExpr (e-l, cid),
   .14     mk-AS`SeqConcPattern (lp, rp, cid) → return mk-AS`BinaryExpr (GetExpr (lp),
   .15                                                                   SEQCONC,
   .16                                                                   GetExpr (rp), cid),
   .17     mk-AS`RecordPattern (tag, p-l, cid) → let e-l = [GetExpr (p-l (i)) | i ∈ inds p-l] in
   .18                                           return mk-AS`RecordConstructorExpr (tag, e-l, cid),
   .19     mk-AS`TuplePattern (p-l, cid) → let e-l = [GetExpr (p-l (i)) | i ∈ inds p-l] in
   .20                                     return mk-AS`TupleConstructorExpr (e-l, cid),
   .21     others → error
   .22   end;
```

This operation generates an expression from a pattern.

```
1213.0 Partition : (SEM`BlkEnv-set)* →o (SEM`BlkEnv-set)-set
    .1 Partition (env-ls) △
    .2   let env-ls-m = [env-ls (i) | i ∈ inds (env-ls) · env-ls (i) ≠ {}] in
    .3   if env-ls-m ≠ []
    .4   then let env-ss = SeqOfSetOf2SetOfSeqOf (env-ls-m) in
    .5        return {elems (env-s) | env-s ∈ env-ss}
    .6   else return {{}} ;
```

```
1214.0 Collapse : (SEM`BlkEnv-set)* →o SEM`BlkEnv-set
    .1 Collapse (env-ls) △
    .2   return ⋃(elems (env-ls))
```

Theese two operations control where looseness apears.
values

1215.0 $ITC = \{0 \mapsto \text{'0'},$
.1 $1 \mapsto \text{'1'},$
.2 $2 \mapsto \text{'2'},$
.3 $3 \mapsto \text{'3'},$
.4 $4 \mapsto \text{'4'},$
.5 $5 \mapsto \text{'5'},$
.6 $6 \mapsto \text{'6'},$
.7 $7 \mapsto \text{'7'},$
.8 $8 \mapsto \text{'8'},$
.9 $9 \mapsto \text{'9'}\}$

This value is used to convert natural numbers into string.

functions

1216.0 $NatToId : \mathbb{N} \to AS`Id$
.1 $NatToId\,(n) \triangleq$
.2 if $n = 0$
.3 then "0"
.4 else (let $ndt = n$ div 10 in
.5 if $ndt = 0$
.6 then ""
.7 else $NatToId\,(n$ div $10)) \curvearrowright$
.8 $[ITC\,(n$ mod $10)];$

This function converts a natural number into a string.

1217.0 $NewBase : AS`Name \times \mathbb{N} \to AS`Name$
.1 $NewBase\,(nm, n) \triangleq$
.2 let $str = NatToId\,(n),$
.3 mk-$AS`Name\,(id, cid) = nm$ in
.4 mk-$AS`Name\,(id \curvearrowright [str] \curvearrowright$ ["-"]$, cid)$

Based on the input identifier, and the input number, a new identifier is created.

operations

1218.0 $DoCarePattern : AS`PatternBind \times AS`Name \xrightarrow{o} AS`PatternBind$
.1 $DoCarePattern\,(pat\text{-}p, id\text{-}base) \triangleq$
.2 cases $pat\text{-}p$:
.3 mk-$AS`PatternName\,(id, cid) \to$ if $id =$ nil
.4 then return mk-$AS`PatternName\,(id\text{-}base, cid)$
.5 else return $pat\text{-}p$,
.6 mk-$AS`MatchVal$ (-, -) $\to$ return $pat\text{-}p$,

```
    mk-AS`SetEnumPattern (els, cid) → let e-l = [DoCarePattern (els (i),
                                                   NewBase (id-base, i)) |

                                           i ∈ inds els] in
                                     return mk-AS`SetEnumPattern (e-l, cid),
    mk-AS`SetUnionPattern (lp, rp, cid) → return mk-AS`SetUnionPattern
                                           (
                                            DoCarePattern (lp,
                                                    NewBase (id-base, 1)),
                                            DoCarePattern (rp,
                                                    NewBase (id-base, 2)),
                                            cid),
    mk-AS`SeqEnumPattern (els, cid) → let e-l = [DoCarePattern (els (i),
                                                   NewBase (id-base, i)) |

                                           i ∈ inds els] in
                                     return mk-AS`SeqEnumPattern (e-l, cid),
    mk-AS`SeqConcPattern (lp, rp, cid) → return mk-AS`SeqConcPattern
                                          (
                                           DoCarePattern (lp,
                                                   NewBase (id-base, 1)),
                                           DoCarePattern (rp,
                                                   NewBase (id-base, 2)),
                                           cid),
    mk-AS`RecordPattern (tag, p-l, cid) → let e-l = [DoCarePattern (p-l (i),
                                                       NewBase (id-base, i)) |

                                               i ∈ inds p-l] in
                                         return mk-AS`RecordPattern (tag, e-l, cid),
    mk-AS`TuplePattern (p-l, cid) → let e-l = [DoCarePattern (p-l (i),
                                                 NewBase (id-base, i)) |

                                         i ∈ inds p-l] in
                                   return mk-AS`TuplePattern (e-l, cid),
    mk-AS`SetBind (pat, set-e, cid) → let new-pat = DoCarePattern (pat,
                                                  NewBase (id-base, 1)) in
                                     return mk-AS`SetBind (new-pat, set-e, cid),
    mk-AS`TypeBind (pat, tp, cid) → let new-pat = DoCarePattern (pat,
                                                NewBase (id-base, 1)) in
                                   return mk-AS`TypeBind (new-pat, tp, cid),
    others → error
  end
```

end *PAT*

This operation replaces all "don't care" patterns in the input pattern with unique pattern identifiers.

Test Suite : rtinfo.ast
Module : PAT

Name	**#Calls**	**Coverage**
PAT‘GetExpr	**undefined**	**undefined**
PAT‘NatToId	**undefined**	**undefined**
PAT‘NewBase	**undefined**	**undefined**
PAT‘Collapse	**undefined**	**undefined**
PAT‘Partition	**undefined**	**undefined**
PAT‘MatchLists	**undefined**	**undefined**
PAT‘SelPattern	**undefined**	**undefined**
PAT‘ConstructFN	**undefined**	**undefined**
PAT‘PatternMatch	**undefined**	**undefined**
PAT‘DoCarePattern	**undefined**	**undefined**
PAT‘ConstructExplFN	**undefined**	**undefined**
PAT‘EvalMultBindSeq	**undefined**	**undefined**
PAT‘MatchTuplePattern	**undefined**	**undefined**
PAT‘ConstructExtExplFN	**undefined**	**undefined**
PAT‘MatchRecordPattern	**undefined**	**undefined**
PAT‘MatchSeqConcPattern	**undefined**	**undefined**
PAT‘MatchSeqEnumPattern	**undefined**	**undefined**
PAT‘MatchSetEnumPattern	**undefined**	**undefined**
PAT‘MatchSetUnionPattern	**undefined**	**undefined**
Total Coverage		**0%**

1.21 Auxiliary Functions and Operations

The module AUX contains a collection of auxiliary functions and operations.
module AUX

imports

1219.0 from AS all ,

1220.0 from CI all ,

1221.0 from PAT all ,

1222.0 from POS all ,

1223.0 from REP all ,

1224.0 from SEM all ,

1225.0 from $EXPR$ all ,

1226.0 from $STKM$ all ,

1227.0 from $RTERR$ all ,

1228.0 from $STATE$

1229.0 operations $AccessOk : AS`Access \times GLOBAL`OrigCl \times AS`Name \xrightarrow{o} \mathbb{B}$;
.1 $IsAClass : AS`Name \xrightarrow{o} \mathbb{B}$;
.2 $GetRecSel : AS`Name \xrightarrow{o}$
.3 $AS`Name \xrightarrow{m} (GLOBAL`RecSel \times AS`Access)$;
.4 $GetLocalTps : AS`Name \xrightarrow{o} AS`Name \xrightarrow{m} AS`TypeDef$;
.5 $GetAllSupers : AS`Name \xrightarrow{o} AS`Name$-set;
.6 $ExistsOneChild : AS`Name$-set $\xrightarrow{o} \mathbb{B} \times [AS`Name]$,

1230.0 from $GLOBAL$ all ,

1231.0 from $SCHDTP$ all ,

1232.0 from $INSTRTP$ all

exports

1233.0 functions $IsStmt : AS`Expr \mid AS`Stmt \to \mathbb{B}$;
.1 $Permute : SEM`VAL^* \to SEM`VAL^*$-set;
.2 $PreName : AS`Name \to AS`Name$;
.3 $ErrorFct$: char* $\to$
.4 $\mathbb{B} \mid AS`Name$-set $\mid AS`Name \mid$
.5 $GLOBAL`InhCon$;
.6 $PostName : AS`Name \to AS`Name$;
.7 $ExtractId : AS`Name \to AS`Id$;
.8 $EqualNames : AS`Name \times AS`Name \to \mathbb{B}$;
.9 $ExtractName : AS`Name \to AS`Name$;
.10 $ConstructName : AS`Id \times CI`ContextId \to AS`Name$;
.11 $SingleNameToString : AS`Name \to$ char*;
.12 $ConstructDoubleName : AS`Name \times AS`Name \to AS`Name$

1234.0 operations $IsInt : SEM`VAL \xrightarrow{o} \mathbb{B};$
.1 $IsNat : SEM`VAL \xrightarrow{o} \mathbb{B};$
.2 $IsRat : SEM`VAL \xrightarrow{o} \mathbb{B};$
.3 $IsReal : SEM`VAL \xrightarrow{o} \mathbb{B};$
.4 $Ceiling : \mathbb{R} \xrightarrow{o} \mathbb{Z};$
.5 $ErrorOp : \mathsf{char}^* \xrightarrow{o}$
.6 $\mathbb{B} \mid SEM`VAL \mid AS`Name\text{-}\mathsf{set} \mid$
.7 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL]) \mid AS`Name \mid$
.8 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL] \times [SEM`VAL] \times [AS`Name] \times [AS`Access]) \mid$
.9 $SEM`BlkEnv\text{-}\mathsf{set} \mid (\mathbb{B} \times [GLOBAL`Type] \times [AS`Invariant] \times [AS`Name]) \mid$
.10 $GLOBAL`RecSel;$
.11 $IsNatOne : SEM`VAL \xrightarrow{o} \mathbb{B};$
.12 $IsRecSel : AS`Name \xrightarrow{o} \mathbb{B} \times [AS`Name \xrightarrow{m} \mathbb{N}];$
.13 $MkBlkEnv : AS`Name \times SEM`VAL \times [AS`Type] \times SEM`Permission \xrightarrow{o}$
$SEM`BlkEnv;$
.14 $SetToSeq : SEM`VAL\text{-}\mathsf{set} \xrightarrow{o} SEM`VAL^*;$
.15 $IsTypeDef : AS`Name \xrightarrow{o}$
.16 $\mathbb{B} \times [GLOBAL`Type] \times [AS`Invariant] \times [AS`Name] \times [AS`Access];$
.17 $ValSetToSeq : SEM`VAL\text{-}\mathsf{set} \xrightarrow{o} SEM`VAL^*;$
.18 $ErrorEmptyOp : \mathsf{char}^* \xrightarrow{o} ();$
.19 $LookUpRecSel : AS`Name \xrightarrow{o} [GLOBAL`RecSel];$
.20 $CombineBlkEnv : SEM`BlkEnv \times SEM`BlkEnv \xrightarrow{o} SEM`BlkEnv;$
.21 $MkEmptyBlkEnv : SEM`Permission \xrightarrow{o} SEM`BlkEnv;$
.22 $ExtractTagName : AS`Name \xrightarrow{o} [AS`Name] \times \mathbb{B};$
.23 $DistribCombineBlkEnv : SEM`BlkEnv\text{-}\mathsf{set} \xrightarrow{o} SEM`BlkEnv\text{-}\mathsf{set};$
.24 $SeqOfSetOf2SetOfSeqOf : (SEM`VAL \mid SEM`BlkEnv)\text{-}\mathsf{set}^* \xrightarrow{o}$
.25 $(SEM`VAL \mid SEM`BlkEnv)^*\text{-}\mathsf{set}$

definitions

1.21.1 Auxiliary Error Function

The operation/function *ErrorOp*/*ErrorFct* is a dummy operation/function which is used such that it can be specified which error message that should be returned. The right way to do this, is by using exception handling. However, this is not supported by the code generator.
operations

1235.0 $ErrorOp : \mathsf{char}^* \xrightarrow{o}$
.1 $\mathbb{B} \mid SEM`VAL \mid AS`Name\text{-}\mathsf{set} \mid$
.2 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL]) \mid AS`Name \mid$
.3 $(\mathbb{B} \times \mathbb{B} \times [SEM`VAL] \times [SEM`VAL] \times [AS`Name] \times [AS`Access]) \mid$
.4 $SEM`BlkEnv\text{-}\mathsf{set} \mid (\mathbb{B} \times [GLOBAL`Type] \times [AS`Invariant] \times [AS`Name]) \mid$
.5 $GLOBAL`RecSel$
.6 $ErrorOp\ (\text{-}) \triangleq$
.7 error;

1236.0 $ErrorEmptyOp : \mathsf{char}^* \xrightarrow{o} ()$
.1 $ErrorEmptyOp\,(\text{-}) \triangleq$
.2 $\quad$ error

functions

1237.0 $ErrorFct : \mathsf{char}^* \rightarrow$
.1 $\qquad\qquad \mathbb{B} \mid AS`Name\text{-}\mathsf{set} \mid AS`Name \mid$
.2 $\qquad\qquad GLOBAL`InhCon$
.3 $ErrorFct\,(\text{-}) \triangleq$
.4 $\quad$ undefined

1.21.2 Environments

operations

1238.0 $CombineBlkEnv : SEM`BlkEnv \times SEM`BlkEnv \xrightarrow{o} SEM`BlkEnv$
.1 $CombineBlkEnv\,(\mathsf{mk}\text{-}SEM`BlkEnv\,(env1, permission1), \mathsf{mk}\text{-}SEM`BlkEnv\,(env2, permission2)) \triangleq$
.2 $\quad$ if $(permission1 \neq permission2)$
.3 $\quad$ then error
.4 $\quad$ else return $\mathsf{mk}\text{-}SEM`BlkEnv\,(env1 \dagger env2, permission1)$;

This operation merges two block environments into a new block environment. If duplicate identifiers are bound, the binding is not overwritten.

1239.0 $DistribCombineBlkEnv : SEM`BlkEnv\text{-}\mathsf{set} \xrightarrow{o} SEM`BlkEnv\text{-}\mathsf{set}$
.1 $DistribCombineBlkEnv\,(blkenv\text{-}s) \triangleq$
.2 $\quad$ (dcl $result\text{-}m : AS`Name \xrightarrow{m} SEM`ValTp := \{\mapsto\}$;
.3 $\quad\;$ if $blkenv\text{-}s = \{\}$
.4 $\quad\;$ then return $\{\}$
.5 $\quad\;$ else let $some\text{-}blkenv \in blkenv\text{-}s$ in
.6 $\qquad\quad$ let $permission : SEM`Permission = some\text{-}blkenv.perm$ in
.7 $\qquad\quad$ (for all $\mathsf{mk}\text{-}SEM`BlkEnv\,(id\text{-}m, perm) \in blkenv\text{-}s$
.8 $\qquad\quad\;$ do if $perm \neq permission$
.9 $\qquad\qquad\;$ then error
.10 $\qquad\qquad\;$ else if $\exists\, id \in \mathsf{dom}\ id\text{-}m \cdot$
.11 $\qquad\qquad\qquad\qquad id \in \mathsf{dom}\ result\text{-}m \wedge$
.12 $\qquad\qquad\qquad\qquad id\text{-}m\,(id).val \neq result\text{-}m\,(id).val$
.13 $\qquad\qquad\quad$ then return $\{\}$
.14 $\qquad\qquad\quad$ else $result\text{-}m := result\text{-}m \dagger id\text{-}m$;
.15 $\qquad\quad\;$ return $\{\mathsf{mk}\text{-}SEM`BlkEnv\,(result\text{-}m, permission)\}$));

The operation *DistribCombineBlkEnv* merges a set of block environments, in the following way: If there exists one binding of the same identifier to different values in the set then a empty set is returned, otherwise a set of one block environment is returned.

1240.0 $MkBlkEnv : AS`Name \times SEM`VAL \times [AS`Type] \times SEM`Permission \xrightarrow{o} SEM`BlkEnv$
.1 $MkBlkEnv\,(id, val\text{-}v, tp, perm) \triangleq$
.2 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM`BlkEnv\,(\{id \mapsto \mathsf{mk}\text{-}SEM`ValTp\,(val\text{-}v, tp)\}, perm);$

The result of this operation is a block environment with only one binding.

1241.0 $MkEmptyBlkEnv : SEM`Permission \xrightarrow{o} SEM`BlkEnv$
.1 $MkEmptyBlkEnv\,(perm) \triangleq$
.2 $\quad \mathsf{return}\ \mathsf{mk}\text{-}SEM`BlkEnv\,(\{\mapsto\}, perm);$

This operation returns an empty block environment.

The operation *ExtractTagName* investigates if a tag name *name* is within the current scope. If it is the full tag name is returned, that is, the tag name qualified with the class name where the type definition of the tag is defined. Furthermore, a boolean is returned indicating if tag is defined within the current scope.

The current class scope *clnm* is looked up (using *STKM`GetCurCl*). If the tag name *name* is not qualified and the tag name is defined in a type definition in the current class, this class name is returned. Otherwise, the type definition of the tag is searched in the superlasses of class that qualifies the the tag name, or if tag name is not qualified, the superclasses of the current class.

In case several type definitions occurs within the superclasses, it is investigated if these classes are inherited in direct line, and if they are the type definition in the first subclass is used.

1242.0 $ExtractTagName : AS`Name \xrightarrow{o} [AS`Name] \times \mathbb{B}$
.1 $ExtractTagName\,(name) \triangleq$
.2 $\quad \mathsf{let}\ \mathsf{mk}\text{-}AS`Name\,(l \curvearrowright [id], cid) = name,$
.3 $\qquad thename = ExtractName\,(name)\ \mathsf{in}$
.4 $\quad (\mathsf{if}\ l = [\,] \wedge \neg\, STKM`HasCurCl\,()$
.5 $\quad\ \mathsf{then\ let}\ \text{-} = ErrorOp\,(\text{"}Unknownidentifier * nm * \text{"})\ \mathsf{in}$
.6 $\qquad\quad \mathsf{skip};$
.7 $\quad\ \mathsf{if}\ l = [\,]$
.8 $\quad\ \mathsf{then\ let}\ clnm = STKM`GetCurCl\,(),$
.9 $\qquad\qquad local\text{-}recsel = STATE`GetRecSel\,(clnm)\ \mathsf{in}$
.10 $\qquad\quad (\mathsf{if}\ thename \in \mathsf{dom}\ local\text{-}recsel$
.11 $\qquad\quad\ \mathsf{then\ let}\ tagname = ConstructDoubleName\,(clnm, name)\ \mathsf{in}$
.12 $\qquad\qquad\quad \mathsf{return}\ \mathsf{mk}\text{-}\,(tagname, \mathsf{true})\ );$
.13 $\quad\ \mathsf{let}\ classname = \mathsf{if}\ l \neq [\,]$
.14 $\qquad\qquad\qquad\qquad \mathsf{then}\ ConstructName\,(\mathsf{hd}\ l, cid)$
.15 $\qquad\qquad\qquad\qquad \mathsf{else\ nil}\ ,$

```
.16        allsupers = if l = []
.17                    then STATE`GetAllSupers (STKM`GetCurCl ())
.18                    else STATE`GetAllSupers (classname) ∪ {classname} in
.19    let recsels = {clname ↦ STATE`GetRecSel (clname) |
.20                      clname ∈ allsupers ·
.21                          thename ∈ dom STATE`GetRecSel (clname)} in
.22     cases dom recsels:
.23       {} → return mk- (nil , false),
.24       {cl} → let tag-name = ConstructDoubleName (cl, name) in
.25                return mk- (tag-name, true),
.26       - → if classname ∈ dom recsels
.27            then return mk- (name, true)
.28            else let mk- (doesthere, child) = STATE`ExistsOneChild (dom recsels) in
.29                 if doesthere
.30                 then let tag-name = ConstructDoubleName (child, name) in
.31                      return mk- (tag-name, true)
.32                 else return mk- (nil , false)
.33    end);
```

1.21.3 Types

This operation returns true if the input name is exported from the input module name.

The operation *IsTypeDef* computes if the the input name *name* denotes a type definition within the scope of the current object, if it is the type definition of the name and the class name in which the type definition belongs are returned. Notice, that if the type is a composite type the *tagname* is modified such that it also describes which class it is defined in. This is done by the auxiliary function *ExtComp*

The strategy of the operation is similar to the one used in *ExtractTagName*.

```
1243.0 IsTypeDef : AS`Name o→
    .1            𝔹 × [GLOBAL`Type] × [AS`Invariant] × [AS`Name] × [AS`Access]
    .2 IsTypeDef (name) △
    .3   (if STATE`IsAClass (name)
    .4    then let type = mk-GLOBAL`ObjRefType (name) in
    .5         return mk- (true, type, nil , name, nil )
    .6    else let mk-AS`Name (l ⁀ [-], cid) = name,
    .7             clnm = STKM`GetCurCl (),
    .8             origcl = STKM`GetOrigCl (),
    .9             objnm = STKM`GetCurObjName (),
```

```
.10              thename = ExtractName (name) in
.11          (if l = [] ∧ ¬ STKM`HasCurCl ()
.12           then RTERR`ErrorVal(RTERR`TYPE-UNKNOWN, nil , nil , []) ;
.13           if l = []
.14           then let tps = STATE`GetLocalTps (clnm) in
.15                  (if thename ∈ dom tps
.16                   then let typedef = ExtComp (tps (thename), clnm),
.17                            scopeok = STATE`AccessOk (typedef.access, origcl, clnm) in
.18                        return mk- (scopeok, typedef.shape, typedef.Inv,
.19                                    clnm, typedef.access) );
.20           let classname = if l ≠ []
.21                           then ConstructName (hd l, cid)
.22                           else objnm,
.23               allsupers = if l = []
.24                           then STATE`GetAllSupers (clnm)
.25                           else STATE`GetAllSupers (classname) ∪ {classname} in
.26           let spcl-tps = {clname ↦ STATE`GetLocalTps (clname) |
.27                               clname ∈ allsupers ·
.28                                   thename ∈ dom STATE`GetLocalTps (clname)} in
.29            cases  dom spcl-tps:
.30              {} → return mk- (false, nil , nil , nil , nil ),
.31              {cl} → let typedef = ExtComp (spcl-tps (cl) (thename), cl),
.32                         scopeok = STATE`AccessOk (typedef.access, origcl, cl) in
.33                     return mk- (scopeok, typedef.shape, typedef.Inv, cl,
.34                                 typedef.access),
.35              - → if classname ∈ dom spcl-tps
.36                  then let typedef = ExtComp (spcl-tps (classname) (thename), classname),
.37                           scopeok = STATE`AccessOk (typedef.access, origcl, classname) in
.38                        return mk- (scopeok, typedef.shape, typedef.Inv,
.39                                    classname, typedef.access)
.40                  else let mk- (doesthere, child) = STATE`ExistsOneChild (dom spcl-tps) in
.41                       if doesthere
.42                       then let typedef = ExtComp (spcl-tps (child) (thename), child),
.43                                scopeok = STATE`AccessOk (typedef.access, origcl, child) in
.44                            return mk- (scopeok, typedef.shape,
.45                                        typedef.Inv, child, typedef.access)
.46                       else RTERR`ErrorVal(RTERR`MULT-DEF, nil , nil , [])
.47           end));
```

```
1244.0  ExtComp : AS`TypeDef × AS`Name -o-> AS`TypeDef
    .1  ExtComp (mk-AS`TypeDef (nm, shape, Inv, access, cid), clnm) △
    .2    if is-AS`CompositeType (shape)
    .3    then let mk-AS`CompositeType (name, fields, cid2) = shape in
    .4           let tag-name = ConstructDoubleName (clnm, name) in
    .5           let new-shape = mk-AS`CompositeType (tag-name, fields, cid2) in
    .6           return mk-AS`TypeDef (tag-name, new-shape, Inv, access, cid)
    .7    else return mk-AS`TypeDef (nm, shape, Inv, access, cid) ;
```

1245.0 $IsRecSel : AS`Name \xrightarrow{o} \mathbb{B} \times [AS`Name \xrightarrow{m} \mathbb{N}]$
.1 $IsRecSel\,(nm) \triangleq$
.2 let mk-$AS`Name\,([cl, tag], cid) = nm$ in
.3 let $clname = ConstructName\,(cl, cid)$,
.4 $tagname = ConstructName\,(tag, cid)$ in
.5 let $recsel = STATE`GetRecSel\,(clname)$ in
.6 if $tagname \notin$ dom $recsel$
.7 then return mk- (false, nil)
.8 else let mk- (mk- (-, pos, -), $access$) $= recsel\,(tagname)$ in
.9 return mk- (true, pos) ;

This operation returns *true* if the input name denotes a record selector in the class that of the qualified name, defined by the input name. In this case the position map of the field selectors are also returned.

1246.0 $LookUpRecSel : AS`Name \xrightarrow{o} [GLOBAL`RecSel]$
.1 $LookUpRecSel$ (mk-$AS`Name\,([cl, tagname], cid)) \triangleq$
.2 let $clnm = ConstructName\,(cl, cid)$,
.3 $tgnm = ConstructName\,(tagname, cid)$ in
.4 let $recmap = STATE`GetRecSel\,(clnm)$ in
.5 if $tgnm \notin$ dom $recmap$
.6 then return nil
.7 else let mk- $(recsel, access) = recmap\,(tgnm)$ in
.8 return $recsel$;

The operation *LookUpRecSel* returns the record selector information for the input name, which is assumed to be a tag name. A tag name should contain the class name where the type is defined and the tag.

1.21.4 Mathematical Functions and Operations

1247.0 $Ceiling : \mathbb{R} \xrightarrow{o} \mathbb{Z}$
.1 $Ceiling\,(val) \triangleq$
.2 if is-$\mathbb{Z}\,(val)$
.3 then return val
.4 else return $1 +$ floor val ;

This operation returns the ceiling of a real value, i.e. the nearest integer greater or equal to the input value.

1248.0 $\mathit{IsNat} : \mathit{SEM}`\mathit{VAL} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{IsNat}\,(\mathit{val\text{-}v}) \triangleq$
.2 if is-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val\text{-}v})$
.3 then let mk-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val}) = \mathit{val\text{-}v}$ in
.4 return is-$\mathbb{N}\,(\mathit{val})$
.5 else return false ;

This operation returns true if the input semantic numeric value denotes a natural number.

1249.0 $\mathit{IsNatOne} : \mathit{SEM}`\mathit{VAL} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{IsNatOne}\,(\mathit{val\text{-}v}) \triangleq$
.2 if is-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val\text{-}v})$
.3 then let mk-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val}) = \mathit{val\text{-}v}$ in
.4 return is-$\mathbb{N}_1\,(\mathit{val})$
.5 else return false ;

This operation returns true if the input semantic numeric value denotes a natural number that is greater than 0.

1250.0 $\mathit{IsInt} : \mathit{SEM}`\mathit{VAL} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{IsInt}\,(\mathit{val\text{-}v}) \triangleq$
.2 if is-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val\text{-}v})$
.3 then let mk-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val}) = \mathit{val\text{-}v}$ in
.4 return is-$\mathbb{Z}\,(\mathit{val})$
.5 else return false ;

This operation returns true if the input semantic numeric value denotes an integer number.

1251.0 $\mathit{IsReal} : \mathit{SEM}`\mathit{VAL} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{IsReal}\,(\mathit{val\text{-}v}) \triangleq$
.2 if is-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val\text{-}v})$
.3 then let mk-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val}) = \mathit{val\text{-}v}$ in
.4 return is-$\mathbb{R}\,(\mathit{val})$
.5 else return false ;

This operation returns true if the input semantic numeric value denotes a real number.

1252.0 $\mathit{IsRat} : \mathit{SEM}`\mathit{VAL} \xrightarrow{o} \mathbb{B}$
.1 $\mathit{IsRat}\,(\mathit{val\text{-}v}) \triangleq$
.2 if is-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val\text{-}v})$
.3 then let mk-$\mathit{SEM}`\mathit{NUM}\,(\mathit{val}) = \mathit{val\text{-}v}$ in
.4 return is-$\mathbb{Q}\,(\mathit{val})$

.5 else return false

This operation returns true if the input semantic numeric value is a rational number.
functions

1253.0 $IsStmt : AS`Expr \mid AS`Stmt \to \mathbb{B}$
.1 $IsStmt\,(se) \triangleq$
.2 is-$AS`DefStmt\,(se) \lor$
.3 is-$AS`LetStmt\,(se) \lor$
.4 is-$AS`LetBeSTStmt\,(se) \lor$
.5 is-$AS`AssignStmt\,(se) \lor$
.6 is-$AS`SeqForLoopStmt\,(se) \lor$
.7 is-$AS`SetForLoopStmt\,(se) \lor$
.8 is-$AS`IndexForLoopStmt\,(se) \lor$
.9 is-$AS`WhileLoopStmt\,(se) \lor$
.10 is-$AS`CallStmt\,(se) \lor$
.11 is-$AS`ReturnStmt\,(se) \lor$
.12 is-$AS`IfStmt\,(se) \lor$
.13 is-$AS`CasesStmt\,(se) \lor$
.14 is-$AS`ErrorStmt\,(se) \lor$
.15 is-$AS`ExitStmt\,(se) \lor$
.16 is-$AS`AlwaysStmt\,(se) \lor$
.17 is-$AS`TrapStmt\,(se) \lor$
.18 is-$AS`RecTrapStmt\,(se) \lor$
.19 is-$AS`BlockStmt\,(se) \lor$
.20 is-$AS`NonDetStmt\,(se) \lor$
.21 is-$AS`IdentStmt\,(se) \lor$
.22 is-$AS`SpecificationStmt\,(se)$;

The *IsStmt* function simply checks whether the argument is one of the statements.

1254.0 $RestSeqVal : SEM`VAL^* \times \mathbb{N} \to SEM`VAL^*$
.1 $RestSeqVal\,(l, i) \triangleq$
.2 $[l\,(j) \mid j \in (\text{inds } l \setminus \{i\})]$;

1255.0 $Permute : SEM`VAL^* \to SEM`VAL^*$-set
.1 $Permute\,(l) \triangleq$
.2 cases l :
.3 $[]$,
.4 $[\text{-}] \to \{l\}$,
.5 others $\to \bigcup \{\{[l\,(i)] \curvearrowright j \mid j \in Permute\,(RestSeqVal\,(l, i))\} \mid$
.6 $i \in \text{inds } l\}$
.7 end

This operation returns the set with all permutations of the input sequence.
operations

1256.0 $ValSetToSeq : SEM`VAL\text{-set} \xrightarrow{o} SEM`VAL^*$
.1 $ValSetToSeq\,(set\text{-}sv) \triangleq$
.2 $(\mathsf{dcl}\ res\text{-}lv : SEM`VAL^* := [\,];$
.3 $\mathsf{for\ all}\ val\text{-}v \in set\text{-}sv$
.4 $\mathsf{do}\ res\text{-}lv := res\text{-}lv \curvearrowright [val\text{-}v];$
.5 $\mathsf{return}\ res\text{-}lv\ );$

This operation converts a set of semantic values into a sequence. This means that the sequence is ordered.

1257.0 $SetToSeq : SEM`VAL\text{-set} \xrightarrow{o} SEM`VAL^*$
.1 $SetToSeq\,(val\text{-}sv) \triangleq$
.2 $\mathsf{if}\ val\text{-}sv = \{\}$
.3 $\mathsf{then\ return}\ [\,]$
.4 $\mathsf{else\ let}\ elem \in val\text{-}sv\ \mathsf{be\ st}\ Min\,(elem, val\text{-}sv)\ \mathsf{in}$
.5 $\mathsf{return}\ [elem] \curvearrowright SetToSeq\,(val\text{-}sv \setminus \{elem\})\ ;$

This operation converts the input set with semantic numeric values into a sorted sequence of semantic numeric values.

1258.0 $SeqOfSetOf2SetOfSeqOf : (SEM`VAL \mid SEM`BlkEnv)\text{-set}^* \xrightarrow{o}$
.1 $(SEM`VAL \mid SEM`BlkEnv)^*\text{-set}$
.2 $SeqOfSetOf2SetOfSeqOf\,(seq\text{-}ls) \triangleq$
.3 $(\mathsf{dcl}\ res\text{-}s : (SEM`VAL \mid SEM`BlkEnv)^*\text{-set} := \{[\,]\},$
.4 $tmpres\text{-}s : (SEM`VAL \mid SEM`BlkEnv)^*\text{-set};$
.5 $\mathsf{for}\ tmp\text{-}s\ \mathsf{in}\ seq\text{-}ls$
.6 $\mathsf{do}\ (tmpres\text{-}s := \{\};$
.7 $\mathsf{for\ all}\ tmp\text{-}l \in res\text{-}s$
.8 $\mathsf{do\ for\ all}\ e \in tmp\text{-}s$
.9 $\mathsf{do}\ tmpres\text{-}s := tmpres\text{-}s \cup \{tmp\text{-}l \curvearrowright [e]\};$
.10 $res\text{-}s := tmpres\text{-}s);$
.11 $\mathsf{return}\ res\text{-}s\ );$

This operation converts a sequence of sets into a set of sequences.

1259.0 $Min : SEM`VAL \times SEM`VAL\text{-set} \xrightarrow{o} \mathbb{B}$
.1 $Min\,(n\text{-}v, set\text{-}sv) \triangleq$
.2 $\mathsf{if\ is\text{-}}SEM`NUM\,(n\text{-}v)$
.3 $\mathsf{then}\ (\mathsf{let\ mk\text{-}}SEM`NUM\,(n) = n\text{-}v\ \mathsf{in}$
.4 $\mathsf{for\ all}\ e\text{-}v \in set\text{-}sv$

.5 do if is-$SEM`NUM\,(e\text{-}v)$
.6 then let mk-$SEM`NUM\,(e) = e\text{-}v$ in
.7 (if $e < n$
.8 then return false)
.9 else error;
.10 return true)
.11 else error

This operation return true if the semantic numeric value n-v is less or equal to all the semantic numeric values in the input set.
functions

1260.0 $PreName : AS`Name \rightarrow AS`Name$

.1 $PreName\,($mk-$AS`Name\,(id\text{-}l, cid)) \triangleq$
.2 mk-$AS`Name\,([$"pre-"$] \frown$ hd $id\text{-}l] \frown$ tl $id\text{-}l, cid)$;

Extends the name of function or operation by an prefix "pre" , which relates to the pre-conditions.

1261.0 $PostName : AS`Name \rightarrow AS`Name$

.1 $PostName\,($mk-$AS`Name\,(id\text{-}l, cid)) \triangleq$
.2 mk-$AS`Name\,([$"$post$-"$\frown$ hd $id\text{-}l] \frown$ tl $id\text{-}l, cid)$;

Extends the name of function or operation by an prefix "post", which relates to the post-conditions.

1262.0 $ExtractId : AS`Name \rightarrow AS`Id$

.1 $ExtractId\,($mk-$AS`Name\,(name, \text{-})) \triangleq$
.2 $name\,($len $name)$;

1263.0 $ExtractName : AS`Name \rightarrow AS`Name$

.1 $ExtractName\,($mk-$AS`Name\,(name, cid)) \triangleq$
.2 let $id = name\,($len $name)$ in
.3 mk-$AS`Name\,([id], cid)$;

The following two functions are used to convert an AS'Name to a string. The first one takes a single variable with the name (either qualified or not), while the second one takes the module name as the first argument and the element name as the second argument. If the element name is qualified, then the invariant says that its module part should be equal to the module name given as the first argument.

1264.0 $SingleNameToString : AS`Name \to \mathsf{char}^*$

.1 $SingleNameToString\,(\mathsf{mk}\text{-}AS`Name\,(ids, \text{-})) \triangleq$

.2 $\quad \mathsf{if\ len}\ ids = 2$

.3 $\quad \mathsf{then\ hd}\ ids \curvearrowright \texttt{"`"} \curvearrowright \mathsf{hd\ tl}\ ids$

.4 $\quad \mathsf{else\ hd}\ ids;$

1265.0 $ConstructName : AS`Id \times CI`ContextId \to AS`Name$

.1 $ConstructName\,(id, cid) \triangleq$

.2 $\quad \mathsf{mk}\text{-}AS`Name\,([id], cid);$

1266.0 $ConstructDoubleName : AS`Name \times AS`Name \to AS`Name$

.1 $ConstructDoubleName\,(\mathsf{mk}\text{-}AS`Name\,([clmod] \curvearrowright \text{-}, \text{-}), \mathsf{mk}\text{-}AS`Name\,(\text{-} \curvearrowright [fnop], cid)) \triangleq$

.2 $\quad \mathsf{mk}\text{-}AS`Name\,([clmod, fnop], cid);$

This function extracts the identifier name of a construct from the input name.

1267.0 $EqualNames : AS`Name \times AS`Name \to \mathbb{B}$

.1 $EqualNames\,(\mathsf{mk}\text{-}AS`Name\,(name1, \text{-}), \mathsf{mk}\text{-}AS`Name\,(name2, \text{-})) \triangleq$

.2 $\quad name1 = name2$

end AUX

This function returns true if the two input names are equal.

Test Suite : rtinfo.ast
Module : AUX

Name	#Calls	Coverage
AUX`Min	**undefined**	**undefined**
AUX`IsInt	**undefined**	**undefined**
AUX`IsNat	**undefined**	**undefined**
AUX`IsRat	**undefined**	**undefined**
AUX`IsReal	**undefined**	**undefined**
AUX`IsStmt	**undefined**	**undefined**
AUX`Ceiling	**undefined**	**undefined**
AUX`ErrorOp	**undefined**	**undefined**
AUX`ExtComp	**undefined**	**undefined**
AUX`Permute	**undefined**	**undefined**
AUX`PreName	**undefined**	**undefined**
AUX`ErrorFct	**undefined**	**undefined**
AUX`IsNatOne	**undefined**	**undefined**
AUX`IsRecSel	**undefined**	**undefined**

Name	#Calls	Coverage
AUX‘MkBlkEnv	**undefined**	**undefined**
AUX‘PostName	**undefined**	**undefined**
AUX‘SetToSeq	**undefined**	**undefined**
AUX‘ExtractId	**undefined**	**undefined**
AUX‘IsTypeDef	**undefined**	**undefined**
AUX‘EqualNames	**undefined**	**undefined**
AUX‘RestSeqVal	**undefined**	**undefined**
AUX‘ExtractName	**undefined**	**undefined**
AUX‘ValSetToSeq	**undefined**	**undefined**
AUX‘ErrorEmptyOp	**undefined**	**undefined**
AUX‘LookUpRecSel	**undefined**	**undefined**
AUX‘CombineBlkEnv	**undefined**	**undefined**
AUX‘ConstructName	**undefined**	**undefined**
AUX‘MkEmptyBlkEnv	**undefined**	**undefined**
AUX‘ExtractTagName	**undefined**	**undefined**
AUX‘SingleNameToString	**undefined**	**undefined**
AUX‘ConstructDoubleName	**undefined**	**undefined**
AUX‘DistribCombineBlkEnv	**undefined**	**undefined**
AUX‘SeqOfSetOf2SetOfSeqOf	**undefined**	**undefined**
Total Coverage		**0%**

1.22 Closure Environment Creation

The module FREE contains the definition related to the closure environment.

module $FREE$

imports

1268.0 from AS all ,

1269.0 from CI all ,

1270.0 from AUX

1271.0 operations $MkBlkEnv : AS`Name \times SEM`VAL \times [AS`Type] \times SEM`Permission \xrightarrow{o}$

.1 $SEM`BlkEnv$ renamed $MkBlkEnv$;

.2 $CombineBlkEnv : SEM`BlkEnv \times SEM`BlkEnv \xrightarrow{o} SEM`BlkEnv$ renamed $CombineBlkEnv$;

.3 $MkEmptyBlkEnv : (SEM`Permission) \xrightarrow{o} SEM`BlkEnv$ renamed $MkEmptyBlkEnv$

,

1272.0 from PAT all ,

1273.0 from POS all ,

1274.0 from REP all ,

1275.0 from SEM all ,

1276.0 from $STKM$ all ,

1277.0 from $RTERR$ all ,

1278.0 from $GLOBAL$ all ,

1279.0 from $SCHDTP$ all ,

1280.0 from $INSTRTP$ all

exports

1281.0 operations $FreeInExpr : AS`Expr \times AS`Name\text{-set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL;$
.1 $IdentInPattern : AS`PatternBind \xrightarrow{o} AS`Name\text{-set};$
.2 $FreeMapToBlkEnv : AS`Name \xrightarrow{m} SEM`VAL \xrightarrow{o} SEM`BlkEnv$

definitions
operations

1282.0 $IdentInPattern : AS`PatternBind \xrightarrow{o} AS`Name\text{-set}$
.1 $IdentInPattern\,(pat) \triangleq$
.2 cases pat:
.3 mk-$AS`PatternName\,(id, \text{-}) \rightarrow$ return if $id = \text{nil}$
.4 then $\{\}$
.5 else $\{id\}$,
.6 mk-$AS`MatchVal\,(\text{-}, \text{-}) \rightarrow$ return $\{\}$,
.7 mk-$AS`SetEnumPattern\,(Elems, \text{-}) \rightarrow$ return $\bigcup \{IdentInPattern\,(Elems\,(p)) \mid$
.8 $p \in$ inds $Elems\}$,
.9 mk-$AS`SetUnionPattern\,(lp, rp, \text{-}) \rightarrow$ return $IdentInPattern\,(lp) \cup$
.10 $IdentInPattern\,(rp)$,
.11 mk-$AS`SeqEnumPattern\,(els, \text{-}) \rightarrow$ return $\bigcup \{IdentInPattern\,(els\,(i)) \mid$
.12 $i \in$ inds $els\}$,
.13 mk-$AS`SeqConcPattern\,(lp, rp, \text{-}) \rightarrow$ return $IdentInPattern\,(lp) \cup$
.14 $IdentInPattern\,(rp)$,
.15 mk-$AS`RecordPattern\,(\text{-}, fields, \text{-}) \rightarrow$ return $\bigcup \{IdentInPattern\,(fields\,(i)) \mid$
.16 $i \in$ inds $fields\}$,
.17 mk-$AS`TuplePattern\,(fields, \text{-}) \rightarrow$ return $\bigcup \{IdentInPattern\,(fields\,(i)) \mid$
.18 $i \in$ inds $fields\}$,
.19 mk-$AS`SetBind\,(pat, \text{-}, \text{-}) \rightarrow IdentInPattern(pat)$,
.20 mk-$AS`TypeBind\,(pat, \text{-}, \text{-}) \rightarrow IdentInPattern(pat)$,
.21 others $\rightarrow$ error
.22 end;

The operation *IdentInPattern* is used to extract all the pattern identifiers from either a pattern or a bind.

```
1283.0  IdentInBind : AS`Bind -o-> AS`Name-set
    .1  IdentInBind (bind) △
    .2    IdentInPattern(bind.pat) ;
```

```
1284.0  IdentInMultBindSeq : AS`MultBind* -o-> AS`Name-set
    .1  IdentInMultBindSeq (bind-l) △
    .2    (dcl id-s : AS`Name-set := {};
    .3     for bind in bind-l
    .4     do for pat-p in bind.pat
    .5         do id-s := id-s ∪ IdentInPattern (pat-p);
    .6     return id-s );
```

```
1285.0  FreeInBind : AS`Bind × AS`Name-set -o-> AS`Name -m-> SEM`VAL
    .1  FreeInBind (bind, id-s) △
    .2    if is-AS`SetBind (bind)
    .3    then let mk-AS`SetBind (-, exp-e, -) = bind in
    .4           FreeInExpr(exp-e, id-s)
    .5    else return {↦} ;
```

The operation *FreeInBind* returns the free variable mapping for the expression in a set bind, or an empty map otherwise.

```
1286.0  FreeInMultBindSeq : AS`MultBind* × AS`Name-set -o-> AS`Name -m-> SEM`VAL
    .1  FreeInMultBindSeq (bind-l, id-s) △
    .2    (dcl newid-s : AS`Name-set := id-s,
    .3         res-m : AS`Name -m-> SEM`VAL := {↦};
    .4     for bind in bind-l
    .5     do  cases  bind:
    .6           mk-AS`MultSetBind (pat-l, set-e, -) → (for pat in pat-l
    .7                                                  do newid-s := newid-s ∪
    .8                                                                IdentInPattern (pat);
    .9                                                  let tmp-m = FreeInExpr (set-e, newid-s) in
    .10                                                 (res-m := res-m ⊎ tmp-m;
    .11                                                  newid-s := newid-s∪dom tmp-m)),
    .12          mk-AS`MultTypeBind (pat-l, -, -) → for pat in pat-l
    .13                                             do newid-s := newid-s ∪
    .14                                                           IdentInPattern (pat)
    .15        end ;
    .16    return res-m );
```

The operation *FreeInMultBindSeq* returns the free variable mapping for a sequence of multi binds.

1.22.1 Expressions

1287.0 $FreeInExpr : AS‘Expr \times AS‘Name\text{-set} \xrightarrow{o} AS‘Name \xrightarrow{m} SEM‘VAL$
.1 $FreeInExpr\,(expr, id\text{-}s) \triangleq$
.2 cases true:
.3 (is-AS‘DefExpr (expr)) → FreeInDefExpr(expr, id-s) ,
.4 (is-AS‘LetExpr (expr)) → FreeInLetExpr(expr, id-s) ,
.5 (is-AS‘LetBeSTExpr (expr)) → FreeInLetBeSTExpr(expr, id-s) ,
.6 (is-AS‘IfExpr (expr)) → FreeInIfExpr(expr, id-s) ,
.7 (is-AS‘CasesExpr (expr)) → FreeInCasesExpr(expr, id-s) ,
.8 (is-AS‘PrefixExpr (expr)) → FreeInPrefixExpr(expr, id-s) ,
.9 (is-AS‘MapInverseExpr (expr)) → FreeInMapInverseExpr(expr, id-s) ,
.10 (is-AS‘BinaryExpr (expr)) → FreeInBinaryExpr(expr, id-s) ,
.11 (is-AS‘AllOrExistsExpr (expr)) → FreeInAllOrExistsExpr(expr, id-s) ,
.12 (is-AS‘ExistsUniqueExpr (expr)) → FreeInExistsUniqueExpr(expr, id-s) ,
.13 (is-AS‘SetEnumerationExpr (expr)) → FreeInSetEnumerationExpr(expr, id-s) ,
.14 (is-AS‘SetComprehensionExpr (expr)) → FreeInSetComprehensionExpr(expr, id-s),
.15 (is-AS‘SetRangeExpr (expr)) → FreeInSetRangeExpr(expr, id-s) ,
.16 (is-AS‘SeqEnumerationExpr (expr)) → FreeInSeqEnumerationExpr(expr, id-s) ,
.17 (is-AS‘SeqComprehensionExpr (expr)) → FreeInSeqComprehensionExpr(expr, id-s),
.18 (is-AS‘SubSequenceExpr (expr)) → FreeInSubSequenceExpr(expr, id-s) ,
.19 (is-AS‘SeqModifyMapOverrideExpr (expr)) → FreeInSeqModifyMapOverrideExpr(expr, id-s),
.20 (is-AS‘MapEnumerationExpr (expr)) → FreeInMapEnumerationExpr(expr, id-s),
.21 (is-AS‘MapComprehensionExpr (expr)) → FreeInMapComprehensionExpr(expr, id-s),
.22 (is-AS‘TupleConstructorExpr (expr)) → FreeInTupleConstructorExpr(expr, id-s),
.23 (is-AS‘RecordConstructorExpr (expr)) → FreeInRecordConstructorExpr(expr, id-s),
.24 (is-AS‘RecordModifierExpr (expr)) → FreeInRecordModifierExpr(expr, id-s) ,
.25 (is-AS‘TokenConstructorExpr (expr)) → FreeInTokenConstructorExpr(expr, id-s),
.26 (is-AS‘TupleSelectExpr (expr)) → FreeInTupleSelectExpr(expr, id-s) ,
.27 (is-AS‘TypeJudgementExpr (expr)) → FreeInTypeJudgementExpr(expr, id-s) ,
.28 (is-AS‘PreConditionApplyExpr (expr)) → FreeInPreConditionApplyExpr(expr, id-s),
.29 (is-AS‘ApplyExpr (expr)) → FreeInApplyExpr(expr, id-s) ,
.30 (is-AS‘LambdaExpr (expr)) → FreeInLambdaExpr(expr, id-s) ,
.31 (is-AS‘FieldSelectExpr (expr)) → FreeInFieldSelectExpr(expr, id-s) ,
.32 (is-AS‘FctTypeInstExpr (expr)) → FreeInFctTypeInstExpr(expr, id-s) ,
.33 (is-AS‘IsExpr (expr)) → FreeInIsExpr(expr, id-s) ,
.34 (is-AS‘IotaExpr (expr)) → FreeInIotaExpr(expr, id-s) ,
.35 (is-AS‘BoolLit (expr)),
.36 (is-AS‘CharLit (expr)),
.37 (is-AS‘TextLit (expr)),
.38 (is-AS‘QuoteLit (expr)),
.39 (is-AS‘RealLit (expr)),
.40 (is-AS‘NilLit (expr)) → return {↦},
.41 (is-AS‘Name (expr)) → FreeInName(expr, id-s) ,
.42 (is-AS‘UndefinedExpr (expr)) → return {↦},
.43 (is-AS‘BracketedExpr (expr)) → FreeInBracketedExpr(expr, id-s) ,
.44 others → return {↦}
.45 end;

The operation *FreeInExpr* is the entry point for the calculation of the free variable mapping. The other operations call this operation recursively. The resulting map of the operation must be converted to a block environment (*BlkEnv*) before it can be used.

Local Binding Expressions

```
1288.0 FreeInDefExpr : AS`DefExpr × AS`Name-set −o→ AS`Name −m→ SEM`VAL
    .1 FreeInDefExpr (mk-AS`DefExpr (def-l, in-e, -), id-s) △
    .2   (dcl res-m : AS`Name −m→ SEM`VAL := {↦},
    .3        newid-s : AS`Name-set := id-s;
    .4    for mk- (pat-p, val-e) in def-l
    .5    do (res-m := res-m ⊎m FreeInExpr (val-e, id-s);
    .6         newid-s := newid-s ∪ IdentInPattern (pat-p));
    .7     newid-s := newid-s ∪ dom res-m;
    .8     res-m := res-m ⊎m FreeInExpr (in-e, newid-s);
    .9     return res-m );
```

```
1289.0 FreeInLetExpr : AS`LetExpr × AS`Name-set −o→ AS`Name −m→ SEM`VAL
    .1 FreeInLetExpr (mk-AS`LetExpr (localdef, in-e, -), id-s) △
    .2   (dcl res-m : AS`Name −m→ SEM`VAL := {↦},
    .3        newid-s : AS`Name-set := id-s;
    .4    for ldef in localdef
    .5    do cases true:
    .6         (is-AS`ExplFnDef (ldef)) → newid-s := newid-s ∪ {ldef.nm},
    .7         (is-AS`ImplFnDef (ldef)) → newid-s := newid-s ∪ {ldef.nm},
    .8         (is-AS`ValueDef (ldef)) → newid-s := newid-s ∪ IdentInPattern (ldef.pat)
    .9       end;
   .10    for ldef in localdef
   .11    do cases true:
   .12         (is-AS`ExplFnDef (ldef)) → let tmp-m = FreeInFnDef (ldef, newid-s) in
   .13                                     (res-m := res-m ⊎m tmp-m;
   .14                                      newid-s := newid-s ∪ dom tmp-m),
   .15         (is-AS`ImplFnDef (ldef)) → let tmp-m = FreeInFnDef (ldef, newid-s) in
   .16                                     (res-m := res-m ⊎m tmp-m;
   .17                                      newid-s := newid-s ∪ dom tmp-m),
   .18         (is-AS`ValueDef (ldef)) → let tmp-m = FreeInExpr (ldef.val, newid-s) in
   .19                                    (res-m := res-m ⊎m tmp-m;
   .20                                     newid-s := newid-s ∪ dom tmp-m)
   .21       end ;
   .22     res-m := res-m ⊎m FreeInExpr (in-e, newid-s);
   .23     return res-m );
```

```
1290.0 FreeInFnDef : AS`FnDef × AS`Name-set →(o) AS`Name →(m) SEM`VAL
   .1  FreeInFnDef (fn, id-s) ≜
   .2    (dcl res-m : AS`Name →(m) SEM`VAL,
   .3         newid-s : AS`Name-set := id-s;
   .4     if is-AS`ExplFnDef (fn)
   .5     then let mk-AS`ExplFnDef (-, -, -, parms, body, Pre-e, Post-e, access,
   .6                                          static, -) =
   .7                    fn in
   .8           (for pat-l in parms
   .9            do for pat in pat-l
   .10               do newid-s := newid-s ∪ IdentInPattern (pat);
   .11           res-m := if body.body = NOTYETSPEC
   .12                    then {↦}
   .13                    else FreeInExpr (body.body, newid-s);
   .14           newid-s := newid-s ∪ dom res-m;
   .15           let tmp-m = FreeInPreExpr (Pre-e, newid-s) in
   .16           (res-m := res-m ⊎ tmp-m;
   .17            newid-s := newid-s ∪ dom tmp-m);
   .18           let tmp-m = FreeInPostExpr (Post-e, newid-s) in
   .19           res-m := res-m ⊎ tmp-m;
   .20           return res-m )
   .21    else let mk-AS`ImplFnDef (-, -, partps, -, Pre-e, Post-e, access, static,
   .22                                          -) =
   .23                   fn in
   .24          (for mk-AS`PatTypePair (pat-l, -, -) in partps
   .25           do for pat in pat-l
   .26              do newid-s := newid-s ∪ IdentInPattern (pat);
   .27          res-m := FreeInPreExpr (Pre-e, newid-s);
   .28          newid-s := newid-s ∪ dom res-m;
   .29          let tmp-m = FreeInPostExpr (Post-e, newid-s) in
   .30          res-m := res-m ⊎ tmp-m;
   .31          return res-m ));
```

```
1291.0 FreeInPreExpr : [AS`Expr] × AS`Name-set →(o) AS`Name →(m) SEM`VAL
   .1  FreeInPreExpr (expr, id-s) ≜
   .2    if expr = nil
   .3    then return {↦}
   .4    else FreeInExpr(expr, id-s) ;
```

```
1292.0 FreeInPostExpr : [AS`Expr] × AS`Name-set →(o) AS`Name →(m) SEM`VAL
   .1  FreeInPostExpr (expr, id-s) ≜
   .2    if expr = nil
   .3    then return {↦}
   .4    else FreeInExpr(expr, id-s) ;
```

1293.0 $FreeInLetBeSTExpr : AS`LetBeSTExpr \times AS`Name\text{-set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInLetBeSTExpr\,(\text{mk-}AS`LetBeSTExpr\,(lhs, st\text{-}e, in\text{-}e, \text{-}), id\text{-}s) \triangleq$
.2 $\quad \text{let } patid\text{-}s = IdentInBind\,(lhs),$
.3 $\qquad expid\text{-}m = FreeInBind\,(lhs, id\text{-}s) \text{ in}$
.4 $\quad \text{let } stid\text{-}m = FreeInExpr\,(st\text{-}e, id\text{-}s \cup patid\text{-}s \cup \text{dom } expid\text{-}m) \text{ in}$
.5 $\quad \text{let } newid\text{-}s = \bigcup\{id\text{-}s, \text{dom } expid\text{-}m, \text{dom } stid\text{-}m, patid\text{-}s\} \text{ in}$
.6 $\quad \text{let } tmp\text{-}m = FreeInExpr\,(in\text{-}e, newid\text{-}s) \text{ in}$
.7 $\quad \text{let } res\text{-}m = \text{merge } \{expid\text{-}m, stid\text{-}m, tmp\text{-}m\} \text{ in}$
.8 $\quad \text{return } res\text{-}m;$

Conditional Expressions

1294.0 $FreeInIfExpr : AS`IfExpr \times AS`Name\text{-set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInIfExpr\,(\text{mk-}AS`IfExpr\,(test\text{-}e, cons\text{-}e, elif, altn\text{-}e, \text{-}), id\text{-}s) \triangleq$
.2 $\quad (\text{dcl } res\text{-}m : AS`Name \xrightarrow{m} SEM`VAL,$
.3 $\qquad newid\text{-}s : AS`Name\text{-set} := id\text{-}s;$
.4 $\quad res\text{-}m := FreeInExpr\,(test\text{-}e, newid\text{-}s);$
.5 $\quad newid\text{-}s := newid\text{-}s \cup \text{dom } res\text{-}m;$
.6 $\quad \text{let } tmp\text{-}m = FreeInExpr\,(cons\text{-}e, newid\text{-}s) \text{ in}$
.7 $\quad (res\text{-}m := res\text{-}m \uplus tmp\text{-}m;$
.8 $\quad newid\text{-}s := newid\text{-}s \cup \text{dom } tmp\text{-}m);$
.9 $\quad \text{let } tmp\text{-}m = FreeInElseifExpr\,(elif, newid\text{-}s) \text{ in}$
.10 $\quad (res\text{-}m := res\text{-}m \uplus tmp\text{-}m;$
.11 $\quad newid\text{-}s := newid\text{-}s \cup \text{dom } tmp\text{-}m);$
.12 $\quad res\text{-}m := FreeInExpr\,(altn\text{-}e, newid\text{-}s);$
.13 $\quad \text{return } res\text{-}m\ );$

1295.0 $FreeInElseifExpr : AS`ElseifExpr^* \times AS`Name\text{-set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInElseifExpr\,(elif\text{-}l, id\text{-}s) \triangleq$
.2 $\quad (\text{dcl } res\text{-}m : AS`Name \xrightarrow{m} SEM`VAL := \{\mapsto\},$
.3 $\qquad newid\text{-}s : AS`Name\text{-set} := id\text{-}s;$
.4 $\quad \text{for mk-}AS`ElseifExpr\,(test\text{-}e, cons\text{-}e, \text{-}) \text{ in } elif\text{-}l$
.5 $\quad \text{do } (\text{let } tmp\text{-}m = FreeInExpr\,(test\text{-}e, newid\text{-}s) \text{ in}$
.6 $\qquad (res\text{-}m := res\text{-}m \uplus tmp\text{-}m;$
.7 $\qquad newid\text{-}s := newid\text{-}s \cup \text{dom } tmp\text{-}m);$
.8 $\qquad \text{let } tmp\text{-}m = FreeInExpr\,(cons\text{-}e, newid\text{-}s) \text{ in}$
.9 $\qquad (res\text{-}m := res\text{-}m \uplus tmp\text{-}m;$
.10 $\qquad newid\text{-}s := newid\text{-}s \cup \text{dom } tmp\text{-}m));$
.11 $\quad \text{return } res\text{-}m\ );$

1296.0 $FreeInCasesExpr : AS`CasesExpr \times AS`Name\text{-set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInCasesExpr\,(\text{mk-}AS`CasesExpr\,(sel\text{-}e, altns\text{-}l, others\text{-}e, \text{-}), id\text{-}s) \triangleq$
.2 $\quad (\text{dcl } alt\text{-}l : AS`CaseAltn^* := altns\text{-}l,$

.3 $\quad\quad pat\text{-}lp : AS`Pattern^*,$
.4 $\quad\quad newid\text{-}s : AS`Name\text{-}\mathsf{set} := id\text{-}s,$
.5 $\quad\quad res\text{-}m : AS`Name \xrightarrow{m} SEM`VAL := \{\mapsto\};$
.6 $\quad \mathsf{let}\ sel\text{-}m = FreeInExpr\,(sel\text{-}e, newid\text{-}s)\ \mathsf{in}$
.7 $\quad (res\text{-}m := sel\text{-}m;$
.8 $\quad\ newid\text{-}s := newid\text{-}s \cup \mathsf{dom}\ sel\text{-}m;$
.9 $\quad\ \mathsf{while}\ alt\text{-}l \neq []$
.10 $\quad\ \mathsf{do\ let\ mk\text{-}}AS`CaseAltn\,(match\text{-}lp, body\text{-}e, \text{-}) = \mathsf{hd}\ alt\text{-}l\ \mathsf{in}$
.11 $\quad\quad\ (pat\text{-}lp := match\text{-}lp;$
.12 $\quad\quad\quad \mathsf{while}\ pat\text{-}lp \neq []$
.13 $\quad\quad\quad \mathsf{do\ let}\ pat\text{-}p = \mathsf{hd}\ pat\text{-}lp\ \mathsf{in}$
.14 $\quad\quad\quad\quad (newid\text{-}s := newid\text{-}s \cup IdentInPattern\,(pat\text{-}p);$
.15 $\quad\quad\quad\quad\ pat\text{-}lp := \mathsf{tl}\ pat\text{-}lp);$
.16 $\quad\quad\quad \mathsf{let}\ tmp\text{-}m = FreeInExpr\,(body\text{-}e, newid\text{-}s)\ \mathsf{in}$
.17 $\quad\quad\quad (res\text{-}m := res\text{-}m \overset{m}{\uplus} tmp\text{-}m;$
.18 $\quad\quad\quad\ newid\text{-}s := newid\text{-}s \cup \mathsf{dom}\ tmp\text{-}m);$
.19 $\quad\quad\quad alt\text{-}l := \mathsf{tl}\ alt\text{-}l);$
.20 $\quad\ \mathsf{let}\ tmp\text{-}m = FreeInExpr\,(others\text{-}e, newid\text{-}s)\ \mathsf{in}$
.21 $\quad\ res\text{-}m := res\text{-}m \overset{m}{\uplus} tmp\text{-}m;$
.22 $\quad\ \mathsf{return}\ res\text{-}m\ ));$

Unary Expressions

1297.0 $FreeInPrefixExpr : AS`PrefixExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInPrefixExpr\,(\mathsf{mk\text{-}}AS`PrefixExpr\,(\text{-}, arg\text{-}e, \text{-}), id\text{-}s) \triangleq$
.2 $\quad FreeInExpr(arg\text{-}e, id\text{-}s)\ ;$

1298.0 $FreeInMapInverseExpr : AS`MapInverseExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInMapInverseExpr\,(\mathsf{mk\text{-}}AS`MapInverseExpr\,(op, \text{-}), id\text{-}s) \triangleq$
.2 $\quad FreeInExpr(op, id\text{-}s)\ ;$

Quantified Expressions

1299.0 $FreeInBinaryExpr : AS`BinaryExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInBinaryExpr\,(\mathsf{mk\text{-}}AS`BinaryExpr\,(left\text{-}e, \text{-}, right\text{-}e, \text{-}), id\text{-}s) \triangleq$
.2 $\quad \mathsf{let}\ tmp\text{-}m = FreeInExpr\,(left\text{-}e, id\text{-}s)\ \mathsf{in}$
.3 $\quad \mathsf{let}\ res\text{-}m = FreeInExpr\,(right\text{-}e, id\text{-}s \cup \mathsf{dom}\ tmp\text{-}m)\ \mathsf{in}$
.4 $\quad \mathsf{return}\ res\text{-}m \overset{m}{\uplus} tmp\text{-}m;$

1300.0 $\mathit{FreeInAllOrExistsExpr} : \mathit{AS`AllOrExistsExpr} \times \mathit{AS`Name}\text{-}\mathsf{set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInAllOrExistsExpr}\,(\mathsf{mk}\text{-}\mathit{AS`AllOrExistsExpr}\,(\text{-}, \mathit{bind\text{-}l}, \mathit{pred\text{-}e}, \text{-}), \mathit{id\text{-}s}) \triangleq$

.2 $(\mathsf{dcl}\ \mathit{res\text{-}m} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL},$

.3 $\quad \mathit{newid\text{-}s} : \mathit{AS`Name}\text{-}\mathsf{set};$

.4 $\mathit{res\text{-}m} := \mathit{FreeInMultBindSeq}\,(\mathit{bind\text{-}l}, \mathit{id\text{-}s});$

.5 $\mathit{newid\text{-}s} := \bigcup \{\mathsf{dom}\ \mathit{res\text{-}m}, \mathit{id\text{-}s}, \mathit{IdentInMultBindSeq}\,(\mathit{bind\text{-}l})\};$

.6 $\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus \mathit{FreeInExpr}\,(\mathit{pred\text{-}e}, \mathit{newid\text{-}s});$

.7 $\mathsf{return}\ \mathit{res\text{-}m}\ );$

1301.0 $\mathit{FreeInExistsUniqueExpr} : \mathit{AS`ExistsUniqueExpr} \times \mathit{AS`Name}\text{-}\mathsf{set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInExistsUniqueExpr}\,(\mathsf{mk}\text{-}\mathit{AS`ExistsUniqueExpr}\,(\mathit{bind}, \mathit{pred\text{-}e}, \text{-}), \mathit{id\text{-}s}) \triangleq$

.2 $\mathsf{let}\ \mathit{patid\text{-}s} = \mathit{IdentInBind}\,(\mathit{bind}),$

.3 $\quad \mathit{expid\text{-}m} = \mathit{FreeInBind}\,(\mathit{bind}, \mathit{id\text{-}s})\ \mathsf{in}$

.4 $\mathsf{let}\ \mathit{newid\text{-}s} = \bigcup \{\mathit{id\text{-}s}, \mathit{patid\text{-}s}, \mathsf{dom}\ \mathit{expid\text{-}m}\}\ \mathsf{in}$

.5 $\mathsf{let}\ \mathit{tmp\text{-}m} = \mathit{FreeInExpr}\,(\mathit{pred\text{-}e}, \mathit{newid\text{-}s})\ \mathsf{in}$

.6 $\mathsf{let}\ \mathit{res\text{-}m} = \mathit{expid\text{-}m} \uplus \mathit{tmp\text{-}m}\ \mathsf{in}$

.7 $\mathsf{return}\ \mathit{res\text{-}m};$

Set Expressions

1302.0 $\mathit{FreeInSetEnumerationExpr} : \mathit{AS`SetEnumerationExpr} \times \mathit{AS`Name}\text{-}\mathsf{set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInSetEnumerationExpr}\,(\mathsf{mk}\text{-}\mathit{AS`SetEnumerationExpr}\,(\mathit{elm\text{-}le}, \text{-}), \mathit{id\text{-}s}) \triangleq$

.2 $(\mathsf{dcl}\ \mathit{res\text{-}m} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL} := \{\mapsto\},$

.3 $\quad \mathit{newid\text{-}s} : \mathit{AS`Name}\text{-}\mathsf{set} := \mathit{id\text{-}s};$

.4 $\mathsf{for}\ \mathit{elm\text{-}e}\ \mathsf{in}\ \mathit{elm\text{-}le}$

.5 $\mathsf{do}\ \mathsf{let}\ \mathit{tmp\text{-}m} = \mathit{FreeInExpr}\,(\mathit{elm\text{-}e}, \mathit{newid\text{-}s})\ \mathsf{in}$

.6 $\quad (\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus \mathit{tmp\text{-}m};$

.7 $\quad \mathit{newid\text{-}s} := \mathit{newid\text{-}s} \cup \mathsf{dom}\ \mathit{tmp\text{-}m});$

.8 $\mathsf{return}\ \mathit{res\text{-}m}\ );$

1303.0 $\mathit{FreeInSetComprehensionExpr} : \mathit{AS`SetComprehensionExpr} \times \mathit{AS`Name}\text{-}\mathsf{set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInSetComprehensionExpr}\,(\mathsf{mk}\text{-}\mathit{AS`SetComprehensionExpr}\,(\mathit{elem\text{-}e}, \mathit{bind\text{-}l},$

.2 $\qquad \mathit{pred\text{-}e}, \text{-}), \mathit{id\text{-}s}) \triangleq$

.3 $(\mathsf{dcl}\ \mathit{res\text{-}m} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL},$

.4 $\quad \mathit{newid\text{-}s} : \mathit{AS`Name}\text{-}\mathsf{set};$

.5 $res\text{-}m := FreeInMultBindSeq\,(bind\text{-}l, id\text{-}s);$

.6 $newid\text{-}s := \bigcup\{id\text{-}s, \mathsf{dom}\ res\text{-}m, IdentInMultBindSeq\,(bind\text{-}l)\};$

.7 $\mathsf{let}\ tmp\text{-}m = FreeInExpr\,(elem\text{-}e, newid\text{-}s)\ \mathsf{in}$

.8 $(res\text{-}m := res\text{-}m \overset{m}{\cup} tmp\text{-}m;$

.9 $\ newid\text{-}s := newid\text{-}s \cup \mathsf{dom}\ tmp\text{-}m);$

.10 $res\text{-}m := res\text{-}m \overset{m}{\cup} FreeInExpr\,(pred\text{-}e, newid\text{-}s);$

.11 $\mathsf{return}\ res\text{-}m\ );$

1304.0 $FreeInSetRangeExpr : AS`SetRangeExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$

.1 $FreeInSetRangeExpr\,(\mathsf{mk}\text{-}AS`SetRangeExpr\,(lb\text{-}e, ub\text{-}e, \text{-}), id\text{-}s) \triangleq$

.2 $(\mathsf{dcl}\ res\text{-}m : AS`Name \xrightarrow{m} SEM`VAL,$

.3 $\quad newid\text{-}s : AS`Name\text{-}\mathsf{set};$

.4 $res\text{-}m := FreeInExpr\,(lb\text{-}e, id\text{-}s);$

.5 $newid\text{-}s := id\text{-}s \cup \mathsf{dom}\ res\text{-}m;$

.6 $res\text{-}m := res\text{-}m \overset{m}{\cup} FreeInExpr\,(ub\text{-}e, newid\text{-}s);$

.7 $\mathsf{return}\ res\text{-}m\ );$

Sequence Expressions

1305.0 $FreeInSeqEnumerationExpr : AS`SeqEnumerationExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$

.1 $FreeInSeqEnumerationExpr\,(\mathsf{mk}\text{-}AS`SeqEnumerationExpr\,(els\text{-}l, \text{-}), id\text{-}s) \triangleq$

.2 $(\mathsf{dcl}\ res\text{-}m : AS`Name \xrightarrow{m} SEM`VAL := \{\mapsto\},$

.3 $\quad newid\text{-}s : AS`Name\text{-}\mathsf{set} := id\text{-}s;$

.4 $\mathsf{for}\ elm\ \mathsf{in}\ els\text{-}l$

.5 $\mathsf{do}\ \mathsf{let}\ tmp\text{-}m = FreeInExpr\,(elm, newid\text{-}s)\ \mathsf{in}$

.6 $\quad (res\text{-}m := res\text{-}m \overset{m}{\cup} tmp\text{-}m;$

.7 $\quad\ newid\text{-}s := newid\text{-}s \cup \mathsf{dom}\ tmp\text{-}m);$

.8 $\mathsf{return}\ res\text{-}m\ );$

1306.0 $FreeInSeqComprehensionExpr : AS`SeqComprehensionExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$

.1 $FreeInSeqComprehensionExpr\,(\mathsf{mk}\text{-}AS`SeqComprehensionExpr\,(elem\text{-}e, bind,$

.2 $\qquad pred\text{-}e, \text{-}), id\text{-}s) \triangleq$

.3 $(\mathsf{dcl}\ res\text{-}m : AS`Name \xrightarrow{m} SEM`VAL,$

.4 $\quad newid\text{-}s : AS`Name\text{-}\mathsf{set};$

.5 $res\text{-}m := FreeInBind\,(bind, id\text{-}s);$

.6 $newid\text{-}s := \bigcup\{id\text{-}s, \mathsf{dom}\ res\text{-}m, IdentInBind\,(bind)\};$

.7 $\mathsf{let}\ tmp\text{-}m = FreeInExpr\,(elem\text{-}e, newid\text{-}s)\ \mathsf{in}$

.8 $(res\text{-}m := res\text{-}m \overset{m}{\cup} tmp\text{-}m;$

.9 $\ newid\text{-}s := newid\text{-}s \cup \mathsf{dom}\ tmp\text{-}m);$

.10 $res\text{-}m := res\text{-}m \overset{m}{\cup} FreeInExpr\,(pred\text{-}e, newid\text{-}s);$

.11 $\mathsf{return}\ res\text{-}m\ );$

1307.0 $\mathit{FreeInSubSequenceExpr} : \mathit{AS`SubSequenceExpr} \times \mathit{AS`Name}\text{-set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$
.1 $\mathit{FreeInSubSequenceExpr}\,(\text{mk-}\mathit{AS`SubSequenceExpr}\,(\mathit{seq\text{-}e}, \mathit{from\text{-}e}, \mathit{to\text{-}e}, \text{-}), \mathit{id\text{-}s}) \triangleq$
.2 (dcl $\mathit{res\text{-}m} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL} := \{\mapsto\}$,
.3 $\mathit{newid\text{-}s} : \mathit{AS`Name}\text{-set} := \mathit{id\text{-}s}$;
.4 let $\mathit{tmp\text{-}m} = \mathit{FreeInExpr}\,(\mathit{seq\text{-}e}, \mathit{newid\text{-}s})$ in
.5 $(\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus \mathit{tmp\text{-}m}$;
.6 $\mathit{newid\text{-}s} := \mathit{newid\text{-}s} \cup \mathrm{dom}\ \mathit{tmp\text{-}m})$;
.7 let $\mathit{tmp\text{-}m} = \mathit{FreeInExpr}\,(\mathit{from\text{-}e}, \mathit{newid\text{-}s})$ in
.8 $(\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus \mathit{tmp\text{-}m}$;
.9 $\mathit{newid\text{-}s} := \mathit{newid\text{-}s} \cup \mathrm{dom}\ \mathit{tmp\text{-}m})$;
.10 $\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus \mathit{FreeInExpr}\,(\mathit{to\text{-}e}, \mathit{newid\text{-}s})$;
.11 return $\mathit{res\text{-}m}$);

1308.0 $\mathit{FreeInSeqModifyMapOverrideExpr} : \mathit{AS`SeqModifyMapOverrideExpr} \times \mathit{AS`Name}\text{-set} \xrightarrow{o}$
.1 $\mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$
.2 $\mathit{FreeInSeqModifyMapOverrideExpr}\,(\text{mk-}\mathit{AS`SeqModifyMapOverrideExpr}\,(\mathit{seqmap\text{-}e},$
.3 $\mathit{map\text{-}e}, \text{-}), \mathit{id\text{-}s}) \triangleq$
.4 let $\mathit{tmp\text{-}m} = \mathit{FreeInExpr}\,(\mathit{seqmap\text{-}e}, \mathit{id\text{-}s})$ in
.5 let $\mathit{res\text{-}m} = \mathit{FreeInExpr}\,(\mathit{map\text{-}e}, \mathit{id\text{-}s} \cup \mathrm{dom}\ \mathit{tmp\text{-}m})$ in
.6 return $\mathit{res\text{-}m} \uplus \mathit{tmp\text{-}m}$;

1309.0 $\mathit{FreeInMapEnumerationExpr} : \mathit{AS`MapEnumerationExpr} \times \mathit{AS`Name}\text{-set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m}$ $\mathit{SEM`VAL}$
.1 $\mathit{FreeInMapEnumerationExpr}\,(\text{mk-}\mathit{AS`MapEnumerationExpr}\,(\mathit{els\text{-}l}, \text{-}), \mathit{id\text{-}s}) \triangleq$
.2 (dcl $\mathit{res\text{-}m} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL} := \{\mapsto\}$,
.3 $\mathit{newid\text{-}s} : \mathit{AS`Name}\text{-set} := \mathit{id\text{-}s}$;
.4 for mk-$\mathit{AS`Maplet}\,(\mathit{dom\text{-}e}, \mathit{rng\text{-}e}, \text{-})$ in $\mathit{els\text{-}l}$
.5 do (let $\mathit{tmp\text{-}m} = \mathit{FreeInExpr}\,(\mathit{dom\text{-}e}, \mathit{newid\text{-}s})$ in
.6 $(\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus \mathit{tmp\text{-}m}$;
.7 $\mathit{newid\text{-}s} := \mathit{newid\text{-}s} \cup \mathrm{dom}\ \mathit{tmp\text{-}m})$;
.8 let $\mathit{tmp\text{-}m} = \mathit{FreeInExpr}\,(\mathit{rng\text{-}e}, \mathit{newid\text{-}s})$ in
.9 $(\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus \mathit{tmp\text{-}m}$;
.10 $\mathit{newid\text{-}s} := \mathit{newid\text{-}s} \cup \mathrm{dom}\ \mathit{tmp\text{-}m}))$;
.11 return $\mathit{res\text{-}m}$);

Map Expressions

1310.0 $\mathit{FreeInMapComprehensionExpr} : \mathit{AS`MapComprehensionExpr} \times \mathit{AS`Name}\text{-set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInMapComprehensionExpr}\ (\text{mk-}\mathit{AS`MapComprehensionExpr}\ (\mathit{elem}, \mathit{bind\text{-}l},$
.2 $\mathit{pred\text{-}e}, \text{-}), \mathit{id\text{-}s}) \triangleq$
.3 $(\text{dcl } \mathit{res\text{-}m} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL},$
.4 $\mathit{newid\text{-}s} : \mathit{AS`Name}\text{-set};$
.5 $\mathit{res\text{-}m} := \mathit{FreeInMultBindSeq}\ (\mathit{bind\text{-}l}, \mathit{id\text{-}s});$
.6 $\mathit{newid\text{-}s} := \bigcup \{\mathit{id\text{-}s}, \text{dom } \mathit{res\text{-}m}, \mathit{IdentInMultBindSeq}\ (\mathit{bind\text{-}l})\};$
.7 $\text{let mk-}\mathit{AS`Maplet}\ (\mathit{dom\text{-}e}, \mathit{rng\text{-}e}, \text{-}) = \mathit{elem} \text{ in}$
.8 $(\text{let } \mathit{tmp\text{-}m} = \mathit{FreeInExpr}\ (\mathit{dom\text{-}e}, \mathit{newid\text{-}s}) \text{ in}$
.9 $(\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus_m \mathit{tmp\text{-}m};$
.10 $\mathit{newid\text{-}s} := \mathit{newid\text{-}s} \cup \text{dom } \mathit{tmp\text{-}m});$
.11 $\text{let } \mathit{tmp\text{-}m} = \mathit{FreeInExpr}\ (\mathit{rng\text{-}e}, \mathit{newid\text{-}s}) \text{ in}$
.12 $(\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus_m \mathit{tmp\text{-}m};$
.13 $\mathit{newid\text{-}s} := \mathit{newid\text{-}s} \cup \text{dom } \mathit{tmp\text{-}m}));$
.14 $\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus_m \mathit{FreeInExpr}\ (\mathit{pred\text{-}e}, \mathit{newid\text{-}s});$
.15 $\text{return } \mathit{res\text{-}m}\ );$

Tuple Expressions

1311.0 $\mathit{FreeInTupleConstructorExpr} : \mathit{AS`TupleConstructorExpr} \times \mathit{AS`Name}\text{-set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInTupleConstructorExpr}\ (\text{mk-}\mathit{AS`TupleConstructorExpr}\ (\mathit{fields\text{-}le}, \text{-}), \mathit{id\text{-}s}) \triangleq$
.2 $(\text{dcl } \mathit{res\text{-}m} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL} := \{\mapsto\},$
.3 $\mathit{newid\text{-}s} : \mathit{AS`Name}\text{-set} := \mathit{id\text{-}s};$
.4 $\text{for } \mathit{elm} \text{ in } \mathit{fields\text{-}le}$
.5 $\text{do let } \mathit{tmp\text{-}m} = \mathit{FreeInExpr}\ (\mathit{elm}, \mathit{newid\text{-}s}) \text{ in}$
.6 $(\mathit{res\text{-}m} := \mathit{res\text{-}m} \uplus_m \mathit{tmp\text{-}m};$
.7 $\mathit{newid\text{-}s} := \mathit{newid\text{-}s} \cup \text{dom } \mathit{tmp\text{-}m});$
.8 $\text{return } \mathit{res\text{-}m}\ );$

1312.0 $\mathit{FreeInTupleSelectExpr} : \mathit{AS`TupleSelectExpr} \times \mathit{AS`Name}\text{-set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInTupleSelectExpr}\ (\text{mk-}\mathit{AS`TupleSelectExpr}\ (\mathit{tuple}, \text{-}, \text{-}), \mathit{id\text{-}s}) \triangleq$
.2 $\mathit{FreeInExpr}(\mathit{tuple}, \mathit{id\text{-}s})\ ;$

Record Expressions

1313.0 $\mathit{FreeInRecordConstructorExpr} : \mathit{AS`RecordConstructorExpr} \times \mathit{AS`Name}\text{-set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInRecordConstructorExpr}\ (\text{mk-}\mathit{AS`RecordConstructorExpr}\ (\text{-}, \mathit{fields\text{-}le}, \text{-}), \mathit{id\text{-}s}) \triangleq$
.2 $(\text{dcl } \mathit{res\text{-}m} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL} := \{\mapsto\},$
.3 $\mathit{newid\text{-}s} : \mathit{AS`Name}\text{-set} := \mathit{id\text{-}s};$

.4 for elm in $fields\text{-}le$
.5 do let $tmp\text{-}m = FreeInExpr\,(elm, newid\text{-}s)$ in
.6 $(res\text{-}m := res\text{-}m \uplus tmp\text{-}m;$
.7 $newid\text{-}s := newid\text{-}s \cup$ dom $tmp\text{-}m);$
.8 return $res\text{-}m$);

1314.0 $FreeInRecordModifierExpr : AS`RecordModifierExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInRecordModifierExpr$ (mk-$AS`RecordModifierExpr\,(rec\text{-}e, modifiers, \text{-}), id\text{-}s) \triangleq$
.2 (dcl $res\text{-}m : AS`Name \xrightarrow{m} SEM`VAL,$
.3 $newid\text{-}s : AS`Name\text{-}\mathsf{set};$
.4 $res\text{-}m := FreeInExpr\,(rec\text{-}e, id\text{-}s);$
.5 $newid\text{-}s := id\text{-}s \cup$ dom $res\text{-}m;$
.6 for mk-$AS`RecordModification\,(\text{-}, expr, \text{-})$ in $modifiers$
.7 do let $tmp\text{-}m = FreeInExpr\,(expr, newid\text{-}s)$ in
.8 $(res\text{-}m := res\text{-}m \uplus tmp\text{-}m;$
.9 $newid\text{-}s := newid\text{-}s \cup$ dom $tmp\text{-}m);$
.10 return $res\text{-}m$);

Token Constructor

1315.0 $FreeInTokenConstructorExpr : AS`TokenConstructorExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInTokenConstructorExpr$ (mk-$AS`TokenConstructorExpr\,(expr, \text{-}), id\text{-}s) \triangleq$
.2 $FreeInExpr(expr, id\text{-}s)$;

Apply Expressions

1316.0 $FreeInApplyExpr : AS`ApplyExpr \times AS`Name\text{-}\mathsf{set} \xrightarrow{o} AS`Name \xrightarrow{m} SEM`VAL$
.1 $FreeInApplyExpr$ (mk-$AS`ApplyExpr\,(fct\text{-}e, arg\text{-}le, \text{-}), id\text{-}s) \triangleq$
.2 (dcl $res\text{-}m : AS`Name \xrightarrow{m} SEM`VAL,$
.3 $newid\text{-}s : AS`Name\text{-}\mathsf{set} := id\text{-}s;$
.4 $res\text{-}m := FreeInExpr\,(fct\text{-}e, id\text{-}s);$
.5 $newid\text{-}s := newid\text{-}s \cup$ dom $res\text{-}m;$
.6 for $arg\text{-}e$ in $arg\text{-}le$
.7 do let $arg\text{-}m = FreeInExpr\,(arg\text{-}e, newid\text{-}s)$ in
.8 $(res\text{-}m := res\text{-}m \uplus arg\text{-}m;$
.9 $newid\text{-}s := newid\text{-}s \cup$ dom $arg\text{-}m);$
.10 return $res\text{-}m$);

1317.0 $FreeInFctTypeInstExpr : AS\text{‘}FctTypeInstExpr \times AS\text{‘}Name\text{-}\mathsf{set} \xrightarrow{o} AS\text{‘}Name \xrightarrow{m} SEM\text{‘}VAL$

.1 $FreeInFctTypeInstExpr\,(\mathsf{mk\text{-}}AS\text{‘}FctTypeInstExpr\,(polyfct, \text{-}, \text{-}), id\text{-}s) \triangleq$

.2 $FreeInExpr(polyfct, id\text{-}s)$;

1318.0 $FreeInIsExpr : AS\text{‘}IsExpr \times AS\text{‘}Name\text{-}\mathsf{set} \xrightarrow{o} AS\text{‘}Name \xrightarrow{m} SEM\text{‘}VAL$

.1 $FreeInIsExpr\,(\mathsf{mk\text{-}}AS\text{‘}IsExpr\,(\text{-}, arg\text{-}e, \text{-}), id\text{-}s) \triangleq$

.2 $FreeInExpr(arg\text{-}e, id\text{-}s)$;

Lambda Expression

1319.0 $FreeInLambdaExpr : AS\text{‘}LambdaExpr \times AS\text{‘}Name\text{-}\mathsf{set} \xrightarrow{o} AS\text{‘}Name \xrightarrow{m} SEM\text{‘}VAL$

.1 $FreeInLambdaExpr\,(\mathsf{mk\text{-}}AS\text{‘}LambdaExpr\,(tb\text{-}l, body\text{-}e, \text{-}), id\text{-}s) \triangleq$

.2 $\mathsf{let}\ newid\text{-}s = \bigcup \{IdentInBind\,(tb\text{-}l\,(i)) \mid i \in \mathsf{inds}\ tb\text{-}l\}\ \mathsf{in}$

.3 $FreeInExpr(body\text{-}e, id\text{-}s \cup newid\text{-}s)$;

Is Expression and Type Judgement

1320.0 $FreeInFieldSelectExpr : AS\text{‘}FieldSelectExpr \times AS\text{‘}Name\text{-}\mathsf{set} \xrightarrow{o} AS\text{‘}Name \xrightarrow{m} SEM\text{‘}VAL$

.1 $FreeInFieldSelectExpr\,(\mathsf{mk\text{-}}AS\text{‘}FieldSelectExpr\,(record\text{-}e, \text{-}, \text{-}), id\text{-}s) \triangleq$

.2 $FreeInExpr(record\text{-}e, id\text{-}s)$;

1321.0 $FreeInTypeJudgementExpr : AS\text{‘}TypeJudgementExpr \times AS\text{‘}Name\text{-}\mathsf{set} \xrightarrow{o} AS\text{‘}Name \xrightarrow{m}$
$SEM\text{‘}VAL$

.1 $FreeInTypeJudgementExpr\,(\mathsf{mk\text{-}}AS\text{‘}TypeJudgementExpr\,(expr, \text{-}, \text{-}), id\text{-}s) \triangleq$

.2 $FreeInExpr(expr, id\text{-}s)$;

Pre-condition Apply expressions

1322.0 $FreeInPreConditionApplyExpr : AS\text{‘}PreConditionApplyExpr \times AS\text{‘}Name\text{-}\mathsf{set} \xrightarrow{o}$

.1 $AS\text{‘}Name \xrightarrow{m} SEM\text{‘}VAL$

.2 $FreeInPreConditionApplyExpr\,(\mathsf{mk\text{-}}AS\text{‘}PreConditionApplyExpr\,(fct, arg, \text{-}), id\text{-}s) \triangleq$

.3 $(\mathsf{dcl}\ res\text{-}m : AS\text{‘}Name \xrightarrow{m} SEM\text{‘}VAL := FreeInExpr\,(fct, id\text{-}s),$

.4 $newid\text{-}s : AS\text{‘}Name\text{-}\mathsf{set} := id\text{-}s \cup \mathsf{dom}\ res\text{-}m;$

.5 $\mathsf{for}\ expr\ \mathsf{in}\ arg$

.6 $\mathsf{do}\ (\mathsf{let}\ tmp\text{-}m = FreeInExpr\,(expr, newid\text{-}s)\ \mathsf{in}$

.7 $(res\text{-}m := res\text{-}m \uplus tmp\text{-}m;$

.8 $newid\text{-}s := newid\text{-}s \cup \mathsf{dom}\ tmp\text{-}m));$

.9 $\mathsf{return}\ res\text{-}m\ );$

Names

1323.0 $\mathit{FreeInName} : \mathit{AS`Name} \times \mathit{AS`Name}\text{-}\mathsf{set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInName}\,(\mathit{name}, \mathit{id\text{-}s}) \triangleq$

.2 $\quad \mathsf{if}\ \mathit{name} \in \mathit{id\text{-}s}$

.3 $\quad \mathsf{then\ return}\ \{\mapsto\}$

.4 $\quad \mathsf{else\ let}\ \mathit{val} = \mathit{LookUpInTopEnv}\,(\mathit{name})\ \mathsf{in}$

.5 $\qquad\quad \mathsf{if}\ \mathit{val} = \mathsf{nil}$

.6 $\qquad\quad \mathsf{then\ return}\ \{\mapsto\}$

.7 $\qquad\quad \mathsf{else\ return}\ \{\mathit{name} \mapsto \mathit{val}\}\ ;$

Bracketed Expression

1324.0 $\mathit{FreeInBracketedExpr} : \mathit{AS`BracketedExpr} \times \mathit{AS`Name}\text{-}\mathsf{set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInBracketedExpr}\,(\mathsf{mk}\text{-}\mathit{AS`BracketedExpr}\,(\mathit{expr}, \text{-}), \mathit{id\text{-}s}) \triangleq$

.2 $\quad \mathit{FreeInExpr}(\mathit{expr}, \mathit{id\text{-}s})\ ;$

Iota Expression

1325.0 $\mathit{FreeInIotaExpr} : \mathit{AS`IotaExpr} \times \mathit{AS`Name}\text{-}\mathsf{set} \xrightarrow{o} \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL}$

.1 $\mathit{FreeInIotaExpr}\,(\mathsf{mk}\text{-}\mathit{AS`IotaExpr}\,(\mathit{bind}, \mathit{pred\text{-}e}, \text{-}), \mathit{id\text{-}s}) \triangleq$

.2 $\quad \mathsf{let}\ \mathit{patid\text{-}s} = \mathit{IdentInBind}\,(\mathit{bind}),$

.3 $\qquad\ \ \mathit{expid\text{-}m} = \mathit{FreeInBind}\,(\mathit{bind}, \mathit{id\text{-}s})\ \mathsf{in}$

.4 $\quad \mathsf{let}\ \mathit{newid\text{-}s} = \bigcup\{\mathit{id\text{-}s}, \mathit{patid\text{-}s}, \mathsf{dom}\ \mathit{expid\text{-}m}\}\ \mathsf{in}$

.5 $\quad \mathsf{let}\ \mathit{tmp\text{-}m} = \mathit{FreeInExpr}\,(\mathit{pred\text{-}e}, \mathit{newid\text{-}s})\ \mathsf{in}$

.6 $\quad \mathsf{let}\ \mathit{res\text{-}m} = \mathit{expid\text{-}m} \overset{m}{\cup} \mathit{tmp\text{-}m}\ \mathsf{in}$

.7 $\quad \mathsf{return}\ \mathit{res\text{-}m};$

1.22.2 Auxiliary Operations

1326.0 $\mathit{FreeMapToBlkEnv} : \mathit{AS`Name} \xrightarrow{m} \mathit{SEM`VAL} \xrightarrow{o} \mathit{SEM`BlkEnv}$

.1 $\mathit{FreeMapToBlkEnv}\,(\mathit{id\text{-}m}) \triangleq$

.2 $\quad (\mathsf{dcl}\ \mathit{env} : \mathit{SEM`BlkEnv} := \mathit{MkEmptyBlkEnv}\,(\mathrm{READ\_ONLY});$

.3 $\quad\ \mathsf{for\ all}\ \mathit{id} \in \mathsf{dom}\ \mathit{id\text{-}m}$

.4 $\quad\ \mathsf{do}\ \mathit{env} := \mathit{CombineBlkEnv}\,(\mathit{env}, \mathit{MkBlkEnv}\,(\mathit{id}, \mathit{id\text{-}m}\,(\mathit{id}), \mathsf{nil}\ , \mathrm{READ\_ONLY}));$

.5 $\quad\ \mathsf{return}\ \mathit{env}\ );$

1327.0 $LookUpInTopEnv : AS`Name \xrightarrow{o} [SEM`VAL]$

.1 $LookUpInTopEnv\,(name) \triangleq$

.2 let mk- $(isit, res) = STKM`IsLocalVal\,(name)$ in

.3 if $isit$

.4 then return res

.5 else return nil

end $FREE$

Test Suite : rtinfo.ast
Module : FREE

Name	**#Calls**	**Coverage**
FREE`FreeInBind	**undefined**	**undefined**
FREE`FreeInExpr	**undefined**	**undefined**
FREE`FreeInName	**undefined**	**undefined**
FREE`FreeInFnDef	**undefined**	**undefined**
FREE`IdentInBind	**undefined**	**undefined**
FREE`FreeInIfExpr	**undefined**	**undefined**
FREE`FreeInIsExpr	**undefined**	**undefined**
FREE`FreeInDefExpr	**undefined**	**undefined**
FREE`FreeInLetExpr	**undefined**	**undefined**
FREE`FreeInPreExpr	**undefined**	**undefined**
FREE`FreeInIotaExpr	**undefined**	**undefined**
FREE`FreeInPostExpr	**undefined**	**undefined**
FREE`IdentInPattern	**undefined**	**undefined**
FREE`LookUpInTopEnv	**undefined**	**undefined**
FREE`FreeInApplyExpr	**undefined**	**undefined**
FREE`FreeInCasesExpr	**undefined**	**undefined**
FREE`FreeMapToBlkEnv	**undefined**	**undefined**
FREE`FreeInBinaryExpr	**undefined**	**undefined**
FREE`FreeInElseifExpr	**undefined**	**undefined**
FREE`FreeInLambdaExpr	**undefined**	**undefined**
FREE`FreeInPrefixExpr	**undefined**	**undefined**
FREE`FreeInLetBeSTExpr	**undefined**	**undefined**
FREE`FreeInMultBindSeq	**undefined**	**undefined**
FREE`FreeInSetRangeExpr	**undefined**	**undefined**
FREE`IdentInMultBindSeq	**undefined**	**undefined**
FREE`FreeInBracketedExpr	**undefined**	**undefined**
FREE`FreeInMapInverseExpr	**undefined**	**undefined**
FREE`FreeInAllOrExistsExpr	**undefined**	**undefined**
FREE`FreeInFctTypeInstExpr	**undefined**	**undefined**
FREE`FreeInFieldSelectExpr	**undefined**	**undefined**
FREE`FreeInSubSequenceExpr	**undefined**	**undefined**
FREE`FreeInTupleSelectExpr	**undefined**	**undefined**
FREE`FreeInExistsUniqueExpr	**undefined**	**undefined**
FREE`FreeInTypeJudgementExpr	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
FREE‘FreeInMapEnumerationExpr	**undefined**	**undefined**
FREE‘FreeInRecordModifierExpr	**undefined**	**undefined**
FREE‘FreeInSeqEnumerationExpr	**undefined**	**undefined**
FREE‘FreeInSetEnumerationExpr	**undefined**	**undefined**
FREE‘FreeInMapComprehensionExpr	**undefined**	**undefined**
FREE‘FreeInSeqComprehensionExpr	**undefined**	**undefined**
FREE‘FreeInSetComprehensionExpr	**undefined**	**undefined**
FREE‘FreeInTokenConstructorExpr	**undefined**	**undefined**
FREE‘FreeInTupleConstructorExpr	**undefined**	**undefined**
FREE‘FreeInPreConditionApplyExpr	**undefined**	**undefined**
FREE‘FreeInRecordConstructorExpr	**undefined**	**undefined**
FREE‘FreeInSeqModifyMapOverrideExpr	**undefined**	**undefined**
Total Coverage		**0%**

Local Variables:

1.23 Old Name Substitution

The module OLD contains all definitions related to the subsitution of old names.

module OLD

imports

from AS all ,

from CI all ,

from POS all ,

from REP all

exports

operations $OldNameInExpr : AS`Expr \xrightarrow{o} AS`Expr$

definitions

1.23.1 Patterns and Binds

operations

$OldNameInPattern : AS`PatternBind \xrightarrow{o} AS`PatternBind$

$OldNameInPattern\,(pat) \triangleq$

cases pat:

mk-$AS`PatternName\,(\text{-},\text{-}) \rightarrow$

return pat,

mk-$AS`MatchVal\,(val, cid) \rightarrow$

return mk-$AS`MatchVal\,(OldNameInExpr\,(val), cid)$,

```
.7        mk-AS`SetEnumPattern (els-l, cid) →
.8            let l = [OldNameInPattern (els-l (i)) | i ∈ inds els-l] in
.9            return mk-AS`SetEnumPattern (l, cid),
.10       mk-AS`SetUnionPattern (lp, rp, cid) →
.11           return mk-AS`SetUnionPattern (OldNameInPattern (lp),
.12                                         OldNameInPattern (rp), cid),
.13       mk-AS`SeqEnumPattern (els-l, cid) →
.14           let l = [OldNameInPattern (els-l (i)) | i ∈ inds els-l] in
.15           return mk-AS`SeqEnumPattern (l, cid),
.16       mk-AS`SeqConcPattern (lp, rp, cid) →
.17           return mk-AS`SeqConcPattern (OldNameInPattern (lp),
.18                                        OldNameInPattern (rp), cid),
.19       mk-AS`RecordPattern (tag, fields, cid) →
.20           let l = [OldNameInPattern (fields (i)) | i ∈ inds fields] in
.21           return mk-AS`RecordPattern (tag, l, cid),
.22       mk-AS`TuplePattern (fields, cid) →
.23           let l = [OldNameInPattern (fields (i)) | i ∈ inds fields] in
.24           return mk-AS`TuplePattern (l, cid),
.25       mk-AS`SetBind (pat, set-e, cid) →
.26           return mk-AS`SetBind (OldNameInPattern (pat), OldNameInExpr (set-e), cid),
.27       mk-AS`TypeBind (pat, tp, cid) →
.28           return mk-AS`TypeBind (OldNameInPattern (pat), tp, cid),
.29       others → error
.30     end;
```

The operation *OldNameInPattern* replaces the possible *OldName* in a *MatchVal* or *SetBind* with a new *Name*:. Patterns and binds are scanned recursively for match values and set binds.

```
1334.0  OldNameInBind : AS`Bind -o-> AS`Bind
.1      OldNameInBind (bind) △
.2        OldNameInPattern(bind) ;
```

```
1335.0  OldNameInMultBindSeq : AS`MultBind* -o-> AS`MultBind*
.1      OldNameInMultBindSeq (bind-l) △
.2        (dcl new-l : AS`MultBind* := [],
.3             tmp-l : AS`Pattern* := [];
.4         for bind in bind-l
.5         do (tmp-l := [];
.6             for pat-p in bind.pat
.7             do tmp-l := tmp-l ⁀ [OldNameInPattern (pat-p)];
.8             if is-AS`MultSetBind (bind)
.9             then let b = mk-AS`MultSetBind (tmp-l, OldNameInExpr (bind.Set), bind.cid) in
.10                   new-l := new-l ⁀ [b]
.11            else new-l := new-l ⁀ [mk-AS`MultTypeBind (tmp-l, bind.tp, bind.cid)]);
.12        return new-l );
```

1.23.2 Expressions

```
1336.0 OldNameInExpr : AS`Expr ⥲ AS`Expr
   .1  OldNameInExpr (expr) △
   .2    cases true:
   .3      (is-AS`DefExpr (expr)) → OldNameInDefExpr(expr) ,
   .4      (is-AS`LetExpr (expr)) → OldNameInLetExpr(expr) ,
   .5      (is-AS`LetBeSTExpr (expr)) → OldNameInLetBeSTExpr(expr) ,
   .6      (is-AS`IfExpr (expr)) → OldNameInIfExpr(expr) ,
   .7      (is-AS`CasesExpr (expr)) → OldNameInCasesExpr(expr) ,
   .8      (is-AS`PrefixExpr (expr)) → OldNameInPrefixExpr(expr) ,
   .9      (is-AS`MapInverseExpr (expr)) → OldNameInMapInverseExpr(expr) ,
   .10     (is-AS`BinaryExpr (expr)) → OldNameInBinaryExpr(expr) ,
   .11     (is-AS`AllOrExistsExpr (expr)) → OldNameInAllOrExistsExpr(expr) ,
   .12     (is-AS`ExistsUniqueExpr (expr)) → OldNameInExistsUniqueExpr(expr) ,
   .13     (is-AS`SetEnumerationExpr (expr)) → OldNameInSetEnumerationExpr(expr) ,
   .14     (is-AS`SetComprehensionExpr (expr)) → OldNameInSetComprehensionExpr(expr),
   .15     (is-AS`SetRangeExpr (expr)) → OldNameInSetRangeExpr(expr) ,
   .16     (is-AS`SeqEnumerationExpr (expr)) → OldNameInSeqEnumerationExpr(expr) ,
   .17     (is-AS`SeqComprehensionExpr (expr)) → OldNameInSeqComprehensionExpr(expr),
   .18     (is-AS`SubSequenceExpr (expr)) → OldNameInSubSequenceExpr(expr) ,
   .19     (is-AS`SeqModifyMapOverrideExpr (expr)) → OldNameInSeqModifyMapOverrideExpr(expr),
   .20     (is-AS`MapEnumerationExpr (expr)) → OldNameInMapEnumerationExpr(expr),
   .21     (is-AS`MapComprehensionExpr (expr)) → OldNameInMapComprehensionExpr(expr),
   .22     (is-AS`TupleConstructorExpr (expr)) → OldNameInTupleConstructorExpr(expr),
   .23     (is-AS`RecordConstructorExpr (expr)) → OldNameInRecordConstructorExpr(expr),
   .24     (is-AS`RecordModifierExpr (expr)) → OldNameInRecordModifierExpr(expr) ,
   .25     (is-AS`TokenConstructorExpr (expr)) → OldNameInTokenConstructorExpr(expr),
   .26     (is-AS`ApplyExpr (expr)) → OldNameInApplyExpr(expr) ,
   .27     (is-AS`LambdaExpr (expr)) → OldNameInLambdaExpr(expr) ,
   .28     (is-AS`FieldSelectExpr (expr)) → OldNameInFieldSelectExpr(expr) ,
   .29     (is-AS`FctTypeInstExpr (expr)) → OldNameInFctTypeInstExpr(expr) ,
   .30     (is-AS`IsExpr (expr)) → OldNameInIsExpr(expr) ,
   .31     (is-AS`IotaExpr (expr)) → OldNameInIotaExpr(expr) ,
   .32     (is-AS`UndefinedExpr (expr)),
   .33     (is-AS`Name (expr)),
   .34     (is-AS`BoolLit (expr)),
   .35     (is-AS`CharLit (expr)),
   .36     (is-AS`TextLit (expr)),
   .37     (is-AS`QuoteLit (expr)),
   .38     (is-AS`RealLit (expr)),
   .39     (is-AS`NilLit (expr)) → return expr,
   .40     (is-AS`OldName (expr)) → OldNameInOldName(expr) ,
   .41     (is-AS`BracketedExpr (expr)) → OldNameInBracketedExpr(expr) ,
   .42     others → error
   .43   end;
```

The operation *OldNameInExpr* is the entry point for the substitution, and the operation is called recursively by the other operations.

Local Binding Expressions

1337.0 $\mathit{OldNameInDefExpr} : AS`\mathit{DefExpr} \xrightarrow{o} AS`\mathit{Expr}$
.1 $\mathit{OldNameInDefExpr}\,(\mathsf{mk}\text{-}AS`\mathit{DefExpr}\,(\mathit{def}\text{-}l, \mathit{in}\text{-}e, \mathit{cid})) \triangleq$
.2 $(\mathsf{dcl}\ \mathit{new}\text{-}l : (AS`\mathit{Pattern} \times AS`\mathit{Expr})^* := [\,];$
.3 $\mathsf{for}\ \mathsf{mk}\text{-}\,(\mathit{pat}\text{-}p, \mathit{val}\text{-}e)\ \mathsf{in}\ \mathit{def}\text{-}l$
.4 $\mathsf{do}\ \mathit{new}\text{-}l := \mathit{new}\text{-}l \curvearrowright [\mathsf{mk}\text{-}\,(\mathit{OldNameInPattern}\,(\mathit{pat}\text{-}p), \mathit{OldNameInExpr}\,(\mathit{val}\text{-}e))];$
.5 $\mathsf{let}\ \mathit{new}\text{-}\mathit{in} = \mathit{OldNameInExpr}\,(\mathit{in}\text{-}e)\ \mathsf{in}$
.6 $\mathsf{return}\ \mathsf{mk}\text{-}AS`\mathit{DefExpr}\,(\mathit{new}\text{-}l, \mathit{new}\text{-}\mathit{in}, \mathit{cid})\ );$

1338.0 $\mathit{OldNameInLetExpr} : AS`\mathit{LetExpr} \xrightarrow{o} AS`\mathit{Expr}$
.1 $\mathit{OldNameInLetExpr}\,(\mathsf{mk}\text{-}AS`\mathit{LetExpr}\,(\mathit{localdef}, \mathit{in}\text{-}e, \mathit{cid})) \triangleq$
.2 $(\mathsf{dcl}\ \mathit{new}\text{-}l : AS`\mathit{LocalDef}^* := [\,];$
.3 $\mathsf{for}\ \mathit{ldef}\ \mathsf{in}\ \mathit{localdef}$
.4 $\mathsf{do}\ \mathsf{if}\ \mathsf{is}\text{-}AS`\mathit{ValueDef}\,(\mathit{ldef})$
.5 $\quad \mathsf{then}\ \mathit{new}\text{-}l := \mathit{new}\text{-}l \curvearrowright [\mathsf{mk}\text{-}AS`\mathit{ValueDef}\,(\mathit{OldNameInPattern}\,(\mathit{ldef}.\mathit{pat}),$
.6 $\quad\quad \mathit{ldef}.\mathit{tp},$
.7 $\quad\quad \mathit{OldNameInExpr}\,(\mathit{ldef}.\mathit{val}),$
.8 $\quad\quad \mathrm{NOT\_INITIALISED\_AS},$
.9 $\quad\quad \mathsf{false},$
.10 $\quad\quad \mathit{ldef}.\mathit{cid})]$
.11 $\quad \mathsf{else}\ \mathit{new}\text{-}l := \mathit{new}\text{-}l \curvearrowright [\mathit{ldef}];$
.12 $\mathsf{return}\ \mathsf{mk}\text{-}AS`\mathit{LetExpr}\,(\mathit{new}\text{-}l, \mathit{OldNameInExpr}\,(\mathit{in}\text{-}e), \mathit{cid})\ );$

1339.0 $\mathit{OldNameInLetBeSTExpr} : AS`\mathit{LetBeSTExpr} \xrightarrow{o} AS`\mathit{Expr}$
.1 $\mathit{OldNameInLetBeSTExpr}\,(\mathsf{mk}\text{-}AS`\mathit{LetBeSTExpr}\,(\mathit{lhs}, \mathit{st}\text{-}e, \mathit{in}\text{-}e, \mathit{cid})) \triangleq$
.2 $\mathsf{return}\ \mathsf{mk}\text{-}AS`\mathit{LetBeSTExpr}\,(\mathit{OldNameInBind}\,(\mathit{lhs}),$
.3 $\quad \mathsf{if}\ \mathit{st}\text{-}e = \mathsf{nil}$
.4 $\quad \mathsf{then}\ \mathsf{nil}$
.5 $\quad \mathsf{else}\ \mathit{OldNameInExpr}\,(\mathit{st}\text{-}e),$
.6 $\quad \mathit{OldNameInExpr}\,(\mathit{in}\text{-}e), \mathit{cid});$

Conditional Expressions

1340.0 $\mathit{OldNameInIfExpr} : AS`\mathit{IfExpr} \xrightarrow{o} AS`\mathit{Expr}$
.1 $\mathit{OldNameInIfExpr}\,(\mathsf{mk}\text{-}AS`\mathit{IfExpr}\,(\mathit{test}\text{-}e, \mathit{cons}\text{-}e, \mathit{elif}, \mathit{altn}\text{-}e, \mathit{cid})) \triangleq$
.2 $(\mathsf{dcl}\ \mathit{new}\text{-}l : AS`\mathit{ElseifExpr}^* := [\,];$
.3 $\mathsf{for}\ \mathsf{mk}\text{-}AS`\mathit{ElseifExpr}\,(\mathit{tst}, \mathit{cns}, \mathit{cid}2)\ \mathsf{in}\ \mathit{elif}$

.4 do $new\text{-}l := new\text{-}l \curvearrowright [\textsf{mk-}AS`ElseifExpr\,(OldNameInExpr\,(tst),$
.5 $OldNameInExpr\,(cns), cid2)];$
.6 return $\textsf{mk-}AS`IfExpr\,(OldNameInExpr\,(test\text{-}e),$
.7 $OldNameInExpr\,(cons\text{-}e),$
.8 $new\text{-}l,$
.9 $OldNameInExpr\,(altn\text{-}e), cid)\ );$

1341.0 $OldNameInCasesExpr : AS`CasesExpr \xrightarrow{o} AS`Expr$
.1 $OldNameInCasesExpr\,(\textsf{mk-}AS`CasesExpr\,(sel\text{-}e, altns\text{-}l, others\text{-}e, cid)) \triangleq$
.2 (dcl $new\text{-}l : AS`CaseAltn^* := [],$
.3 $pat\text{-}lp : AS`Pattern^* := [];$
.4 for $\textsf{mk-}AS`CaseAltn\,(match, body, cid2)$ in $altns\text{-}l$
.5 do $(pat\text{-}lp := [];$
.6 for pat in $match$
.7 do $pat\text{-}lp := pat\text{-}lp \curvearrowright [OldNameInPattern\,(pat)];$
.8 $new\text{-}l := new\text{-}l \curvearrowright [\textsf{mk-}AS`CaseAltn\,(pat\text{-}lp, OldNameInExpr\,(body), cid2)]);$
.9 return $\textsf{mk-}AS`CasesExpr\,(OldNameInExpr\,(sel\text{-}e),$
.10 $new\text{-}l,$
.11 if $others\text{-}e =$ nil
.12 then nil
.13 else $OldNameInExpr\,(others\text{-}e),$
.14 $cid)\ );$

Unary Expressions

1342.0 $OldNameInPrefixExpr : AS`PrefixExpr \xrightarrow{o} AS`Expr$
.1 $OldNameInPrefixExpr\,(\textsf{mk-}AS`PrefixExpr\,(opr, arg\text{-}e, cid)) \triangleq$
.2 return $\textsf{mk-}AS`PrefixExpr\,(opr, OldNameInExpr\,(arg\text{-}e), cid);$

1343.0 $OldNameInMapInverseExpr : AS`MapInverseExpr \xrightarrow{o} AS`Expr$
.1 $OldNameInMapInverseExpr\,(\textsf{mk-}AS`MapInverseExpr\,(op, cid)) \triangleq$
.2 return $\textsf{mk-}AS`MapInverseExpr\,(OldNameInExpr\,(op), cid);$

Binary Expressions

1344.0 $OldNameInBinaryExpr : AS`BinaryExpr \xrightarrow{o} AS`Expr$
.1 $OldNameInBinaryExpr\,(\textsf{mk-}AS`BinaryExpr\,(left\text{-}e, opr, right\text{-}e, cid)) \triangleq$
.2 return $\textsf{mk-}AS`BinaryExpr\,(OldNameInExpr\,(left\text{-}e),$
.3 $opr,$
.4 $OldNameInExpr\,(right\text{-}e), cid);$

Quantified Expressions

1345.0 $\mathit{OldNameInAllOrExistsExpr} : \mathit{AS`AllOrExistsExpr} \xrightarrow{o} \mathit{AS`Expr}$
.1 $\mathit{OldNameInAllOrExistsExpr}\,(\text{mk-}\mathit{AS`AllOrExistsExpr}\,(\mathit{quant}, \mathit{bind\text{-}l}, \mathit{pred\text{-}e}, \mathit{cid})) \triangleq$
.2 return $\text{mk-}\mathit{AS`AllOrExistsExpr}\,(\mathit{quant},$
.3 $\mathit{OldNameInMultBindSeq}\,(\mathit{bind\text{-}l}),$
.4 $\mathit{OldNameInExpr}\,(\mathit{pred\text{-}e}), \mathit{cid});$

1346.0 $\mathit{OldNameInExistsUniqueExpr} : \mathit{AS`ExistsUniqueExpr} \xrightarrow{o} \mathit{AS`Expr}$
.1 $\mathit{OldNameInExistsUniqueExpr}\,(\text{mk-}\mathit{AS`ExistsUniqueExpr}\,(\mathit{bind}, \mathit{pred\text{-}e}, \mathit{cid})) \triangleq$
.2 return $\text{mk-}\mathit{AS`ExistsUniqueExpr}\,(\mathit{OldNameInBind}\,(\mathit{bind}),$
.3 $\mathit{OldNameInExpr}\,(\mathit{pred\text{-}e}), \mathit{cid});$

Set Expressions

1347.0 $\mathit{OldNameInSetEnumerationExpr} : \mathit{AS`SetEnumerationExpr} \xrightarrow{o} \mathit{AS`Expr}$
.1 $\mathit{OldNameInSetEnumerationExpr}\,(\text{mk-}\mathit{AS`SetEnumerationExpr}\,(\mathit{elm\text{-}le}, \mathit{cid})) \triangleq$
.2 let $\mathit{new\text{-}l} = [\mathit{OldNameInExpr}\,(\mathit{elm\text{-}le}\,(i)) \mid i \in \text{inds } \mathit{elm\text{-}le}]$ in
.3 return $\text{mk-}\mathit{AS`SetEnumerationExpr}\,(\mathit{new\text{-}l}, \mathit{cid});$

1348.0 $\mathit{OldNameInSetComprehensionExpr} : \mathit{AS`SetComprehensionExpr} \xrightarrow{o} \mathit{AS`Expr}$
.1 $\mathit{OldNameInSetComprehensionExpr}\,(\text{mk-}\mathit{AS`SetComprehensionExpr}\,(\mathit{elem\text{-}e}, \mathit{bind\text{-}l},$
.2 $\mathit{pred\text{-}e}, \mathit{cid})) \triangleq$
.3 return $\text{mk-}\mathit{AS`SetComprehensionExpr}\,(\mathit{OldNameInExpr}\,(\mathit{elem\text{-}e}),$
.4 $\mathit{OldNameInMultBindSeq}\,(\mathit{bind\text{-}l}),$
.5 if $(\text{is-}\mathit{AS`UndefinedExpr}\,(\mathit{pred\text{-}e}))$
.6 then $\mathit{pred\text{-}e}$
.7 else $\mathit{OldNameInExpr}\,(\mathit{pred\text{-}e}),$
.8 $\mathit{cid});$

1349.0 $\mathit{OldNameInSetRangeExpr} : \mathit{AS`SetRangeExpr} \xrightarrow{o} \mathit{AS`Expr}$
.1 $\mathit{OldNameInSetRangeExpr}\,(\text{mk-}\mathit{AS`SetRangeExpr}\,(\mathit{lb\text{-}e}, \mathit{ub\text{-}e}, \mathit{cid})) \triangleq$
.2 return $\text{mk-}\mathit{AS`SetRangeExpr}\,(\mathit{OldNameInExpr}\,(\mathit{lb\text{-}e}),$
.3 $\mathit{OldNameInExpr}\,(\mathit{ub\text{-}e}), \mathit{cid});$

Sequence Expressions

1350.0 $OldNameInSeqEnumerationExpr : AS`SeqEnumerationExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInSeqEnumerationExpr\,(\mathsf{mk\text{-}}AS`SeqEnumerationExpr\,(els\text{-}l, cid)) \triangleq$

.2 $\quad \mathsf{let}\ l = [OldNameInExpr\,(els\text{-}l\,(i)) \mid i \in \mathsf{inds}\ els\text{-}l]\ \mathsf{in}$

.3 $\quad \mathsf{return}\ \mathsf{mk\text{-}}AS`SeqEnumerationExpr\,(l, cid);$

1351.0 $OldNameInSeqComprehensionExpr : AS`SeqComprehensionExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInSeqComprehensionExpr\,(\mathsf{mk\text{-}}AS`SeqComprehensionExpr\,(elem\text{-}e, bind,$

.2 $\qquad pred\text{-}e, cid)) \triangleq$

.3 $\quad \mathsf{return}\ \mathsf{mk\text{-}}AS`SeqComprehensionExpr\,(OldNameInExpr\,(elem\text{-}e),$

.4 $\qquad OldNameInBind\,(bind),$

.5 $\qquad \mathsf{if}\ (\mathsf{is\text{-}}AS`UndefinedExpr\,(pred\text{-}e))$

.6 $\qquad \mathsf{then}\ pred\text{-}e$

.7 $\qquad \mathsf{else}\ OldNameInExpr\,(pred\text{-}e),$

.8 $\qquad cid);$

1352.0 $OldNameInSubSequenceExpr : AS`SubSequenceExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInSubSequenceExpr\,(\mathsf{mk\text{-}}AS`SubSequenceExpr\,(seq\text{-}e, from\text{-}e, to\text{-}e, cid)) \triangleq$

.2 $\quad \mathsf{return}\ \mathsf{mk\text{-}}AS`SubSequenceExpr\,(OldNameInExpr\,(seq\text{-}e),$

.3 $\qquad OldNameInExpr\,(from\text{-}e),$

.4 $\qquad OldNameInExpr\,(to\text{-}e), cid);$

Map Expressions

1353.0 $OldNameInSeqModifyMapOverrideExpr : AS`SeqModifyMapOverrideExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInSeqModifyMapOverrideExpr\,(\mathsf{mk\text{-}}AS`SeqModifyMapOverrideExpr\,(seqmap\text{-}e,$

.2 $\qquad map\text{-}e, cid)) \triangleq$

.3 $\quad \mathsf{return}\ \mathsf{mk\text{-}}AS`SeqModifyMapOverrideExpr\,(OldNameInExpr\,(seqmap\text{-}e),$

.4 $\qquad OldNameInExpr\,(map\text{-}e), cid);$

1354.0 $OldNameInMapEnumerationExpr : AS`MapEnumerationExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInMapEnumerationExpr\,(\mathsf{mk\text{-}}AS`MapEnumerationExpr\,(els\text{-}l, cid)) \triangleq$

.2 $\quad (\mathsf{dcl}\ l : AS`Maplet^* := [];$

.3 $\quad \mathsf{for}\ \mathsf{mk\text{-}}AS`Maplet\,(md, mr, cid2)\ \mathsf{in}\ els\text{-}l$

.4 $\quad \mathsf{do}\ l := l \curvearrowright [\mathsf{mk\text{-}}AS`Maplet\,(OldNameInExpr\,(md), OldNameInExpr\,(mr), cid2)];$

.5 $\quad \mathsf{return}\ \mathsf{mk\text{-}}AS`MapEnumerationExpr\,(l, cid)\ );$

1355.0 $OldNameInMapComprehensionExpr : AS`MapComprehensionExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInMapComprehensionExpr$ (mk-$AS`MapComprehensionExpr\,(elem, bind\text{-}l, pred\text{-}e, cid)) \triangleq$

.2 let mk-$AS`Maplet\,(md, mr, cid2) = elem$ in

.3 return mk-$AS`MapComprehensionExpr$ (mk-$AS`Maplet\,(OldNameInExpr\,(md),$

.4 $OldNameInExpr\,(mr), cid2),$

.5 $OldNameInMultBindSeq\,(bind\text{-}l),$

.6 if (is-$AS`UndefinedExpr\,(pred\text{-}e))$

.7 then $pred\text{-}e$

.8 else $OldNameInExpr\,(pred\text{-}e),$

.9 $cid);$

Tuple Constructor

1356.0 $OldNameInTupleConstructorExpr : AS`TupleConstructorExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInTupleConstructorExpr$ (mk-$AS`TupleConstructorExpr\,(fields\text{-}le, cid)) \triangleq$

.2 let $l = [OldNameInExpr\,(fields\text{-}le\,(i)) \mid i \in$ inds $fields\text{-}le]$ in

.3 return mk-$AS`TupleConstructorExpr\,(l, cid);$

Record Expressions

1357.0 $OldNameInRecordConstructorExpr : AS`RecordConstructorExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInRecordConstructorExpr$ (mk-$AS`RecordConstructorExpr\,(tag, fields\text{-}le, cid)) \triangleq$

.2 let $l = [OldNameInExpr\,(fields\text{-}le\,(i)) \mid i \in$ inds $fields\text{-}le]$ in

.3 return mk-$AS`RecordConstructorExpr\,(tag, l, cid);$

1358.0 $OldNameInRecordModifierExpr : AS`RecordModifierExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInRecordModifierExpr$ (mk-$AS`RecordModifierExpr\,(rec\text{-}e, modifiers, cid)) \triangleq$

.2 (dcl $l : AS`RecordModification^* := []$;

.3 for mk-$AS`RecordModification\,(field, new, cid2)$ in $modifiers$

.4 do $l := l \curvearrowright$ [mk-$AS`RecordModification\,(field, OldNameInExpr\,(new), cid2)]$;

.5 return mk-$AS`RecordModifierExpr\,(OldNameInExpr\,(rec\text{-}e), l, cid)\ );$

Token Constructor

1359.0 $OldNameInTokenConstructorExpr : AS`TokenConstructorExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInTokenConstructorExpr$ (mk-$AS`TokenConstructorExpr\,(expr, cid)) \triangleq$

.2 return mk-$AS`TokenConstructorExpr\,(OldNameInExpr\,(expr), cid);$

Apply Expressions

1360.0 $OldNameInApplyExpr : AS`ApplyExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInApplyExpr\,(\mathsf{mk}\text{-}AS`ApplyExpr\,(fct\text{-}e, arg\text{-}le, cid)) \triangleq$

.2 $\quad \mathsf{let}\ l = [OldNameInExpr\,(arg\text{-}le\,(i)) \mid i \in \mathsf{inds}\ arg\text{-}le]\ \mathsf{in}$

.3 $\quad \mathsf{return}\ \mathsf{mk}\text{-}AS`ApplyExpr\,(OldNameInExpr\,(fct\text{-}e), l, cid);$

1361.0 $OldNameInFieldSelectExpr : AS`FieldSelectExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInFieldSelectExpr\,(\mathsf{mk}\text{-}AS`FieldSelectExpr\,(record\text{-}e, field, cid)) \triangleq$

.2 $\quad \mathsf{return}\ \mathsf{mk}\text{-}AS`FieldSelectExpr\,(OldNameInExpr\,(record\text{-}e), field, cid);$

1362.0 $OldNameInFctTypeInstExpr : AS`FctTypeInstExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInFctTypeInstExpr\,(\mathsf{mk}\text{-}AS`FctTypeInstExpr\,(polyfct, inst, cid)) \triangleq$

.2 $\quad \mathsf{return}\ \mathsf{mk}\text{-}AS`FctTypeInstExpr\,(OldNameInExpr\,(polyfct), inst, cid);$

Lambda Expression

1363.0 $OldNameInLambdaExpr : AS`LambdaExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInLambdaExpr\,(\mathsf{mk}\text{-}AS`LambdaExpr\,(tb\text{-}l, body\text{-}e, cid)) \triangleq$

.2 $\quad \mathsf{return}\ \mathsf{mk}\text{-}AS`LambdaExpr\,(tb\text{-}l, OldNameInExpr\,(body\text{-}e), cid);$

Is Expression

1364.0 $OldNameInIsExpr : AS`IsExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInIsExpr\,(\mathsf{mk}\text{-}AS`IsExpr\,(type, arg\text{-}e, cid)) \triangleq$

.2 $\quad \mathsf{return}\ \mathsf{mk}\text{-}AS`IsExpr\,(type, OldNameInExpr\,(arg\text{-}e), cid);$

Old Names

1365.0 $OldNameInOldName : AS`OldName \xrightarrow{o} AS`Expr$

.1 $OldNameInOldName\,(\mathsf{mk}\text{-}AS`OldName\,(id\text{-}l, cid)) \triangleq$

.2 $\quad \mathsf{let}\ id = id\text{-}l\,(\mathsf{len}\ id\text{-}l)\ \mathsf{in}$

.3 $\quad \mathsf{let}\ id' = id \curvearrowright \texttt{"\~{}"}\ \mathsf{in}$

.4 $\quad \mathsf{return}\ \mathsf{mk}\text{-}AS`Name\,([id'], cid);$

The operation *OldNameInOldName* is used to substitute an *OldName* with a new *Name*:. A property of the parser is used to create a unique name: in the parser, we cannot use the hook symbol in a pattern as it is a reserved symbol. However, in the interpreter, we can use the hook symbol to create a unique name. The new name is equal to the identifier of the *OldName*, appended with the hook symbol.

Bracketed Expression

1366.0 $OldNameInBracketedExpr : AS`BracketedExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInBracketedExpr\,(\textsf{mk-}AS`BracketedExpr\,(expr, cid)) \triangleq$

.2 $\quad\textsf{return mk-}AS`BracketedExpr\,(OldNameInExpr\,(expr), cid);$

Iota Expression

1367.0 $OldNameInIotaExpr : AS`IotaExpr \xrightarrow{o} AS`Expr$

.1 $OldNameInIotaExpr\,(\textsf{mk-}AS`IotaExpr\,(bind, pred\text{-}e, cid)) \triangleq$

.2 $\quad\textsf{return mk-}AS`IotaExpr\,(OldNameInBind\,(bind), OldNameInExpr\,(pred\text{-}e), cid)$

end *OLD*

Test Suite : rtinfo.ast
Module : OLD

Name	#Calls	Coverage
OLD`OldNameInBind	**undefined**	**undefined**
OLD`OldNameInExpr	**undefined**	**undefined**
OLD`OldNameInIfExpr	**undefined**	**undefined**
OLD`OldNameInIsExpr	**undefined**	**undefined**
OLD`OldNameInDefExpr	**undefined**	**undefined**
OLD`OldNameInLetExpr	**undefined**	**undefined**
OLD`OldNameInOldName	**undefined**	**undefined**
OLD`OldNameInPattern	**undefined**	**undefined**
OLD`OldNameInIotaExpr	**undefined**	**undefined**
OLD`OldNameInApplyExpr	**undefined**	**undefined**
OLD`OldNameInCasesExpr	**undefined**	**undefined**
OLD`OldNameInBinaryExpr	**undefined**	**undefined**
OLD`OldNameInLambdaExpr	**undefined**	**undefined**
OLD`OldNameInPrefixExpr	**undefined**	**undefined**
OLD`OldNameInLetBeSTExpr	**undefined**	**undefined**
OLD`OldNameInMultBindSeq	**undefined**	**undefined**
OLD`OldNameInSetRangeExpr	**undefined**	**undefined**
OLD`OldNameInBracketedExpr	**undefined**	**undefined**
OLD`OldNameInMapInverseExpr	**undefined**	**undefined**

Name	#Calls	Coverage
OLD‘OldNameInAllOrExistsExpr	**undefined**	**undefined**
OLD‘OldNameInFctTypeInstExpr	**undefined**	**undefined**
OLD‘OldNameInFieldSelectExpr	**undefined**	**undefined**
OLD‘OldNameInSubSequenceExpr	**undefined**	**undefined**
OLD‘OldNameInExistsUniqueExpr	**undefined**	**undefined**
OLD‘OldNameInMapEnumerationExpr	**undefined**	**undefined**
OLD‘OldNameInRecordModifierExpr	**undefined**	**undefined**
OLD‘OldNameInSeqEnumerationExpr	**undefined**	**undefined**
OLD‘OldNameInSetEnumerationExpr	**undefined**	**undefined**
OLD‘OldNameInMapComprehensionExpr	**undefined**	**undefined**
OLD‘OldNameInSeqComprehensionExpr	**undefined**	**undefined**
OLD‘OldNameInSetComprehensionExpr	**undefined**	**undefined**
OLD‘OldNameInTokenConstructorExpr	**undefined**	**undefined**
OLD‘OldNameInTupleConstructorExpr	**undefined**	**undefined**
OLD‘OldNameInRecordConstructorExpr	**undefined**	**undefined**
OLD‘OldNameInSeqModifyMapOverrideExpr	**undefined**	**undefined**
Total Coverage		**0%**

module $RTERR$

imports

1368.0 from AS all ,

1369.0 from CI all ,

1370.0 from IO all ,

1371.0 from PAT all ,

1372.0 from REP all ,

1373.0 from SEM all ,

1374.0 from $STKM$ all ,

1375.0 from $STATE$ all ,

1376.0 from $GLOBAL$ all ,

1377.0 from $SCHDTP$ all ,

1378.0 from $INSTRTP$ all ,

1379.0 from $TIMEMAP$ all ,

1380.0 from $TIMEPARSER$ all

exports all

definitions

types

1381.0 $ERR = \text{char}^*$

operations

1382.0 $ErrorVal : ERR \times [SEM‘VAL] \times [AS‘Type] \times \mathsf{char}^{**} \xrightarrow{o}$
.1 $[SEM‘VAL] \mid SEM‘VAL \times (AS‘Name \mid SEM‘VAL)^{*} \mid$
.2 $\mathbb{B} \times [CI‘ContextId] \times [AS‘Name] \times [AS‘Name] \mid$
.3 $\mathbb{B} \times \mathbb{B} \times [SEM‘VAL] \times [AS‘Type] \times [AS‘Name] \times [AS‘Access] \mid$
.4 $SEM‘BlkEnv\text{-}\mathsf{set} \mid AS‘Name\text{-}\mathsf{set} \mid$
.5 $\mathbb{B} \times [GLOBAL‘Type] \times [AS‘Invariant] \times [AS‘Name] \times [AS‘Access] \mid \mathbb{B} \mid$
.6 $\mathbb{B} \times \mathbb{B} \times [SEM‘VAL] \mid STKM‘DebugCmd \mid$
.7 $\mathbb{N} \times (AS‘Name \xrightarrow{m} \mathbb{N}) \times AS‘Type^{*} \mid$
.8 $(\mathbb{B} \times [SEM‘VAL])$
.9 $ErrorVal\,(err, semval, type, txts) \triangleq$
.10 $\quad (Error(err, semval, type, txts)\,;$
.11 $\quad\ \mathsf{return}\ semval\ );$

The *ErrorVal* operation is only needed to ensure that no type errors are reported from the places where the error is produced where a value should be produced. The value will in reality never be returned!

1383.0 $Error : ERR \times [SEM‘VAL] \times [AS‘Type] \times \mathsf{char}^{**} \xrightarrow{o} ()$
.1 $Error\,(err, semval, type, txts) \triangleq$
.2 $\quad (SetError(\text{"}Run\text{-}TimeError\text{"} \curvearrowright err, semval, type, txts)\,;$
.3 $\quad\ \mathsf{let}\ \text{-} = IO‘fwriteval[\mathsf{char}^{*}]\,(\text{"}rterr.msg\text{"}, GetError\,(), \textsc{start})\ \mathsf{in}$
.4 $\quad\ \mathsf{error});$

The *InitError* operation is meant to be used for errors which are to be reported in the initialisation phase (i.e. before the state machine evaluation has really started).

1384.0 $InitError : ERR \times CI‘ContextId \xrightarrow{o} ()$
.1 $InitError\,(err, cid) \triangleq$
.2 $\quad (\mathsf{dcl}\ posmsg : \mathsf{char}^{*};$
.3 $\quad\ \mathsf{if}\ cid = CI‘NilContextId$
.4 $\quad\ \mathsf{then}\ posmsg := \text{"}Unknownposition\backslash n\text{"}$
.5 $\quad\ \mathsf{else\ let\ mk\text{-}}(\text{-}, \text{-}, \text{-}, tokenpos, \text{-}) = CI‘GetFilePos\,(cid),$
.6 $\qquad\qquad \mathsf{mk\text{-}}CI‘TokenPos\,(sec, abs\text{-}line, \text{-}, sec\text{-}line, column) = tokenpos,$
.7 $\qquad\qquad line = Num2Str\,(abs\text{-}line),$
.8 $\qquad\qquad column\text{-}s = Num2Str\,(column)\ \mathsf{in}$
.9 $\qquad\quad posmsg := \text{"}Line\text{"} \curvearrowright line \curvearrowright \text{"}column\text{"} \curvearrowright column\text{-}s \curvearrowright \text{"}\backslash n\text{"};$
.10 $\quad\ errmsg := posmsg \curvearrowright \text{"}Run\text{-}TimeError\text{"} \curvearrowright err;$
.11 $\quad\ \mathsf{let}\ \text{-} = IO‘fwriteval[\mathsf{char}^{*}]\,(\text{"}rterr.msg\text{"}, GetError\,(), \textsc{start})\ \mathsf{in}$
.12 $\quad\ \mathsf{error});$

1385.0 $SetError : ERR \times [SEM`VAL] \times [AS`Type] \times \mathsf{char}^{**} \xrightarrow{o} ()$

.1 $SetError\,(err, semval, type, txts) \triangleq$

.2 $($let $cid = STKM`GetCurCid\,()$ in

.3 if $cid = CI`NilContextId$

.4 then $errmsg :=$ "$Commandlineerror\backslash n$"

.5 else let mk- $(\text{-}, \text{-}, \text{-}, tokenpos, \text{-}) = CI`GetFilePos\,(cid)$,

.6 mk-$CI`TokenPos\,(sec, abs\text{-}line, \text{-}, sec\text{-}line, column) = tokenpos$,

.7 $line = Num2Str\,(abs\text{-}line)$,

.8 $column\text{-}s = Num2Str\,(column)$ in

.9 $errmsg :=$ "$Line$" $\curvearrowright line \curvearrowright$ "$column$" $\curvearrowright column\text{-}s \curvearrowright$ "$\backslash n$";

.10 $errmsg := errmsg \curvearrowright err)$;

1386.0 $GetError : () \xrightarrow{o} \mathsf{char}^{*}$

.1 $GetError\,() \triangleq$

.2 return $errmsg$

functions

1387.0 $Num2Str : \mathbb{N} \to \mathsf{char}^{*}$

.1 $Num2Str\,(n) \triangleq$

.2 let $e = n$ div 10 in

.3 if $e = 0$

.4 then $ConvertChar\,(n)$

.5 else $Num2Str\,(e) \curvearrowright ConvertChar\,(n$ rem $10)$;

1388.0 $ConvertChar : \mathbb{N} \to \mathsf{char}^{*}$

.1 $ConvertChar\,(e) \triangleq$

.2 cases e :

.3 $0 \to$ "0",

.4 $1 \to$ "1",

.5 $2 \to$ "2",

.6 $3 \to$ "3",

.7 $4 \to$ "4",

.8 $5 \to$ "5",

.9 $6 \to$ "6",

.10 $7 \to$ "7",

.11 $8 \to$ "8",

.12 $9 \to$ "9",

.13 others $\to$ undefined

.14 end

1389.0 state $sigma$ of

.1 $errmsg : \mathsf{char}^{*}$

.2 init $s \triangleq s =$ mk-$sigma\,([])$

.3 end

values

1390.0 $BOOL\text{-}EXPECTED = \texttt{"}1 : Abooleanwasexpected.\texttt{"};$

1391.0 $INT\text{-}EXPECTED = \texttt{"}2 : Anintegerwasexpected.\texttt{"};$

1392.0 $NUM\text{-}EXPECTED = \texttt{"}3 : Anumberwasexpected\texttt{"};$

1393.0 $SET\text{-}EXPECTED = \texttt{"}4 : Asetwasexpected\texttt{"};$

1394.0 $SEQ\text{-}EXPECTED = \texttt{"}5 : Asequencewasexpected\texttt{"};$

1395.0 $REC\text{-}EXPECTED = \texttt{"}6 : Arecordwasexpected\texttt{"};$

1396.0 $MAP\text{-}EXPECTED = \texttt{"}7 : Amapwasexpected\texttt{"};$

1397.0 $TUPLE\text{-}EXPECTED = \texttt{"}8 : Atuplewasexpected\texttt{"};$

1398.0 $SYMB\text{-}LIT\text{-}EXPECTED = \texttt{"}9 : Asymbolicliteralwasexpected\texttt{"};$

1399.0 $TWO\text{-}BOOL\text{-}EXPECTED = \texttt{"}11 : Twobooleanswereexpected\texttt{"};$

1400.0 $TWO\text{-}INT\text{-}EXPECTED = \texttt{"}12 : Twointegerswereexpected\texttt{"};$

1401.0 $TWO\text{-}NUM\text{-}EXPECTED = \texttt{"}13 : Twonumberswereexpected\texttt{"};$

1402.0 $TWO\text{-}SET\text{-}EXPECTED = \texttt{"}14 : Twosetswereexpected\texttt{"};$

1403.0 $TWO\text{-}SEQ\text{-}EXPECTED = \texttt{"}15 : Twosequenceswereexpected\texttt{"};$

1404.0 $MAP\text{-}OR\text{-}SEQ\text{-}EXPECTED = \texttt{"}16 : Amaporasequencewasexpected\texttt{"};$

1405.0 $TWO\text{-}MAP\text{-}EXPECTED = \texttt{"}17: Twomapswereexpected\texttt{"};$

1406.0 $SET\text{-}AND\text{-}MAP\text{-}EXPECTED = \texttt{"}18: Asetandamapwereexpected\texttt{"};$

1407.0 $SEQ\text{-}AND\text{-}INT\text{-}EXPECTED = \texttt{"}19: Asequenceandtwointegerswereexpected\texttt{"};$

1408.0 $PATTERNNAME\text{-}EXPECTED = \texttt{"}20: Thebindingpatternmustevaluatetoonepatternname\texttt{"};$

1409.0 $CHAR\text{-}EXPECTED = \texttt{"}21: Acharwasexpected\texttt{"};$

1410.0 $ALL\text{-}INTS\text{-}EXPECTED = \texttt{"}22: Thebindingsetcanonlycontainintegers\texttt{"};$

1411.0 $ALL\text{-}NUMS\text{-}EXPECTED = \texttt{"}23: Theindicesfortheloopmustallbenumbers\texttt{"};$

1412.0 $ALL\text{-}SETS\text{-}EXPECTED = \texttt{"}24: Allelementsto'dunion'or'dinter'mustbesets\texttt{"};$

1413.0 $ALL\text{-}SEQS\text{-}EXPECTED = \texttt{"}25: Allelementsto'conc'mustbesequences\texttt{"};$

1414.0 $ALL\text{-}MAPS\text{-}EXPECTED = \texttt{"}27: Allelementsto'merge'mustbemaps\texttt{"};$

1415.0 $OLDID\text{-}NOT\text{-}IN\text{-}OS = \texttt{"}51: Theoldnameisnotstoredinthestate\texttt{"};$

1416.0 $MAP\text{-}MERGE\text{-}DOM\text{-}OVERLAP = \texttt{"}52: Duplicateentriesfor'merge'haddifferentvalues\texttt{"};$

1417.0 $EMPTY\text{-}ENV\text{-}S = \texttt{"}53: Thebindingenvironmentwasempty\texttt{"};$

1418.0 $ZERO\text{-}BY\text{-}STEP = \texttt{"}57: Steplengthinloopwas0\texttt{"};$

1419.0 $EVAL\text{-}PRE\text{-}GIVES\text{-}FALSE = \texttt{"}58: Thepre\text{-}conditionevaluatedtofalse\texttt{"};$

1420.0 *EVAL-POST-GIVES-FALSE* = "59 : *Thepost-conditionevaluatedtofalse*";

1421.0 *ERROR-STATEMENT* = "61 : *Cannotevaluate'error'statement*";

1422.0 *UNDEFINED-EXPRESSION* = "62 : *Cannotevaluate'undefined'expression*";

1423.0 *NO-OTHERS-EXPR* = "63 : *No'others'branchin'cases'expr*";

1424.0 *WRONG-QUANTIFIER* = "64 : *Wrongquantifier*";

1425.0 *PAT-NAME-IN-SEQCOMP* = "65:*Onlypatternnameisallowedinsequencecomprehension*";

1426.0 *ILLEGAL-INDICES* = "66 : *Illegalindex*";

1427.0 *DUPLICATES-NOT-EQUAL* = "67 : *Duplicateentrieshaddifferentvalues*";

1428.0 *RECORD-TAG-UNKNOWN* = "68 : *Unknownrecordtag*";

1429.0 *RECORD-SIZE-WRONG* = "69 : *Actualrecordsizedifferentfromdefinition*";

1430.0 *RECORD-FIELD-ID-UNKNOWN* = "70 : *Unknownrecordfieldselector*";

1431.0 *ARG-NOT-IN-DOM* = "71 : *Argumentnotfoundinmapdomain*";

1432.0 *FCT-V-TYPE-EXPECTED* = "72 : *Anexplicitfunction/operationwasexpected*";

1433.0 *UNDEF-ENCOUNTERED* = "73 : *Identifierisundefined/notinitialized*";

1434.0 *ID-UNKNOWN* = "74 : *Unknownidentifier*";

1435.0 *OPERATOR-UNKNOWN* = "75 : *Unknownoperator*";

1436.0 $DIVISION\text{-}WITH\text{-}ZERO = \texttt{"}76 : Divisionwithzero\texttt{"};$

1437.0 $ZERO\text{-}LENGTH\text{-}DETECTED = \texttt{"}77 : Thesequencewasempty\texttt{"};$

1438.0 $PATTERN\text{-}UNKNOWN = \texttt{"}78 : Unknownpattern\texttt{"};$

1439.0 $SET\text{-}TOO\text{-}BIG = \texttt{"}79 : Settoobigfor\text{'}power\text{'}\text{-}limitis16\texttt{"};$

1440.0 $EXPR\text{-}UNKNOWN = \texttt{"}80 : Expressionunknown\texttt{"};$

1441.0 $WRONG\text{-}NO\text{-}OF\text{-}ARGS = \texttt{"}81 : Wrongnumberofarguments\texttt{"};$

1442.0 $STMT\text{-}UNKNOWN = \texttt{"}83 : Unknownstatement\texttt{"};$

1443.0 $REF\text{-}UNKNOWN = \texttt{"}84 : Unknownreferenceinassignstatement\texttt{"};$

1444.0 $TYPE\text{-}UNKNOWN = \texttt{"}85 : Unknowntype\texttt{"};$

1445.0 $IS\text{-}TYPE\text{-}UNKNOWN = \texttt{"}86 : Unknowntypeinis\text{-}expression\texttt{"};$

1446.0 $IDENTICAL\text{-}FIELDS = \texttt{"}89 : identicalselectornamesincompositetype\texttt{"};$

1447.0 $TYPE\text{-}INCOMP = \texttt{"}98 : Incompatibletypesfoundindynamictypecheck\texttt{"};$

1448.0 $STATE\text{-}INVARIANT\text{-}ERROR = \texttt{"}99 : Stateinvariantwasbroken\texttt{"};$

1449.0 $TAG\text{-}MULTIPLE\text{-}DEF = \texttt{"}100 : Multipledefinitionsoftag\texttt{"};$

1450.0 $NO\text{-}UNIQ\text{-}ELEM = \texttt{"}110 : Nouniqueelementin\text{'}iota\text{'}\texttt{"};$

1451.0 *OP-EXIT-CONT* = "111 : *Operationexitedorreturnednoresult*";

1452.0 *NO-POLY-FUNC* = "112 : *Instantiatedfunctionisnotpolymorhic*";

1453.0 *NO-INJECTIVE-MAP* = "113 : *Mapisnotinjective-'inverse'fails*";

1454.0 *NOT-RNG-DOM-SUBSET* = "114 : *Therangeisnotasubsetofthedomain*";

1455.0 *TWO-FN-OR-MAP-EXPECTED* = "115: *Twofunctionsormapsexpectedfor'comp'*";

1456.0 *FN-OR-MAP-OR-NUM-EXPECTED* = "116 : *Wrongargumentsfor'* $**$'";

1457.0 *NAT-EXPECTED* = "117 : *Anaturalnumberwasexpected*";

1458.0 *OP-RETURNED-CONT* = "119 : *Theoperationdidnotreturnavalue*";

1459.0 *STATE-DESIG-UNKNOWN* = "120 : *Unknownstatecomponent*";

1460.0 *IMPL-OP-CALL* = "121 : *Triedtocallanimplicitoperation*";

1461.0 *ILLEGAL-STATE-INIT* = "122 : *Illegalstateinitialisation*";

1462.0 *ILL-STATE-INIT-PAT* = "123 : *Illegalstateinitialisationpattern*";

1463.0 *REC-PAT-EXPECTED* = "124 : *Recordpatternexpectedforstateinitialisation*";

1464.0 *ALL-PATID-EXPECTED* = "125:*Onlypatternidentifiersallowedforstateinitialisation*";

1465.0 *TYPE-BIND-EVAL* = "126 : *Cannotevaluatetypebinds*";

1466.0 *FNDEF-EXPECTED* = "127 : *Functiondefinitionexpected*";

1467.0 $IMPL\text{-}FN\text{-}APPLY = \texttt{"}128 : Triedtoapplyanimplicitfunction\texttt{"};$

1468.0 $POLY\text{-}NOT\text{-}INST = \texttt{"}129 : Theappliedpolymorhicfunctionisnotinstantiated\texttt{"};$

1469.0 $MAP\text{-}AND\text{-}SET\text{-}EXPECTED = \texttt{"}130 : Amapandasetwereexpected\texttt{"};$

1470.0 $NOT\text{-}EXPL\text{-}OP\text{-}CALL = \texttt{"}131 : Thecalledobjectisnotanexplicitoperation\texttt{"};$

1471.0 $GET\text{-}VALUE\text{-}EXPR = \texttt{"}132 : Noinputvaluetoget\text{-}value()\texttt{"};$

1472.0 $OP\text{-}CANT\text{-}RETURN\text{-}A\text{-}VALUE = \texttt{"}133 : Theoperation\text{'}srangeisempty\texttt{"};$

1473.0 $MOD\text{-}ALREADY\text{-}DEF = \texttt{"}150 : Themoduleisalreadydefined\texttt{"};$

1474.0 $EQUAL\text{-}RENAMINGS = \texttt{"}151 : Nameclashforrenaming\texttt{"};$

1475.0 $PARMOD\text{-}NOT\text{-}DEFINED = \texttt{"}152 : Parameterisedmoduleisnotdefined\texttt{"};$

1476.0 $NOT\text{-}FULLY\text{-}INST = \texttt{"}153 : Parameterisedmoduleisnotfullyinstantiated\texttt{"};$

1477.0 $TYPE\text{-}ALREADY\text{-}DEF = \texttt{"}154 : Thetypeisalreadydefined\texttt{"};$

1478.0 $MOD\text{-}NOT\text{-}DEFINED = \texttt{"}155 : Themoduleisnotdefined\texttt{"};$

1479.0 $TYPE\text{-}NOT\text{-}EXPORTED = \texttt{"}156 : Thetypeisnotexported\texttt{"};$

1480.0 $CONSTRUCT\text{-}NOT\text{-}EXPORTED = \texttt{"}157 : Theconstructisnotexported\texttt{"};$

1481.0 $WRONG\text{-}STATE\text{-}TYPE = \texttt{"}158 : Wrongstatetype\texttt{"};$

1482.0 $NOT\text{-}DEFINED\text{-}IN\text{-}MOD =$ "159 : *Incorrectuseof'using'*";

1483.0 $LIB\text{-}NOT\text{-}DEFINED =$ "160 : *couldn'topenlibrary* : ";

1484.0 $LIB\text{-}SYMBOL\text{-}NOT\text{-}DEFINED =$ "161 : *tokennotdefinedinlibrary*";

1485.0 $LIB\text{-}ARGUMENT\text{-}WRONG\text{-}NUMBER =$ "162:*actualnumberofargumentdidn'tmatchthenumbertheexternfunc*

1486.0 $LIB\text{-}NOT\text{-}DECLARED =$ "163 : *Notdeclaredindlmodule*";

1487.0 $INTERNAL\text{-}ERROR =$ "164 : *Internalerror, pleasereport*";

1488.0 $LIB\text{-}TYPE\text{-}ERROR =$ "165 : *Incompatibletypeindlmodulecall*";

1489.0 $TYPE\text{-}NOT\text{-}SUPPORTED =$ "166 : *Typeisnotyetsupported*";

1490.0 $LIB\text{-}CLOSE\text{-}ERR =$ "167 : *Couldn'tclosedynamiclibrary*";

1491.0 $TAG\text{-}NOT\text{-}IN\text{-}NAME\text{-}MAP =$ "168:*NamenotdefinedinNameMapindynamiclibraryfile*";

1492.0 $LIB\text{-}WRONG\text{-}SIGN =$ "169 : *Libraryfunctionhaswrongsignature*";

1493.0 $FILE\text{-}DOES\text{-}NOT\text{-}EXISTS =$ "170 : *LibraryNamenotfoundinthesearchpath*";

1494.0 $CAN\text{-}NOT\text{-}BE\text{-}EVALUATED =$ "171 : *Externfunctioncannotbeevaluated*";

1495.0 $FN\text{-}EXIT\text{-}CONT =$ "172 : *Functionexitedorreturnednoresult*";

1496.0 $LIB\text{-}VERSION\text{-}NOT\text{-}FOUND =$ "173:*SymbolVDMLibVersionnotfoundindynamiclib*";

1497.0 $LIB\text{-}WRONG\text{-}VERSION =$ "174 : *VersionofVDMC + +library*";

1498.0 *EXTENDED-FCT-EXPECTED* = "175 : *Extendedfunctionoroperationexpected*";

1499.0 *UNEXPECTED-INFLOW* = "176 : *Unexpectedinputflow*";

1500.0 *COUND-NOT-MATCH-OUTFLOW* = "177:*Returnvaluecouldnotmatchoutputflows*";

1501.0 *NONEMPTYSET-EXPECTED* = "200 : *Anonemptysetwasexpected*";

1502.0 *NUMBER-ARG-NEQ-NUMBER-TYPE* = "201:*Numberofargumentsdoesnotmatchnumbertypedomaininfunct*

1503.0 *TYPE-INCOMP-RETURN* = "202:*Incompatiblereturntypeinfunctionoroperationapplication*";

1504.0 *TYPE-INCOMP-APPLY* = "203:*Incompatibletypeinvariablesinfunctionoroperationapplication*";

1505.0 *SET-EXP-IN-PATTERN-BIND* = "204:*SetExpectedinpatternbindinDefineExpression*";

1506.0 *VALUE-NOT-IN-SETBIND* = "205 : *ValueinDefExpressionisnotinSetBind*";

1507.0 *MTHD-EXIT-CONT* = "206 : *Methodexitedorreturnednoresult*";

1508.0 *OBJ-REF-EXP* = "207 : *Anobjectreferencewasexpectedintheexpression*";

1509.0 *CLNM-NOT-DEFINED* = "208 : *Classnameisnotdefined*";

1510.0 *TAG-UNKNOWN* = "209 : *Tagisunknownwithinthecurrentscope*";

1511.0 *MULT-DEF-METHS* = "211:*Themethodnameismultipledefinedwithinthecurrentscope*";

1512.0 *MULT-DEF-FCTS* = "212:*Thefunctionnameismultipledefinedwithinthecurrentscope*";

1513.0 *MULT-INST-VARS* = "213:*Theinstancevariablenameismultipledefinedwithinthecurrentscope*";

1514.0 *MULT-VAL-IDS* = "214 : *Theidentifierismultipledefinedasavalueinthecurrentscope*";

1515.0 *TOPOLOGY-STMT* = "215 : *Cannotevaluatetopologystatement*";

1516.0 *SPEC-STMT* = "216 : *Cannotevaluatespecificationstatement*";

1517.0 *INST-ABS-CL* = "217 : *Cannotinstantiateanabstractclass*";

1518.0 *NOT-CL-NAME-IN-NEW-STMT* = "218 : *Unknownclassnameinnewstatement*";

1519.0 *OBJ-REF-EXP-CALL* = "219 : *Anobjectreferencewasexpectedincallstatement*";

1520.0 *NOT-MTHD-NAME* = "220 : *Anameofafullmethodwasexpectedininvokestatement*";

1521.0 *INST-INV-BROKEN* = "221 : *Instanceinvariantwasbroken*";

1522.0 *IND-INH-NOT-SUPPORTED* = "222 : *Indexedinheritanceisnotsupported*";

1523.0 *UNEXP-RET-VAL-INIT-CL* = "223 : *Initialisationstatementreturnedavalue*";

1524.0 *CIRC-CL-DEPENDENCY* = "224 : *Circularinheritancedependecydetected*";

1525.0 *MULT-TPS-NAME* = "225:*Multipledefinedtypeswiththesamenamewithincurrentscope*";

1526.0 *DB-OBJ-EXISTS* = "226:*Theobjectnamealreadyexists.Pleasedestroytheobjectbeforecreatinganewobjectofthesa*

1527.0 *DB-OBJ-NOT-EXISTS* = "227 : *Thenameisnotanobject*";

1528.0 *OBJ-RECORD-EXP* = "228 : *Anobjectorrecordwasexpected*";

1529.0 $CIRC\text{-}CL\text{-}INIT = \texttt{"}229 : Circulardependencyininitialisationdetected\texttt{"};$

1530.0 $NAME\text{-}UNKNOWN = \texttt{"}230 : Nameunknown\texttt{"};$

1531.0 $DC\text{-}NOT\text{-}PATTERN\text{-}NAME = \texttt{"}231{:}Anabstractfieldofarecordmustonlypatternmatchwithapatternname\texttt{"};$

1532.0 $LOCAL\text{-}COMPOSE\text{-}TYPEDEF = \texttt{"}232{:}Theinterpreterdoesnotsupportlocaltypedefinitionsofrecords\texttt{"};$

1533.0 $NOTYETSPECFCT = \texttt{"}233 : Cannotevaluate`notyetspecified'functions\texttt{"};$

1534.0 $NOTYETSPECOP = \texttt{"}234 : Cannotevaluate`notyetspecified'operations\texttt{"};$

1535.0 $BUG\text{-}263 = \texttt{"}236 : Youhashitbugnumber263, pleaseseethebugreportforawork\text{-}around\texttt{"};$

1536.0 $EXIT\text{-}IN\text{-}INIT = \texttt{"}237 : Exitvaluereturnedininitialisationofinstancevariable\texttt{"};$

1537.0 $SUBRESP = \texttt{"}238 : Cannotevaluate`subresponsible'functions\texttt{"};$

1538.0 $NUMERIC\text{-}SET = \texttt{"}239{:}Quantificationinsequencecomprehensionmustbeovernumericvalues\texttt{"};$

1539.0 $WRONG\text{-}NO\text{-}RES = \texttt{"}240 : Wrongnumberofresults\texttt{"};$

1540.0 $LOWER\text{-}BOUND\text{-}NOT\text{-}A\text{-}NUMBER = \texttt{"}241 : Lowerboundisnotanumber\texttt{"};$

1541.0 $UPPER\text{-}BOUND\text{-}NOT\text{-}A\text{-}NUMBER = \texttt{"}242 : Upperboundisnotanumber\texttt{"};$

1542.0 $STEP\text{-}NOT\text{-}A\text{-}NUMBER = \texttt{"}243 : Stepisnotanumber\texttt{"};$

1543.0 $UPPER\text{-}BOUND\text{-}LARGER\text{-}THAN\text{-}LOWER\text{-}BOUND = \texttt{"}244{:}Lowerboundlargerthanupperbound\texttt{"};$

1544.0 *LOWER-BOUND-LARGER-THAN-UPPER-BOUND* = "245:*Lowerboundlargerthanupperbound*";

1545.0 *STEP-INDEX-IS-ZERO* = "246 : *Stepindexiszero*";

1546.0 *LOOP-EXPR-NOT-AN-EXPR* = "247:*Expressioninwhile-statmentdoesnotevaluatetoanexpression*";

1547.0 *TEST-EXPR-NOT-AN-EXPR* = "248:*Testexpressioninif-statmentdoesnotevaluatetoanexpression*";

1548.0 *TUPLE-OUTSIDE-INDEX* = "249 : *Tupleselectionoutsideitsindex*";

1549.0 *INSTVAR-NOT-PUBLIC* = "250 : *Instancevariablemustbepublic*";

1550.0 *INSTVAR-NOT-IN-SCOPE* = "251 : *Instancevariableisnotinscope*";

1551.0 *FUN-NOT-IN-SCOPE* = "252 : *Functionisnotinscope*";

1552.0 *OP-NOT-IN-SCOPE* = "253 : *Operationisnotinscope*";

1553.0 *VAL-NOT-IN-SCOPE* = "254 : *Valueisnotinscope*";

1554.0 *POLYFUN-NOT-IN-SCOPE* = "255 : *Polymorphicfunctionisnotinscope*";

1555.0 *TYPE-NOT-IN-SCOPE* = "256 : *Typeisnotinscope*";

1556.0 *NOT-IN-SCOPE* = "257 : *Constructnotinscope*";

1557.0 *MULT-DEF* = "258 : *Constructismultipledefinedwithinthecurrentscope*";

1558.0 *INIT-NOT-POSSIBLE* = "259 : *Initialisationisnotpossible*";

1559.0 *INDEXED-ASSIGN* = "260 : *Indexedassignmentcanonlybedonewheninitialised*";

1560.0 *OP-IN-GUARD* = "261 : *Youcannotuseanoperationinsideapermissionguard*";

1561.0 *NO-THREAD* = "262 : *Thisclasshavenothreadtostart*";

1562.0 *DEADLOCK-DETECTED* = "263 : *Deadlockisdetected*";

1563.0 *PRE-COND-APPLY-EXPR* = "264: *Thespecialpre-conditionapplicationexpressionisnotsupportedintheinterpr*

1564.0 *NO-GUARD-IN-INIT* = "265 : *Anoperationwithapermissionguardmaynot*"
.1 "*beusedintheinitialisationofaninstancevariable*";

1565.0 *OP-OR-FUN-NOT-IN-SCOPE* = "266 : *Operationorfunctionisnotinscope*";

1566.0 *OP-DEF-IN-MULTSUPERS* = "267: *Operationdefinedinmultiplesuperclasses*, *soitisnotalllowedtoaddpermissio*

1567.0 *SEL-NONVALID-THREADID* = "268 : *Anon-validthreadidwasused*";

1568.0 *MULT-THREAD-INH* = "269 : *Morethanonethreadinherited*";

1569.0 *LOGDLCALL* = "279 : *Logofdlcall* : ";

1570.0 *NOCONSTRUCTOR* = "280 : *Noconstructorwiththisparameterlistisinscope*";

1571.0 *MULTIOVERLOADED* = "281 : *Unabletoresolveoverloadedoperationcall*";

1572.0 *STATIC-NOT-IN-SCOPE* = "282 : *Staticmemberisnotinscope*";

1573.0 *STATIC-IV-NO-VALUE* = "283 : *Staticinstancevariablemustbeinitialised*";

1574.0 *CANNOT-PROCEED-AFTER-RUNTIME-ERROR* = "284: *Cannotproceedafteraruntimeerror*.";

1575.0 $NOOVERLOADED$ = "285:$Nooverloadedoperationorfunctionwiththisparameterlistisinscope$";

1576.0 $NOOBJECT$ = "286 : $Noobjectispresent$"

end $RTERR$

Test Suite : rtinfo.ast
Module : RTERR

Name	#Calls	Coverage
RTERR‘Error	**undefined**	**undefined**
RTERR‘Num2Str	**undefined**	**undefined**
RTERR‘ErrorVal	**undefined**	**undefined**
RTERR‘GetError	**undefined**	**undefined**
RTERR‘SetError	**undefined**	**undefined**
RTERR‘InitError	**undefined**	**undefined**
RTERR‘ConvertChar	**undefined**	**undefined**
Total Coverage		**0%**

1.24 The Debugger module

The module *DEBUG* specifies the debug commands features available in the debugger.
module $DEBUG$

imports

1577.0 from AS all ,

1578.0 from CI all ,

1579.0 from AUX all ,

1580.0 from PAT all ,

1581.0 from REP all ,

1582.0 from SEM all ,

1583.0 from $CMPL$ all ,

1584.0 from $SCHD$ all ,

1585.0 from $STKM$ all ,

1586.0 from $TIME$ all ,

1587.0 from $CEXPR$ all ,

1588.0 from $CSTMT$ all ,

1589.0 from $RTERR$ all ,

1590.0 from $STATE$ all ,

1591.0 from $GLOBAL$ all ,

1592.0 from $SCHDTP$ all ,

1593.0 from $DEBUGTP$ all ,

1594.0 from $INSTRTP$ all ,

1595.0 from $TIMEMAP$ all ,

1596.0 from $SETTINGS$ all ,

1597.0 from $TIMEPARSER$ all

exports all

definitions

1.24.1 EvalDebug, EvalPrint, and aux. functions for these

1598.0 state $sigma$ of
.1 $breakpoints : \mathbb{N} \xrightarrow{m} DEBUGTP`BreakInfo$
.2 $inActiveLevel : \mathbb{N}$
.3 $breakId : \mathbb{N}$
.4 init $s \triangleq s = \textsf{mk-}sigma\,(\{\mapsto\}, 0, 0)$
.5 end

operations

1599.0 $EvalDebug : (AS`Expr \mid AS`Stmt) \times \textsf{char}^* \xrightarrow{o} STKM`EvaluationState \times [SEM`VAL]$
.1 $EvalDebug\,(e, debugString) \triangleq$
.2 $(ResetInActivity()\,;$
.3 return $EvalPrintDebugAux\,(e, debugString)\,)$;

1600.0 $EvalPrintDebugAux : (AS`Expr \mid AS`Stmt) \times \mathsf{char}^* \xrightarrow{o} STKM`EvaluationState \times [SEM`VAL]$

.1 $EvalPrintDebugAux\,(e, debugString) \triangleq$
.2 $\quad \mathsf{let}\ instr : STKM`SubProgram = \mathsf{if}\ AUX`IsStmt\,(e)$
.3 $\qquad \mathsf{then}\ CSTMT`S2I\,(e)$
.4 $\qquad \mathsf{else}\ CEXPR`E2I\,(e)\ \mathsf{in}$
.5 $\quad (STKM`SetContinue()\,;$
.6 $\quad\ STKM`PushDS(STKM`GetEvaluatorState\,(), debugString, \mathsf{mk\text{-}}STKM`DebugCmd\,(instr));$
.7 $\quad\ \mathsf{return}\ EvalRun\,(\mathsf{true})\ );$

The operation *EvalRun* calls for VDM-SL *EvalMainLoop* with nil, and for VDM++ *EvalScheduler*. The operation is an auxiliary operation:

1601.0 $EvalRun : [\mathbb{B}] \xrightarrow{o} STKM`EvaluationState \times [SEM`VAL]$

.1 $EvalRun\,(main) \triangleq$
.2 $\quad (STKM`ResetSlice()\,;$
.3 $\quad\ \mathsf{let\ mk\text{-}}\,(eval\text{-}state, semval) = SCHD`EvalScheduler\,()\ \mathsf{in}$
.4 $\quad\ (\mathsf{if\ is\text{-}}STKM`Success\,(eval\text{-}state)$
.5 $\quad\ \ \mathsf{then}\ STKM`PopCS()\,;$
.6 $\quad\ \ \mathsf{return\ mk\text{-}}\,(eval\text{-}state, semval)\ ));$

1602.0 $EvalAuxCmd : AS`Expr \times STKM`SubProgram \times STKM`SubProgram \times \mathsf{char}^* \xrightarrow{o} STKM`EvaluationState \times [SEM`VAL]$

.1 $EvalAuxCmd\,(e, instr\text{-}pre, instr\text{-}post, debugStr) \triangleq$
.2 $\quad \mathsf{let}\ instr : STKM`SubProgram = instr\text{-}pre \curvearrowright CEXPR`E2I\,(e) \curvearrowright instr\text{-}post\ \mathsf{in}$
.3 $\quad (DeActivateAllBreakpoints()\,;$
.4 $\quad\ STKM`PushCS(\mathsf{mk\text{-}}STKM`DebugCmd\,(instr), debugStr, \mathsf{nil}\ , \mathrm{INTERNAL})\,;$
.5 $\quad\ STKM`ResetSlice()\,;$
.6 $\quad\ \mathsf{let\ mk\text{-}}\,(eval\text{-}state, semval) = STKM`EvalMainLoop\,()\ \mathsf{in}$
.7 $\quad\ (ActivateAllBreakpoints()\,;$
.8 $\quad\ \ \mathsf{if\ is\text{-}}STKM`Success\,(eval\text{-}state)$
.9 $\quad\ \ \mathsf{then}\ STKM`PopCS()\,;$
.10 $\quad\ \ \mathsf{return\ mk\text{-}}\,(eval\text{-}state, semval)\ ));$

1603.0 $\mathit{EvalUninterruptedCmd} : \mathit{AS}`\mathit{Expr} \times \mathit{STKM}`\mathit{SubProgram} \times \mathit{STKM}`\mathit{SubProgram} \times \mathsf{char}^* \xrightarrow{o}$
.1 $\mathit{STKM}`\mathit{EvaluationState} \times \mathit{SEM}`\mathit{VAL}$
.2 $\mathit{EvalUninterruptedCmd}\,(e, \mathit{instr\text{-}pre}, \mathit{instr\text{-}post}, \mathit{debugStr}) \triangleq$
.3 let $\mathit{old\text{-}compiling} = \mathit{TIME}`\mathit{GetCompilingTime}\,()$ in
.4 $(\mathit{TIME}`\mathit{SetCompilingTime}(\mathsf{false})$;
.5 let $\mathit{instr} : \mathit{STKM}`\mathit{SubProgram} = \mathit{instr\text{-}pre} \curvearrowright \mathit{CEXPR}`\mathit{E2I}\,(e) \curvearrowright \mathit{instr\text{-}post}$ in
.6 $(\mathit{DeActivateAllBreakpoints}()$;
.7 $\mathit{STKM}`\mathit{PushCS}(\mathsf{mk\text{-}}\mathit{STKM}`\mathit{DebugCmd}\,(\mathit{instr}), \mathit{debugStr}, \mathsf{nil}, \mathrm{INTERNAL})$;
.8 let mk- $(\mathit{eval\text{-}state}, \mathit{semval}) = \mathit{STKM}`\mathit{EvalUninterruptedLoop}\,()$ in
.9 $(\mathit{ActivateAllBreakpoints}()$;
.10 $\mathit{STKM}`\mathit{PopCS}()$;
.11 return mk- $(\mathit{eval\text{-}state}, \mathit{semval})\ ))$;
.12 $\mathit{TIME}`\mathit{SetCompilingTime}(\mathit{old\text{-}compiling})\ )$;

1604.0 $\mathit{EvalPrint} : (\mathit{AS}`\mathit{Expr} \mid \mathit{AS}`\mathit{Stmt}) \times \mathsf{char}^* \xrightarrow{o} \mathit{STKM}`\mathit{EvaluationState} \times [\mathit{SEM}`\mathit{VAL}]$
.1 $\mathit{EvalPrint}\,(e, \mathit{debugString}) \triangleq$
.2 $(\mathit{ResetInActivity}()$;
.3 $\mathit{DeActivateAllBreakpoints}()$;
.4 let $\mathit{res} = \mathit{EvalPrintDebugAux}\,(e, \mathit{debugString})$ in
.5 $(\mathit{ActivateAllBreakpoints}()$;
.6 return $\mathit{res}\ ))$;

1.24.2 Stepping commands

1605.0 $\mathit{EvalStep} : () \xrightarrow{o} \mathbb{B} \times [\mathit{STKM}`\mathit{EvaluationState}] \times [\mathit{SEM}`\mathit{VAL}]$
.1 $\mathit{EvalStep}\,() \triangleq$
.2 $(\mathit{STKM}`\mathit{SetStep}()$;
.3 return $\mathit{RunIfAllowed}\,()\ )$
.4 pre is-$\mathit{STKM}`\mathit{Breakpoint}\,(\mathit{STKM}`\mathit{MainLoopState}\,())$;

1606.0 $\mathit{EvalStepIn} : () \xrightarrow{o} \mathbb{B} \times [\mathit{STKM}`\mathit{EvaluationState}] \times [\mathit{SEM}`\mathit{VAL}]$
.1 $\mathit{EvalStepIn}\,() \triangleq$
.2 $(\mathit{STKM}`\mathit{SetStepIn}()$;
.3 return $\mathit{RunIfAllowed}\,()\ )$
.4 pre is-$\mathit{STKM}`\mathit{Breakpoint}\,(\mathit{STKM}`\mathit{MainLoopState}\,())$;

1607.0 $\mathit{EvalSingleStep} : () \xrightarrow{o} \mathbb{B} \times [\mathit{STKM}`\mathit{EvaluationState}] \times [\mathit{SEM}`\mathit{VAL}]$
.1 $\mathit{EvalSingleStep}\,() \triangleq$
.2 $(\mathit{STKM}`\mathit{SetSingleStep}()$;
.3 return $\mathit{RunIfAllowed}\,()\ )$

.4 $\textsf{pre}\ \textsf{is-}STKM`Breakpoint\,(STKM`MainLoopState\,())\ ;$

1608.0 $EvalContinue : () \xrightarrow{o} \mathbb{B} \times [STKM`EvaluationState] \times [SEM`VAL]$

.1 $EvalContinue\,() \triangleq$

.2 $(STKM`SetContinue()\,;$

.3 $\textsf{return}\ RunIfAllowed\,()\ )$

.4 $\textsf{pre}\ \textsf{is-}STKM`Breakpoint\,(STKM`MainLoopState\,())\ ;$

1609.0 $EvalFinish : () \xrightarrow{o} \mathbb{B} \times [STKM`EvaluationState] \times [SEM`VAL]$

.1 $EvalFinish\,() \triangleq$

.2 $(STKM`SetFinish()\,;$

.3 $\textsf{return}\ RunIfAllowed\,()\ )$

.4 $\textsf{pre}\ \textsf{is-}STKM`Breakpoint\,(STKM`MainLoopState\,())\ ;$

1610.0 $IsSteppingAllowed : () \xrightarrow{o} \mathbb{B}$

.1 $IsSteppingAllowed\,() \triangleq$

.2 $\textsf{let}\ allow1 = STKM`CallStackLevel\,() > 0,$

.3 $allow2 = \neg\, STKM`IsProgramAtEnd\,(),$

.4 $St = STKM`MainLoopState\,(),$

.5 $allow3 = \textsf{is-}STKM`Interrupt\,(St) \vee \textsf{is-}STKM`Breakpoint\,(St)\ \textsf{in}$

.6 $\textsf{return}\ allow1 \vee allow2 \vee allow3;$

1611.0 $RunIfAllowed : () \xrightarrow{o} \mathbb{B} \times [STKM`EvaluationState] \times [SEM`VAL]$

.1 $RunIfAllowed\,() \triangleq$

.2 $(STKM`ResetUpDn()\,;$

.3 $\textsf{if}\ IsSteppingAllowed\,()$

.4 $\textsf{then let mk-}\,(eval\text{-}state, semval) = EvalRun\,(\textsf{false})\ \textsf{in}$

.5 $\textsf{return mk-}\,(\textsf{true}, eval\text{-}state, semval)$

.6 $\textsf{else return mk-}\,(\textsf{false}, \textsf{nil}\ , \textsf{nil}\ )\ );$

1.24.3 Breakpoints

All the following commands relates to setting/removing breakpoints and enabling/disabling breakpoints.

1612.0 $EvalBreakName : AS`Name \times [\mathbb{N}] \times DEBUGTP`BreakStatus \xrightarrow{o} \mathbb{B} \times \textsf{char}^* \times [\mathbb{N}]$

.1 $EvalBreakName\,(name, id, status) \triangleq$

.2 $(\textsf{dcl}\ id' : \mathbb{N} := id;$

```
.3       let mk- (isit, errMsg, cid, modcl, loc-nm) = STATE`GetFirstCIDOfFctOp (name) in
.4       if isit
.5       then (if id = nil
.6              then (breakId := breakId + 1;
.7                    id′ := breakId);
.8              breakpoints := breakpoints †
.9                              {id′ ↦
.10                              mk-DEBUGTP`FnNameBreakInfo (modcl, loc-nm, status, cid)};
.11             if is-DEBUGTP`Enabled (status)
.12             then CI`SetBreakpoint(cid) ;
.13             return mk- (true, "", id) )
.14      else return mk- (false, errMsg, nil ) );
```

```
1613.0  EvalBreakPos : char* × ℕ × ℕ × [ℕ] × DEBUGTP`BreakStatus →ᵒ
.1                     𝔹 × char* × [ℕ]
.2      EvalBreakPos (fileName, line, col, id, status) △
.3        (dcl id′ : ℕ := id;
.4         let mk- (found, errMsg, cid) = CI`GetCidAtPos (fileName, line, col) in
.5         if found
.6         then let breakable = CI`IsCidBreakable (cid) in
.7              (if id = nil
.8               then (breakId := breakId + 1;
.9                     id′ := breakId);
.10              breakpoints := breakpoints †
.11                             {id′ ↦
.12                              mk-DEBUGTP`PosBreakInfo (fileName, line, col, status, cid)};
.13              if is-DEBUGTP`Enabled (status)
.14              then CI`SetBreakpoint(cid) ;
.15              return mk- (true, "", id) )
.16      else return mk- (false, errMsg, nil ) );
```

```
1614.0  DeleteBreakpoint : ℕ →ᵒ 𝔹 × [char*]
.1      DeleteBreakpoint (num) △
.2        if ¬ num ∈ dom breakpoints
.3        then return mk- (false, "Nosuchbreakpoint")
.4        else let info = breakpoints (num) in
.5             (CI`RemoveBreakpoint(info.cid) ;
.6              breakpoints := {num} ⩤ breakpoints;
.7              return mk- (true, nil ) );
```

When the user reinitializes his specifications, all breakpoints must be set once again. The reason for this is that the locations of the breakpoints will have moved to new context ids in case that the specification has changed.

It is not necessary to remove the break information from the context ids, as one of two situations may present:

- The specification has not changed, and the context id is at the same location as it was before the reinitialization.
- The specification has changed and the context table for the given file have been deleted and rebuild.

1615.0 $\mathit{UpdateBreakpoint} : () \xrightarrow{o} \mathbb{N}\text{-set}$

.1 $\mathit{UpdateBreakpoint}\,() \triangleq$

.2 $\quad \mathsf{let}\ \mathit{ids} = \{\mathit{id} \mid \mathit{id} \in \mathsf{dom}\ \mathit{breakpoints} \cdot$

.3 $\qquad\qquad \mathsf{let}\ \mathit{info} = \mathit{breakpoints}\,(\mathit{id})\ \mathsf{in}$

.4 $\qquad\qquad \neg\,(\mathsf{if}\ \mathsf{is}\text{-}\mathit{DEBUGTP}`\mathit{FnNameBreakInfo}\,(\mathit{info})$

.5 $\qquad\qquad\ \ \mathsf{then}\ \mathit{UpdateFnNameBreakInfo}\,(\mathit{id}, \mathit{info})$

.6 $\qquad\qquad\ \ \mathsf{else}\ \mathit{UpdatePosBreakInfo}\,(\mathit{id}, \mathit{info}))\}\ \mathsf{in}$

.7 $\quad (\mathit{breakpoints} := \mathit{ids} \lhd \mathit{breakpoints};$

.8 $\quad\ \mathsf{return}\ \mathit{ids}\ );$

1616.0 $\mathit{UpdateFnNameBreakInfo} : \mathbb{N} \times \mathit{DEBUGTP}`\mathit{FnNameBreakInfo} \xrightarrow{o} \mathbb{B}$

.1 $\mathit{UpdateFnNameBreakInfo}\,(\mathit{id}, \mathsf{mk}\text{-}\mathit{DEBUGTP}`\mathit{FnNameBreakInfo}\,(\mathit{modcl}, \mathit{name}, \mathit{status}, \text{-})) \triangleq$

.2 $\quad \mathsf{let}\ \mathsf{mk}\text{-}(\mathit{ok}, \text{-}, \text{-}) = \mathit{EvalBreakName}\,(\mathit{AUX}`\mathit{ConstructDoubleName}\,(\mathit{modcl}, \mathit{name}), \mathit{id}, \mathit{status})\ \mathsf{in}$

.3 $\quad \mathsf{return}\ \mathit{ok};$

1617.0 $\mathit{UpdatePosBreakInfo} : \mathbb{N} \times \mathit{DEBUGTP}`\mathit{PosBreakInfo} \xrightarrow{o} \mathbb{B}$

.1 $\mathit{UpdatePosBreakInfo}\,(\mathit{id}, \mathsf{mk}\text{-}\mathit{DEBUGTP}`\mathit{PosBreakInfo}\,(\mathit{fileName}, \mathit{line}, \mathit{col}, \mathit{status}, \text{-})) \triangleq$

.2 $\quad \mathsf{let}\ \mathsf{mk}\text{-}(\mathit{ok}, \text{-}, \text{-}) = \mathit{EvalBreakPos}\,(\mathit{fileName}, \mathit{line}, \mathit{col}, \mathit{id}, \mathit{status})\ \mathsf{in}$

.3 $\quad \mathsf{return}\ \mathit{ok};$

1.24.4 Query Functions

1618.0 $\mathit{ActiveBreakpoint} : \mathit{CI}`\mathit{ContextId} \xrightarrow{o} \mathbb{B}$

.1 $\mathit{ActiveBreakpoint}\,(\mathit{cid}) \triangleq$

.2 $\quad \mathsf{return}\ \mathit{inActiveLevel} = 0 \wedge$

.3 $\qquad\quad \mathit{CI}`\mathit{IsBreakpointAtCid}\,(\mathit{cid});$

1619.0 $\mathit{ExistsBreakpointForName} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{Name} \xrightarrow{o} \mathbb{B}$

.1 $\mathit{ExistsBreakpointForName}\,(\mathit{modClName}, \mathit{name}) \triangleq$

.2 $\quad \mathsf{return}\ \exists\, \mathit{id} \in \mathsf{dom}\ \mathit{breakpoints} \cdot$

.3 $\qquad\qquad \mathsf{is}\text{-}\mathit{DEBUGTP}`\mathit{FnNameBreakInfo}\,(\mathit{breakpoints}\,(\mathit{id})) \wedge$

.4 $\qquad\qquad \mathit{breakpoints}\,(\mathit{id}).\mathit{modcl} = \mathit{modClName} \wedge$

.5 $\qquad\qquad \mathit{breakpoints}\,(\mathit{id}).\mathit{name} = \mathit{name};$

```
1620.0 ExistsBreakpointForPos : char* × ℕ × ℕ →(o) 𝔹
    .1 ExistsBreakpointForPos (fileName, line, col) ≜
    .2   return ∃ id ∈ dom breakpoints ·
    .3              is-DEBUGTP‘PosBreakInfo (breakpoints (id)) ∧
    .4              breakpoints (id).fileName = fileName ∧
    .5              breakpoints (id).line = line ∧
    .6              breakpoints (id).col = col;
```

```
1621.0 GetBreakpointNumForName : AS‘Name × AS‘Name →(o) ℕ
    .1 GetBreakpointNumForName (modClName, name) ≜
    .2   let {num} = {id | id ∈ dom breakpoints ·
    .3                    breakpoints (id).modcl = modClName ∧
    .4                    breakpoints (id).name = name} in
    .5   return num
    .6 pre  card {id | id ∈ dom breakpoints ·
    .7              breakpoints (id).modcl = modClName ∧
    .8              breakpoints (id).name = name} =
    .9     1 ;
```

```
1622.0 GetBreakpointNumForPos : char* × ℕ × ℕ →(o) ℕ
    .1 GetBreakpointNumForPos (fileName, line, col) ≜
    .2   let {num} = {id | id ∈ dom breakpoints ·
    .3                    breakpoints (id).fileName = fileName ∧
    .4                    breakpoints (id).line = line ∧
    .5                    breakpoints (id).col = col} in
    .6   return num
    .7 pre  card {id | id ∈ dom breakpoints ·
    .8              breakpoints (id).fileName = fileName ∧
    .9              breakpoints (id).line = line ∧
   .10              breakpoints (id).col = col} =
   .11     1 ;
```

Enable/disable

```
1623.0 EnableBreakpoint : ℕ →(o) 𝔹 × [char*]
    .1 EnableBreakpoint (num) ≜
    .2   if ¬ num ∈ dom breakpoints
    .3   then return mk- (false, "Nosuchbreakpoint")
    .4   else let info = breakpoints (num) in
    .5        if info.status = mk-DEBUGTP‘Enabled ()
    .6        then return mk- (false, "Breakpointisalreadyenabled")
```

```
.7          else (CI`SetBreakpoint(info.cid) ;
.8              breakpoints(num) := μ (info,
.9
.10                                     status ↦ mk-DEBUGTP`Enabled ());
.11             return mk- (true, nil ) );
```

```
1624.0  DisableBreakpoint : ℕ -o-> 𝔹 × [char*]
.1      DisableBreakpoint (num) △
.2        if ¬ num ∈ dom breakpoints
.3        then return mk- (false, "Nosuchbreakpoint")
.4        else let info = breakpoints (num) in
.5            if info.status = mk-DEBUGTP`Disabled ()
.6            then return mk- (false, "Breakpointisalreadydisabled")
.7            else (CI`RemoveBreakpoint(info.cid) ;
.8                breakpoints(num) := μ (info,
.9
.10                                       status ↦ mk-DEBUGTP`Disabled ());
.11               return mk- (true, nil ) );
```

Activate/deactivate

When a print command has been issued, or when an internal recursive commands has been issued, then the toolbox should not break at any breakpoint.

An example of a recursive command is evaluation of invariants or permission predicates. (It is the commands which called by the EvalAuxCmd).

Several levels of recursive invocation of the EvalAuxCmd may be invoked. This means that it is not enough with just a boolean flag to indicate wether breakpoints may be used or not.

```
1625.0  ResetInActivity : () -o-> ()
.1      ResetInActivity () △
.2        inActiveLevel := 0;
```

```
1626.0  ActivateAllBreakpoints : () -o-> ()
.1      ActivateAllBreakpoints () △
.2        (inActiveLevel := inActiveLevel − 1);
```

```
1627.0  DeActivateAllBreakpoints : () -o-> ()
.1      DeActivateAllBreakpoints () △
.2        (inActiveLevel := inActiveLevel + 1);
```

1.24.5 Up/down commands

1628.0 $EvalStackUp : () \xrightarrow{o} ()$

.1 $EvalStackUp\ () \triangleq$

.2 $\quad STKM`GoUp()\ ;$

1629.0 $EvalStackDown : () \xrightarrow{o} ()$

.1 $EvalStackDown\ () \triangleq$

.2 $\quad STKM`GoDown()\ ;$

1.24.6 Thread Related Debug Commands

The opeation *EvalThreads* returns the sequence of thread id available.

1630.0 $EvalThreads : () \xrightarrow{o} SCHDTP`ThreadId \xrightarrow{m} SCHDTP`ThreadInfo$

.1 $EvalThreads\ () \triangleq$

.2 $\quad SCHD`GiveAllThreads()\ ;$

1631.0 $EvalSelThread : SCHDTP`ThreadId \xrightarrow{o} ()$

.1 $EvalSelThread\ (id) \triangleq$

.2 $\quad SCHD`SelThread(id)\ ;$

1632.0 $EvalCurThread : () \xrightarrow{o} SCHDTP`ThreadId$

.1 $EvalCurThread\ () \triangleq$

.2 $\quad SCHD`CurThreadId()$

end *DEBUG*

Test Suite : rtinfo.ast
Module : DEBUG

Name	**#Calls**	**Coverage**
DEBUG‘EvalRun	**undefined**	**undefined**
DEBUG‘EvalStep	**undefined**	**undefined**
DEBUG‘EvalDebug	**undefined**	**undefined**
DEBUG‘EvalPrint	**undefined**	**undefined**
DEBUG‘EvalAuxCmd	**undefined**	**undefined**
DEBUG‘EvalFinish	**undefined**	**undefined**
DEBUG‘EvalStepIn	**undefined**	**undefined**

Name	**#Calls**	**Coverage**
DEBUG‘EvalStackUp	**undefined**	**undefined**
DEBUG‘EvalThreads	**undefined**	**undefined**
DEBUG‘EvalBreakPos	**undefined**	**undefined**
DEBUG‘EvalContinue	**undefined**	**undefined**
DEBUG‘RunIfAllowed	**undefined**	**undefined**
DEBUG‘EvalBreakName	**undefined**	**undefined**
DEBUG‘EvalCurThread	**undefined**	**undefined**
DEBUG‘EvalSelThread	**undefined**	**undefined**
DEBUG‘EvalStackDown	**undefined**	**undefined**
DEBUG‘EvalSingleStep	**undefined**	**undefined**
DEBUG‘ResetInActivity	**undefined**	**undefined**
DEBUG‘ActiveBreakpoint	**undefined**	**undefined**
DEBUG‘DeleteBreakpoint	**undefined**	**undefined**
DEBUG‘EnableBreakpoint	**undefined**	**undefined**
DEBUG‘UpdateBreakpoint	**undefined**	**undefined**
DEBUG‘DisableBreakpoint	**undefined**	**undefined**
DEBUG‘EvalPrintDebugAux	**undefined**	**undefined**
DEBUG‘IsSteppingAllowed	**undefined**	**undefined**
DEBUG‘UpdatePosBreakInfo	**undefined**	**undefined**
DEBUG‘EvalUninterruptedCmd	**undefined**	**undefined**
DEBUG‘UpdateFnNameBreakInfo	**undefined**	**undefined**
DEBUG‘ActivateAllBreakpoints	**undefined**	**undefined**
DEBUG‘ExistsBreakpointForPos	**undefined**	**undefined**
DEBUG‘GetBreakpointNumForPos	**undefined**	**undefined**
DEBUG‘ExistsBreakpointForName	**undefined**	**undefined**
DEBUG‘GetBreakpointNumForName	**undefined**	**undefined**
DEBUG‘DeActivateAllBreakpoints	**undefined**	**undefined**
Total Coverage		**0%**

1.25 Settings

This module define those settings which belongs to the interface, this is:

1. Should dynamic type checking be done
2. Should pre conditions be evaluated
3. Should post conditions be evaluated

module *SETTINGS*

imports

1633.0 from $SCHDTP$

1634.0 types $TimeSlice$;
.1 $InstrnumSlice$;
.2 $PureCooperative$;
.3 $PrimarySchedulerAlgorithm$

exports

1635.0 operations $DTC : () \xrightarrow{o} \mathbb{B}$;
.1 $INV : () \xrightarrow{o} \mathbb{B}$;
.2 $DtcOn : () \xrightarrow{o} ()$;
.3 $InvOn : () \xrightarrow{o} ()$;
.4 $PreOn : () \xrightarrow{o} ()$;
.5 $DtcOff : () \xrightarrow{o} ()$;
.6 $InvOff : () \xrightarrow{o} ()$;
.7 $PostOn : () \xrightarrow{o} ()$;
.8 $PreOff : () \xrightarrow{o} ()$;
.9 $Random : () \xrightarrow{o} \mathbb{Z}$;
.10 $PostOff : () \xrightarrow{o} ()$;
.11 $PreCheck : () \xrightarrow{o} \mathbb{B}$;
.12 $RandomOn : \mathbb{Z} \xrightarrow{o} ()$;
.13 $PostCheck : () \xrightarrow{o} \mathbb{B}$;
.14 $RandomOff : () \xrightarrow{o} ()$;
.15 $GetMaxInstr : () \xrightarrow{o} \mathbb{N}$;
.16 $SetMaxInstr : \mathbb{N} \xrightarrow{o} ()$;
.17 $GetTimeSlice : () \xrightarrow{o} \mathbb{N}$;
.18 $SetTimeSlice : \mathbb{N} \xrightarrow{o} ()$;
.19 $GetTaskSwitch : () \xrightarrow{o} \mathbb{N}$;
.20 $GetTimeFactor : () \xrightarrow{o} \mathbb{N}$;
.21 $PriorityBased : () \xrightarrow{o} \mathbb{B}$;
.22 $SetTaskSwitch : \mathbb{N} \xrightarrow{o} ()$;
.23 $SetTimeFactor : \mathbb{N} \xrightarrow{o} ()$;
.24 $PriorityBasedOn : () \xrightarrow{o} ()$;
.25 $PriorityBasedOff : () \xrightarrow{o} ()$;
.26 $GetPrimaryAlgorithm : () \xrightarrow{o} SCHDTP`PrimarySchedulerAlgorithm$;
.27 $SetPrimaryAlgorithm : SCHDTP`PrimarySchedulerAlgorithm \xrightarrow{o} ()$

definitions

Default no of the checks shall be set.

The following operations set the state variabes

1636.0 state $Settings$ of
.1 $\quad dtc : \mathbb{B}$
.2 $\quad precheck : \mathbb{B}$
.3 $\quad postcheck : \mathbb{B}$
.4 $\quad invar : \mathbb{B}$
.5 $\quad random : \mathbb{Z}$
.6 $\quad maxinstr : \mathbb{N}$
.7 $\quad maxTime : \mathbb{N}$
.8 $\quad priorityBased : \mathbb{B}$
.9 $\quad taskSwitch : \mathbb{N}$
.10 $\quad primaryAlgorithm : SCHDTP\text{‘}PrimarySchedulerAlgorithm$
.11 $\quad timeFactor : \mathbb{N}$
.12 $\quad$ init $s \triangleq s = \text{mk-}Settings\,(\mathsf{false}, \mathsf{false}, \mathsf{false}, \mathsf{false}, -1, 1000, 1000, \mathsf{false}, 0,$
.13 $\qquad\qquad\qquad \text{mk-}SCHDTP\text{‘}InstrnumSlice\,(), 1)$
.14 end

operations

1637.0 $DtcOn : () \xrightarrow{o} ()$
.1 $DtcOn\,() \triangleq$
.2 $\quad dtc := \mathsf{true};$

1638.0 $DtcOff : () \xrightarrow{o} ()$
.1 $DtcOff\,() \triangleq$
.2 $\quad dtc := \mathsf{false};$

1639.0 $InvOn : () \xrightarrow{o} ()$
.1 $InvOn\,() \triangleq$
.2 $\quad invar := \mathsf{true};$

1640.0 $InvOff : () \xrightarrow{o} ()$
.1 $InvOff\,() \triangleq$
.2 $\quad invar := \mathsf{false};$

1641.0 $PreOn : () \xrightarrow{o} ()$
.1 $PreOn\,() \triangleq$
.2 $\quad precheck := \mathsf{true};$

1642.0 $PreOff : () \xrightarrow{o} ()$
.1 $PreOff\,() \triangleq$
.2 $\quad precheck := \mathsf{false};$

1643.0 $PostOn : () \xrightarrow{o} ()$

.1 $PostOn\,() \triangleq$

.2 $postcheck := \mathsf{true};$

1644.0 $PostOff : () \xrightarrow{o} ()$

.1 $PostOff\,() \triangleq$

.2 $postcheck := \mathsf{false};$

1645.0 $RandomOn : \mathbb{Z} \xrightarrow{o} ()$

.1 $RandomOn\,(n) \triangleq$

.2 $random := n;$

1646.0 $RandomOff : () \xrightarrow{o} ()$

.1 $RandomOff\,() \triangleq$

.2 $random := -1;$

1647.0 $PriorityBasedOn : () \xrightarrow{o} ()$

.1 $PriorityBasedOn\,() \triangleq$

.2 $priorityBased := \mathsf{true};$

1648.0 $PriorityBasedOff : () \xrightarrow{o} ()$

.1 $PriorityBasedOff\,() \triangleq$

.2 $priorityBased := \mathsf{false};$

These functions read the values of the state variables

1649.0 $DTC : () \xrightarrow{o} \mathbb{B}$

.1 $DTC\,() \triangleq$

.2 $\mathsf{return}\ dtc;$

1650.0 $PreCheck : () \xrightarrow{o} \mathbb{B}$

.1 $PreCheck\,() \triangleq$

.2 $\mathsf{return}\ precheck;$

1651.0 $PostCheck : () \xrightarrow{o} \mathbb{B}$
.1 $PostCheck\,() \triangleq$
.2 $\quad \mathsf{return}\ postcheck;$

1652.0 $INV : () \xrightarrow{o} \mathbb{B}$
.1 $INV\,() \triangleq$
.2 $\quad \mathsf{return}\ invar;$

1653.0 $Random : () \xrightarrow{o} \mathbb{Z}$
.1 $Random\,() \triangleq$
.2 $\quad \mathsf{return}\ random;$

1654.0 $GetMaxInstr : () \xrightarrow{o} \mathbb{N}$
.1 $GetMaxInstr\,() \triangleq$
.2 $\quad \mathsf{return}\ maxinstr;$

1655.0 $SetMaxInstr : \mathbb{N} \xrightarrow{o} ()$
.1 $SetMaxInstr\,(no) \triangleq$
.2 $\quad maxinstr := no;$

1656.0 $PriorityBased : () \xrightarrow{o} \mathbb{B}$
.1 $PriorityBased\,() \triangleq$
.2 $\quad \mathsf{return}\ priorityBased;$

1657.0 $GetTaskSwitch : () \xrightarrow{o} \mathbb{N}$
.1 $GetTaskSwitch\,() \triangleq$
.2 $\quad \mathsf{return}\ taskSwitch;$

1658.0 $SetTaskSwitch : \mathbb{N} \xrightarrow{o} ()$
.1 $SetTaskSwitch\,(newTaskSwitch) \triangleq$
.2 $\quad taskSwitch := newTaskSwitch;$

1659.0 $GetTimeSlice : () \xrightarrow{o} \mathbb{N}$
.1 $GetTimeSlice\,() \triangleq$
.2 $\quad \mathsf{return}\ maxTime;$

1660.0 $SetTimeSlice : \mathbb{N} \xrightarrow{o} ()$

.1 $SetTimeSlice\,(newMaxTime) \triangleq$

.2 $\quad maxTime := newMaxTime;$

1661.0 $GetTimeFactor : () \xrightarrow{o} \mathbb{N}$

.1 $GetTimeFactor\,() \triangleq$

.2 $\quad$ return $timeFactor;$

1662.0 $SetTimeFactor : \mathbb{N} \xrightarrow{o} ()$

.1 $SetTimeFactor\,(newTimeFactor) \triangleq$

.2 $\quad timeFactor := newTimeFactor;$

1663.0 $GetPrimaryAlgorithm : () \xrightarrow{o} SCHDTP`PrimarySchedulerAlgorithm$

.1 $GetPrimaryAlgorithm\,() \triangleq$

.2 $\quad$ return $primaryAlgorithm;$

1664.0 $SetPrimaryAlgorithm : SCHDTP`PrimarySchedulerAlgorithm \xrightarrow{o} ()$

.1 $SetPrimaryAlgorithm\,(newPrimaryAlgorithm) \triangleq$

.2 $\quad primaryAlgorithm := newPrimaryAlgorithm$

end $SETTINGS$

Test Suite : rtinfo.ast
Module : SETTINGS

Name	#Calls	Coverage
SETTINGS‘DTC	**undefined**	**undefined**
SETTINGS‘INV	**undefined**	**undefined**
SETTINGS‘DtcOn	**undefined**	**undefined**
SETTINGS‘InvOn	**undefined**	**undefined**
SETTINGS‘PreOn	**undefined**	**undefined**
SETTINGS‘DtcOff	**undefined**	**undefined**
SETTINGS‘InvOff	**undefined**	**undefined**
SETTINGS‘PostOn	**undefined**	**undefined**
SETTINGS‘PreOff	**undefined**	**undefined**
SETTINGS‘Random	**undefined**	**undefined**
SETTINGS‘PostOff	**undefined**	**undefined**
SETTINGS‘PreCheck	**undefined**	**undefined**
SETTINGS‘RandomOn	**undefined**	**undefined**

Name	#Calls	Coverage
SETTINGS‘PostCheck	**undefined**	**undefined**
SETTINGS‘RandomOff	**undefined**	**undefined**
SETTINGS‘GetMaxInstr	**undefined**	**undefined**
SETTINGS‘SetMaxInstr	**undefined**	**undefined**
SETTINGS‘GetTimeSlice	**undefined**	**undefined**
SETTINGS‘SetTimeSlice	**undefined**	**undefined**
SETTINGS‘GetTaskSwitch	**undefined**	**undefined**
SETTINGS‘GetTimeFactor	**undefined**	**undefined**
SETTINGS‘PriorityBased	**undefined**	**undefined**
SETTINGS‘SetTaskSwitch	**undefined**	**undefined**
SETTINGS‘SetTimeFactor	**undefined**	**undefined**
SETTINGS‘PriorityBasedOn	**undefined**	**undefined**
SETTINGS‘PriorityBasedOff	**undefined**	**undefined**
SETTINGS‘GetPrimaryAlgorithm	**undefined**	**undefined**
SETTINGS‘SetPrimaryAlgorithm	**undefined**	**undefined**
Total Coverage		**0%**

1.26 Introduction

This module defines the abstract syntax for time files. A time file is used to record any deviations from the default timing behaviour for a particular model. Timing behaviour is defined in terms of atomic instructions listed below, to which times are attached. The time may be made dependent on a basic type, to allow modeling of type dependent timing behaviour.

module *TIMEPARSER*

imports

1665.0 from *REP*

1666.0 types *BasicTypeRep*

exports all

definitions

A time file consists of a sequence of time entries.

types

1667.0 $Timefile = TimeEntry^*;$

A time entry may be simple, typed or an assembly language entry. The difference between simple and typed is the presence of a basic type with the latter.

1668.0 $TimeEntry = SimpleEntry \mid TypedEntry \mid AssemblyEntry;$

1669.0 $SimpleEntry :: instr : Instruction$

.1 $time : TimeExpr$

.2 $cid : ContextInfo;$

1670.0 $\mathit{TypedEntry} :: \mathit{instr} : \mathit{Instruction}$
.1 $\mathit{time} : \mathit{TimeExpr}$
.2 $\mathit{tp} : \mathit{REP}`\mathit{BasicTypeRep}$
.3 $\mathit{cid} : \mathit{ContextInfo};$

Context information is currently just a line number, though this could be made more sophisticated in the future if necessary.

1671.0 $\mathit{ContextInfo} :: \mathit{line} : \mathbb{N};$

The actual instructions are described in the table below. Note that the interpretation of these instructions should be the overhead of the task described, without regard to the time required to evaluate the arguments to the instruction - this is calculated separately.

Instruction	Description
AddEnv	Add an identifier to
Addset	Add a single elemen
AllorExists	Quantification expr
Always	Always statement (
Appendmap	Append a single ma
Appendseq	Append a single ele
Appendtup	Append a componen
Apply	Function/operation
BindList	Form a list of bindi set comprehension)
BlockStmt	Block statement
Br	Unconditional branc
Call	Operation call
Cbr	Conditional branch
Exit	Exit statement (exc
Fieldsel	Field selection expr
Guard	Evaluation of permi
History	Evaluation of histor
Iota	Iota expression
Isexpr	Is expression
Isofbaseclass	IsofBaseClass expre
Isofclass	IsOfClass expression
LambdaExpr	Lambda expression
LetBeST	Let be such that ex
Lookup	Look up a name in
Loop	Loop statement (se
MapInverse	Map inverse express
MatchPattern	Match pattern in ca
Newobj	Object creation
NonDet	Non deterministic s
Pattern	Evaluation of arbitr
Polyinst	Instantiation of pol
RecTrap	Recursive exception
Reccons	Record constructor
Recmod	Record modification
Return	Return statement
Samebaseclass	SameBaseClass exp
Sameclass	SameClass expressio
Selfexpr	Self expression
Seqlen	Sequence length exp
Seqmapover	seq or map override
Setcard	set cardinality expr
Setrng	set range expression
Start	start statement (thr
Subseq	subsequence express
Threadid	thread id expression
Trap	trap statement (exc
Tupsel	Tuple selection expr
Update	Assignment
Plus	Addition
Minus	Subtraction
Mult	Multiplication
Div	Division
Rem	Remainder
Mod	Modulo
Intdiv	Integer division
And	Logical and
Or	Logical or
Equal	Equality

The instruction types are given below without further comment.

1672.0 $\mathit{Instruction} = \mathit{AddEnv} \mid$
.1 $\mathit{Addset} \mid$
.2 $\mathit{AllorExists} \mid$
.3 $\mathit{Always} \mid$
.4 $\mathit{Appendmap} \mid$
.5 $\mathit{Appendseq} \mid$
.6 $\mathit{Appendtup} \mid$
.7 $\mathit{Apply} \mid$
.8 $\mathit{BinaryOp} \mid$
.9 $\mathit{BindList} \mid$
.10 $\mathit{BlockStmt} \mid$
.11 $\mathit{Br} \mid$
.12 $\mathit{Call} \mid$
.13 $\mathit{Cbr} \mid$
.14 $\mathit{Exit} \mid$
.15 $\mathit{Fieldsel} \mid$
.16 $\mathit{Guard} \mid$
.17 $\mathit{History} \mid$
.18 $\mathit{Iota} \mid$
.19 $\mathit{Isexpr} \mid$
.20 $\mathit{Isofbaseclass} \mid$
.21 $\mathit{Isofclass} \mid$
.22 $\mathit{LambdaExpr} \mid$
.23 $\mathit{LetBeST} \mid$
.24 $\mathit{Lookup} \mid$
.25 $\mathit{Loop} \mid$
.26 $\mathit{MapInverse} \mid$
.27 $\mathit{MatchPattern} \mid$
.28 $\mathit{Newobj} \mid$
.29 $\mathit{NonDet} \mid$
.30 $\mathit{Pattern} \mid$
.31 $\mathit{Polyinst} \mid$
.32 $\mathit{RecTrap} \mid$
.33 $\mathit{Reccons} \mid$
.34 $\mathit{Recmod} \mid$
.35 $\mathit{Return} \mid$
.36 $\mathit{Samebaseclass} \mid$
.37 $\mathit{Sameclass} \mid$
.38 $\mathit{Selfexpr} \mid$
.39 $\mathit{Seqlen} \mid$
.40 $\mathit{Seqmapover} \mid$
.41 $\mathit{Setcard} \mid$
.42 $\mathit{Setrng} \mid$
.43 $\mathit{Start} \mid$
.44 $\mathit{Subseq} \mid$
.45 $\mathit{Threadid} \mid$
.46 $\mathit{Trap} \mid$
.47 $\mathit{Tupsel} \mid$
.48 $\mathit{UnaryOp} \mid$
.49 $\mathit{Update};$

1673.0 $AddEnv ::\ ;$

1674.0 $Addset ::\ ;$

1675.0 $AllorExists ::\ ;$

1676.0 $Always ::\ ;$

1677.0 $Appendmap ::\ ;$

1678.0 $Appendseq ::\ ;$

1679.0 $Appendtup ::\ ;$

1680.0 $Apply ::\ ;$

1681.0 $BindList ::\ ;$

1682.0 $BlockStmt ::\ ;$

1683.0 $Br ::\ ;$

1684.0 $Call ::\ ;$

1685.0 $Cbr ::\ ;$

1686.0 $Exit ::\ ;$

1687.0 $Fieldsel ::\ ;$

1688.0 $Guard ::\ ;$

1689.0 $History ::\ ;$

1690.0 $Iota ::\ ;$

1691.0 $Isexpr ::\ ;$

1692.0 $Isofbaseclass ::\ ;$

1693.0 $Isofclass ::\ ;$

1694.0 $LambdaExpr ::\ ;$

1695.0 $LetBeST ::\ ;$

1696.0 $Lookup ::\ ;$

1697.0 $Loop ::\ ;$

1698.0 $MapInverse ::\ ;$

1699.0 $MatchPattern ::\ ;$

1700.0 $Newobj ::\ ;$

1701.0 $NonDet ::\ ;$

1702.0 $Pattern ::\ ;$

1703.0 $Polyinst ::\ ;$

1704.0 $RecTrap ::\ ;$

1705.0 $Reccons ::\ ;$

1706.0 $Recmod ::\ ;$

1707.0 $Return ::\ ;$

1708.0 $Samebaseclass ::\ ;$

1709.0 $Sameclass ::\ ;$

1710.0 $Selfexpr ::\ ;$

1711.0 $Seqlen ::\ ;$

1712.0 $Seqmapover ::\ ;$

1713.0 $Setcard ::\ ;$

1714.0 $Setrng ::\ ;$

1715.0 $Start ::\ ;$

1716.0 $Subseq ::\ ;$

1717.0 $Threadid ::\ ;$

1718.0 $Trap ::\ ;$

1719.0 $Tupsel :: \;\; ;$

1720.0 $Update :: \;\; ;$

1721.0 $BinaryOp = Plus \mid Minus \mid Mult \mid Div \mid Rem \mid Mod \mid Intdiv \mid And \mid$
.1 $Or \mid Equal \mid GreaterThan \mid GreaterThanOrEqual;$

1722.0 $Plus :: \;\; ;$

1723.0 $Minus :: \;\; ;$

1724.0 $Mult :: \;\; ;$

1725.0 $Div :: \;\; ;$

1726.0 $Rem :: \;\; ;$

1727.0 $Mod :: \;\; ;$

1728.0 $Intdiv :: \;\; ;$

1729.0 $And :: \;\; ;$

1730.0 $Or :: \;\; ;$

1731.0 $Equal :: \;\; ;$

1732.0 $GreaterThan :: \;\; ;$

1733.0 $GreaterThanOrEqual :: \;\; ;$

1734.0 $UnaryOp = Abs \mid Floor \mid Not;$

1735.0 $Abs :: \;\; ;$

1736.0 $Floor :: \;\; ;$

1737.0 $Not :: \;\; ;$

A time expression may be an assembly instruction, a natural number, a binary expression or a bracketed expression.

1738.0 $TimeExpr = AssemblyInstruction \mid \mathbb{N} \mid TimeBinaryExpr \mid TimeBracketedExpr;$

A binary expression may be an addition or a multiplication.

1739.0 $\mathit{TimeBinaryExpr} :: \mathit{lexpr} : \mathit{TimeExpr}$
.1 $\mathit{op} : \mathit{TimeBinaryOp}$
.2 $\mathit{rexpr} : \mathit{TimeExpr};$

1740.0 $\mathit{TimeBinaryOp} = \mathit{TimePlus} \mid \mathit{TimeMultiply};$

1741.0 $\mathit{TimePlus} :: \ ;$

1742.0 $\mathit{TimeMultiply} :: \ ;$

A bracketed expression consists of just the expression enclosed in the brackets.

1743.0 $\mathit{TimeBracketedExpr} :: \mathit{expr} : \mathit{TimeExpr};$

The choice of assembly instructions corresponds to those typically found on a processor.

1744.0 $\mathit{AssemblyInstruction} = \mathit{Assembly\text{-}Add} \mid \mathit{Assembly\text{-}Branch} \mid \mathit{Assembly\text{-}Call} \mid$
.1 $\mathit{Assembly\text{-}Compare} \mid \mathit{Assembly\text{-}Return} \mid \mathit{Assembly\text{-}Sub} \mid$
.2 $\mathit{Assembly\text{-}Div} \mid \mathit{Assembly\text{-}Mul} \mid \mathit{Assembly\text{-}Neg} \mid$
.3 $\mathit{Assembly\text{-}Fsqrt} \mid \mathit{Assembly\text{-}Logic} \mid \mathit{Assembly\text{-}Cas} \mid$
.4 $\mathit{Assembly\text{-}Push} \mid \mathit{Assembly\text{-}Pop};$

1745.0 $\mathit{AssemblyEntry} :: \mathit{instr} : \mathit{AssemblyInstruction}$
.1 $\mathit{time} : \mathbb{N}$
.2 $\mathit{cid} : \mathit{ContextInfo};$

1746.0 $\mathit{Assembly\text{-}Add} :: \ ;$

1747.0 $\mathit{Assembly\text{-}Branch} :: \ ;$

1748.0 $\mathit{Assembly\text{-}Call} :: \ ;$

1749.0 $\mathit{Assembly\text{-}Compare} :: \ ;$

1750.0 $\mathit{Assembly\text{-}Return} :: \ ;$

1751.0 $\mathit{Assembly\text{-}Sub} :: \ ;$

1752.0 $\mathit{Assembly\text{-}Div} :: \ ;$

1753.0 $\mathit{Assembly\text{-}Mul} :: \ ;$

1754.0 $\mathit{Assembly\text{-}Neg} :: \ ;$

1755.0 $\mathit{Assembly\text{-}Fsqrt} :: \ ;$

1756.0 $Assembly\text{-}Logic ::\ \ ;$

1757.0 $Assembly\text{-}Cas ::\ \ ;$

1758.0 $Assembly\text{-}Push ::\ \ ;$

1759.0 $Assembly\text{-}Pop ::$

The function *InstrMap* is used to map instructions to a string representation.
functions

1760.0 $\mathit{InstrMap} : \mathit{Instruction} \to \mathsf{char}^*$
.1 $\mathit{InstrMap}\,(i) \triangleq$
.2 cases i :
.3 mk-$\mathit{AddEnv}\,() \to$ "AddEnv",
.4 mk-$\mathit{Addset}\,() \to$ "Addset",
.5 mk-$\mathit{AllorExists}\,() \to$ "$\mathit{AllorExists}$",
.6 mk-$\mathit{Always}\,() \to$ "Always",
.7 mk-$\mathit{Appendmap}\,() \to$ "$\mathit{Appendmap}$",
.8 mk-$\mathit{Appendseq}\,() \to$ "$\mathit{Appendseq}$",
.9 mk-$\mathit{Appendtup}\,() \to$ "$\mathit{Appendtup}$",
.10 mk-$\mathit{Apply}\,() \to$ "Apply",
.11 mk-$\mathit{BindList}\,() \to$ "$\mathit{BindList}$",
.12 mk-$\mathit{BlockStmt}\,() \to$ "$\mathit{BlockStmt}$",
.13 mk-$\mathit{Br}\,() \to$ "Br",
.14 mk-$\mathit{Call}\,() \to$ "Call",
.15 mk-$\mathit{Cbr}\,() \to$ "Cbr",
.16 mk-$\mathit{Exit}\,() \to$ "Exit",
.17 mk-$\mathit{Fieldsel}\,() \to$ "$\mathit{Fieldsel}$",
.18 mk-$\mathit{Guard}\,() \to$ "Guard",
.19 mk-$\mathit{History}\,() \to$ "$\mathit{History}$",
.20 mk-$\mathit{Iota}\,() \to$ "Iota",
.21 mk-$\mathit{Isexpr}\,() \to$ "Isexpr",
.22 mk-$\mathit{Isofbaseclass}\,() \to$ "$\mathit{Isofbaseclass}$",
.23 mk-$\mathit{Isofclass}\,() \to$ "$\mathit{Isofclass}$",
.24 mk-$\mathit{LambdaExpr}\,() \to$ "$\mathit{LambdaExpr}$",
.25 mk-$\mathit{LetBeST}\,() \to$ "$\mathit{LetBeST}$",
.26 mk-$\mathit{Lookup}\,() \to$ "Lookup",
.27 mk-$\mathit{Loop}\,() \to$ "Loop",
.28 mk-$\mathit{MapInverse}\,() \to$ "$\mathit{MapInverse}$",
.29 mk-$\mathit{MatchPattern}\,() \to$ "$\mathit{MatchPattern}$",
.30 mk-$\mathit{Newobj}\,() \to$ "Newobj",
.31 mk-$\mathit{NonDet}\,() \to$ "NonDet",
.32 mk-$\mathit{Pattern}\,() \to$ "$\mathit{Pattern}$",
.33 mk-$\mathit{Polyinst}\,() \to$ "$\mathit{Polyinst}$",
.34 mk-$\mathit{RecTrap}\,() \to$ "$\mathit{RecTrap}$",
.35 mk-$\mathit{Reccons}\,() \to$ "$\mathit{Reccons}$",
.36 mk-$\mathit{Recmod}\,() \to$ "Recmod",
.37 mk-$\mathit{Return}\,() \to$ "Return",
.38 mk-$\mathit{Samebaseclass}\,() \to$ "$\mathit{Samebaseclass}$",
.39 mk-$\mathit{Sameclass}\,() \to$ "$\mathit{Sameclass}$",
.40 mk-$\mathit{Selfexpr}\,() \to$ "$\mathit{Selfexpr}$",
.41 mk-$\mathit{Seqlen}\,() \to$ "Seqlen",
.42 mk-$\mathit{Seqmapover}\,() \to$ "$\mathit{Seqmapover}$",
.43 mk-$\mathit{Setcard}\,() \to$ "$\mathit{Setcard}$",
.44 mk-$\mathit{Setrng}\,() \to$ "Setrng",
.45 mk-$\mathit{Start}\,() \to$ "Start",
.46 mk-$\mathit{Subseq}\,() \to$ "Subseq",
.47 mk-$\mathit{Threadid}\,() \to$ "$\mathit{Threadid}$",
.48 mk-$\mathit{Trap}\,() \to$ "Trap",
.49 mk-$\mathit{Tupsel}\,() \to$ "Tupsel",
.50 mk-$\mathit{Update}\,() \to$ "Update",
.51 mk-$\mathit{Plus}\,() \to$ "Plus",
.52 mk-$\mathit{Minus}\,() \to$ "Minus",
.53 mk-$\mathit{Mult}\,() \to$ "Mult",
.54 mk-$\mathit{Div}\,() \to$ "Div",
.55 mk-$\mathit{Rem}\,() \to$ "Rem",
.56 mk-$\mathit{Mod}\,() \to$ "Mod",
.57 mk-$\mathit{Intdiv}\,() \to$ "Intdiv",
.58 mk-$\mathit{And}\,() \to$ "And",
.59 mk-$\mathit{Or}\,() \to$ "Or",

1761.0 $AssemblyInstrMap : AssemblyInstruction \rightarrow \mathsf{char}^*$
.1 $AssemblyInstrMap\,(ai) \triangleq$
.2 cases ai :
.3 mk-$Assembly\text{-}Add\,() \rightarrow$ "add",
.4 mk-$Assembly\text{-}Branch\,() \rightarrow$ "$branch$",
.5 mk-$Assembly\text{-}Call\,() \rightarrow$ "$call$",
.6 mk-$Assembly\text{-}Compare\,() \rightarrow$ "$compare$",
.7 mk-$Assembly\text{-}Return\,() \rightarrow$ "$return$",
.8 mk-$Assembly\text{-}Sub\,() \rightarrow$ "sub",
.9 mk-$Assembly\text{-}Div\,() \rightarrow$ "div",
.10 mk-$Assembly\text{-}Mul\,() \rightarrow$ "mul",
.11 mk-$Assembly\text{-}Neg\,() \rightarrow$ "neg",
.12 mk-$Assembly\text{-}Fsqrt\,() \rightarrow$ "$fsqrt$",
.13 mk-$Assembly\text{-}Logic\,() \rightarrow$ "$logic$",
.14 mk-$Assembly\text{-}Cas\,() \rightarrow$ "cas",
.15 mk-$Assembly\text{-}Push\,() \rightarrow$ "$push$",
.16 mk-$Assembly\text{-}Pop\,() \rightarrow$ "pop"
.17 end

end $TIMEPARSER$

Test Suite : rtinfo.ast
Module : TIMEPARSER

Name	**#Calls**	**Coverage**
TIMEPARSER‘InstrMap	**undefined**	**undefined**
TIMEPARSER‘AssemblyInstrMap	**undefined**	**undefined**
Total Coverage		**0%**

1.27 Introduction

This module provides support for the time map.
module $TIMEMAP$

imports

1762.0 from REP

1763.0 types $BasicTypeRep$,

1764.0 from $TIMEPARSER$ all

exports all

definitions

We arbitrarily set the default duration of a time parser instruction to be 2 time units. Note that there is no particular motivation for this choice, and any natural would suffice (including 0).

values

1765.0 $\mathit{defaultDuration} : \mathbb{N} = 2$

A time map is a map from (instruction,type) pairs to a natural number. The type is a parameter so that type specific timing behaviour (e.g. integer addition vs floating point addition) can be modeled. If it is not meaningful to assign such a type, then the value nil is used.

types

1766.0 $\mathit{Timemap} = \mathit{TIMEPARSER`Instruction} \times [\mathit{REP`BasicTypeRep}] \xrightarrow{m} \mathbb{N};$

The type *Error* represents a well-formedness error in a user-defined time file.

1767.0 $\mathit{Error} :: \mathit{entry} : \mathit{TIMEPARSER`TimeEntry}$
.1 $\quad \mathit{reps} : \mathit{TIMEPARSER`TimeEntry}\text{-set}$

Well-formedness of time files is checked using the function $wf_T imefile$. A time file is well formed if there exists no two entries in the time file with identical instructions and types, but different times.

functions

1768.0 $\mathit{wf\text{-}Timefile} : \mathit{TIMEPARSER`Timefile} \to \mathit{Error}\text{-set}$
.1 $\mathit{wf\text{-}Timefile}\,(\mathit{tf}) \triangleq$
.2 $\quad \text{let } \mathit{pos\text{-}errors} =$
.3 $\qquad \{\text{mk-}\mathit{Error}\,(\mathit{tf}\,(i),$
.4 $\qquad\qquad \{f \mid f \in \text{elems } \mathit{tf}\,(i+1,\ldots,\text{len } \mathit{tf}) \cdot$
.5 $\qquad\qquad\qquad f = \mu\,(\mathit{tf}\,(i), \mathit{cid} \mapsto f.\mathit{cid}, \mathit{time} \mapsto f.\mathit{time})\}) \mid$
.6 $\qquad\quad i \in \text{inds } \mathit{tf}\} \text{ in}$
.7 $\quad \{e \mid e \in \mathit{pos\text{-}errors} \cdot e.\mathit{reps} \neq \{\}\};$

The function *MkTimeMap* takes a well formed timefile and merges it with the default time map to generate a global time map that will be used by the interpreter in its timing calculations.

1769.0 $\mathit{MkTimeMap} : \mathit{TIMEPARSER`Timefile} \to \mathit{Timemap}$
.1 $\mathit{MkTimeMap}\,(\mathit{tf}) \triangleq$
.2 $\quad \mathit{defaultTimemap}\,() \dagger$
.3 $\quad \{(\text{if is-}\mathit{TIMEPARSER`SimpleEntry}\,(\mathit{entry})$
.4 $\qquad \text{then mk-}\,(\mathit{entry.instr}, \text{nil})$
.5 $\qquad \text{else mk-}\,(\mathit{entry.instr}, \mathit{entry.tp})) \mapsto$
.6 $\qquad \mathit{entry.time} \mid$
.7 $\qquad\quad \mathit{entry} \in \text{elems } \mathit{tf}\}$
.8 $\text{pre } \mathit{wf\text{-}Timefile}\,(\mathit{tf}) = \{\}\ ;$

The default time map simply maps each instruction with nil type to the default duration.

1770.0 $\mathit{defaultTimemap} : () \to \mathit{Timemap}$

.1 $\mathit{defaultTimemap}\,() \triangleq$

.2 $\{$

.3 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{AddEnv}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.4 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Addset}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.5 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{AllorExists}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.6 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Always}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.7 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Appendmap}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.8 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Appendseq}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.9 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Appendtup}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.10 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Apply}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.11 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{BindList}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.12 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{BlockStmt}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.13 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Br}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.14 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Call}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.15 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Cbr}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.16 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Exit}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.17 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Fieldsel}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.18 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Guard}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.19 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{History}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.20 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Iota}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.21 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Isexpr}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.22 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Isofbaseclass}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.23 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Isofclass}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.24 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{LambdaExpr}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.25 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{LetBeST}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.26 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Lookup}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.27 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Loop}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.28 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{MapInverse}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.29 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{MatchPattern}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.30 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Newobj}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.31 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{NonDet}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.32 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Pattern}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.33 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Polyinst}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.34 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{RecTrap}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.35 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Reccons}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.36 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Recmod}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.37 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Return}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.38 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Samebaseclass}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.39 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Sameclass}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.40 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Selfexpr}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.41 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Seqlen}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.42 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Seqmapover}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.43 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Setcard}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.44 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Setrng}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.45 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Start}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.46 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Subseq}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.47 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Threadid}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.48 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Trap}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.49 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Tupsel}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.50 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Update}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.51 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Plus}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.52 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Minus}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.53 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Mult}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.54 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Div}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.55 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Rem}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.56 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Mod}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.57 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Intdiv}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.58 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{And}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

.59 $\mathsf{mk\text{-}}(\mathsf{mk\text{-}}\mathit{TIMEPARSER}\text{‘}\mathit{Or}\,(), \mathsf{nil}) \mapsto \mathit{defaultDuration},$

end *TIMEMAP*

Test Suite : rtinfo.ast
Module : TIMEMAP

Name	**#Calls**	**Coverage**
TIMEMAP‘MkTimeMap	**undefined**	**undefined**
TIMEMAP‘wf-Timefile	**undefined**	**undefined**
TIMEMAP‘defaultTimemap	**undefined**	**undefined**
Total Coverage		**0%**

1.28 Introduction

This module specifies how to increment the internal clock in the stack machine (STKM). The approach is based on estimating how long each construct in AS‘Expr and AS‘Statement would take to execute on some mythical target processor. The time for each such construct is subdivided into the time taken to execute the components of the construct, perhaps together with some overhead to execute the whole. The overhead may be dependent on the size of the construct and/or arguments, or some constant for the target processor, or both. The constants for the target processor are read from the time table specified in TIMEMAP.

For example, the time taken to execute an assignment statement consists of the time taken to execute the right hand side expression, the time taken to update a memory location and the time taken to update the environment. The latter two will be constants for a given model, so are looked up in the time table.
module *TIME*

imports

1771.0 from AS all ,

1772.0 from CI all ,

1773.0 from IO all ,

1774.0 from PAT all ,

1775.0 from REP all ,

1776.0 from SEM all ,

1777.0 from $CMPL$ all ,

1778.0 from $STKM$ all ,

1779.0 from $RTERR$ all ,

1780.0 from $GLOBAL$ all ,

1781.0 from $SCHDTP$ all ,

1782.0 from $INSTRTP$ all ,

1783.0 from $TIMEMAP$ all ,

1784.0 from $TIMEPARSER$ all

exports

1785.0 functions $MkBr : () \rightarrow STKM`SubProgram;$
.1 $MkCbr : () \rightarrow STKM`SubProgram;$
.2 $E2Time : AS`Expr \rightarrow STKM`SubProgram;$
.3 $S2Time : AS`Stmt \rightarrow STKM`SubProgram;$
.4 $MkLoopBind : \mathbb{N} \rightarrow STKM`SubProgram;$
.5 $MkMatchPattern : () \rightarrow STKM`SubProgram;$
.6 $MkMapCompInsert : () \rightarrow STKM`SubProgram;$
.7 $MkSeqCompInsert : () \rightarrow STKM`SubProgram;$
.8 $MkSetCompInsert : () \rightarrow STKM`SubProgram;$
.9 $IsRuntimeBinaryOp : AS`BinaryOp \rightarrow \mathbb{B};$
.10 $IsRuntimePrefixOp : AS`UnaryOp \rightarrow \mathbb{B};$
.11 $MkRuntimeBinaryOp : AS`BinaryOp \rightarrow STKM`SubProgram;$
.12 $MkRuntimePrefixOp : AS`UnaryOp \rightarrow STKM`SubProgram;$
.13 $MkRuntimeSetSeqMap : AS`SetRangeExpr \mid AS`SubSequenceExpr \mid$
.14 $AS`SeqModifyMapOverrideExpr \rightarrow$
.15 $STKM`SubProgram;$
.16 $MkRuntimeStartList : () \rightarrow STKM`SubProgram$

1786.0 operations $GetCompilingTime : () \xrightarrow{o} \mathbb{B};$
.1 $SetCompilingTime : \mathbb{B} \xrightarrow{o} ()$

definitions

1787.0 state TS of
.1 $compilingTime : \mathbb{B}$
.2 init $ts \triangleq ts =$ mk-TS (true)
.3 end

operations

1788.0 $SetCompilingTime : \mathbb{B} \xrightarrow{o} ()$

.1 $SetCompilingTime\,(newCompilingTime) \triangleq$

.2 $\quad compilingTime := newCompilingTime;$

1789.0 $GetCompilingTime : () \xrightarrow{o} \mathbb{B}$

.1 $GetCompilingTime\,() \triangleq$

.2 $\quad \mathsf{return}\ compilingTime$

1.29 Expressions

We subdivide expressions according to the kind of expression given. Notice that for some expressions, the time taken to evaluate the expression depends on runtime values e.g. for a set range expression, the time taken to compute the actual set depends on the values determined at run time of the lower and upper bounds. The following expressions fall into this category:

- set distributed union
- set distributed intersect
- set power
- sequence distributed concatenation
- the elements of a sequence
- the indiciates of a sequence
- the tail of a sequence
- the domain and range of a map
- map distributed merge
- raising a number to a power
- set union and intersection
- set difference
- subset and proper subset
- set membership (and its negation)
- sequence concatenation
- map merge
- dom and range restriction for maps
- composition expressions

- set range expressions
- subsequence expressions
- sequence/map override expressions
- new expressions

In addition, if expressions and cases expressions are dealt with in CEXPR‘CompileIfExpr and CEXPR‘CompileCasesExpr respectively.
functions

```
1790.0 E2Time : AS`Expr → STKM`SubProgram
E2Time (e) △
  cases true :
    (is-AS`BracketedExpr (e)) → E2Time (e.expr),
    (is-AS`DefExpr (e)) → TimeDefExpr (e),
    (is-AS`LetExpr (e)) → TimeLetExpr (e),
    (is-AS`LetBeSTExpr (e)) → TimeLetBeSTExpr (),
    (is-AS`AllOrExistsExpr (e)) → TimeAllOrExistsExpr (e),
    (is-AS`ExistsUniqueExpr (e)) → TimeExistsUniqueExpr (),
    (is-AS`IotaExpr (e)) → TimeIotaExpr (),
    (is-AS`ApplyExpr (e)) → TimeApplyExpr (),
    (is-AS`FieldSelectExpr (e)) → TimeFieldSelectExpr (),
    (is-AS`MapInverseExpr (e)) → TimeMapInverseExpr (),
    (is-AS`PrefixExpr (e)) → TimePrefixExpr (e),
    (is-AS`BinaryExpr (e)) → TimeBinaryExpr (e),
    (is-AS`SetEnumerationExpr (e)) → TimeSetEnumExpr (e),
    (is-AS`SeqEnumerationExpr (e)) → TimeSeqEnumExpr (e),
    (is-AS`MapEnumerationExpr (e)) → TimeMapEnumExpr (e),
    (is-AS`SetComprehensionExpr (e)) → TimeSetComprehensionExpr (e),
    (is-AS`SeqComprehensionExpr (e)) → TimeSeqComprehensionExpr (),
    (is-AS`MapComprehensionExpr (e)) → TimeMapComprehensionExpr (e),
    (is-AS`TupleConstructorExpr (e)) → TimeTupleConstructorExpr (e),
    (is-AS`RecordConstructorExpr (e)) → TimeRecordConstructorExpr (e),
    (is-AS`RecordModifierExpr (e)) → TimeRecordModifierExpr (e),
    (is-AS`LambdaExpr (e)) → TimeLambdaExpr (e),
    (is-AS`FctTypeInstExpr (e)) → TimeFctTypeInstExpr (e),
    (is-AS`IsExpr (e)) → TimeIsExpr (e),
    (is-AS`TupleSelectExpr (e)) → TimeTupleSelectExpr (),
    (is-AS`TypeJudgementExpr (e)) → TimeTypeJudgementExpr (e),
    (is-AS`Name (e)) → TimeNameLookUp (),
    (is-AS`OldName (e)) → TimeNameLookUp (),
    (is-AS`SelfExpr (e)) → TimeSelfExpr (),
    (is-AS`NewExpr (e)) → TimeNewExpr (e),
    (is-AS`IsOfClassExpr (e)) → TimeIsOfClassExpr (),
    (is-AS`IsOfBaseClassExpr (e)) → TimeIsOfBaseClassExpr (),
    (is-AS`SameBaseClassExpr (e)) → TimeSameBaseClassExpr (),
    (is-AS`SameClassExpr (e)) → TimeSameClassExpr (),
    (is-AS`ThreadIdExpr (e)) → TimeThreadIdExpr (),
    (is-AS`ActExpr (e)) → TimeHistoryExpr (),
    (is-AS`FinExpr (e)) → TimeHistoryExpr (),
    (is-AS`ActiveExpr (e)) → TimeHistoryExpr (),
    (is-AS`WaitingExpr (e)) → TimeHistoryExpr (),
    (is-AS`ReqExpr (e)) → TimeHistoryExpr (),
    others → []
  end;
```

The time taken to evaluate a def expression is given by (number of bindings × overhead for each binding) + time to eval rhs of each bind + time to eval body. Thus in *TimeDefExpr* we just consider the overhead component. It is not meaningful to parametrize on type, so the nil entry in the time table is looked up.

1791.0 $TimeDefExpr : AS`DefExpr \rightarrow STKM`SubProgram$
.1 $TimeDefExpr\,(\textsf{mk-}AS`DefExpr\,(def\text{-}l, \text{-}, \text{-})) \triangleq$
.2 $\quad \textsf{let } oh = TimeLookup\,(\textsf{mk-}TIMEPARSER`AddEnv\,(), \textsf{nil}\,) \textsf{ in}$
.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\,(oh \times \textsf{len } def\text{-}l)];$

Let expressions are similar to def expressions. It is assumed that the definition on the RHS can be compiled statically, so the time taken to bind it is just the fixed overhead for binding.

1792.0 $TimeLetExpr : AS`LetExpr \rightarrow STKM`SubProgram$
.1 $TimeLetExpr\,(\textsf{mk-}AS`LetExpr\,(def\text{-}l, \text{-}, \text{-})) \triangleq$
.2 $\quad \textsf{let } oh = TimeLookup\,(\textsf{mk-}TIMEPARSER`AddEnv\,(), \textsf{nil}\,) \textsf{ in}$
.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\,(oh \times \textsf{len } def\text{-}l)];$

For let be st expressions we assume a different overhead on top of the binding overhead. This overhead relates to selection of a matching expression.

1793.0 $TimeLetBeSTExpr : () \rightarrow STKM`SubProgram$
.1 $TimeLetBeSTExpr\,() \triangleq$
.2 $\quad \textsf{let } oh = TimeLookup\,(\textsf{mk-}TIMEPARSER`LetBeST\,(), \textsf{nil}\,) \textsf{ in}$
.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\,(oh)];$

For quantification expressions, the time taken depends on the overhead of a quantification expression and the number of bindings to be performed.

1794.0 $TimeAllOrExistsExpr : AS`AllOrExistsExpr \rightarrow STKM`SubProgram$
.1 $TimeAllOrExistsExpr\,(\textsf{mk-}AS`AllOrExistsExpr\,(\text{-}, bind, \text{-}, \text{-})) \triangleq$
.2 $\quad \textsf{let } numbinds = \textsf{len conc } [bind\,(i).pat \mid i \in \textsf{inds } bind],$
.3 $\qquad oh1 = TimeLookup\,(\textsf{mk-}TIMEPARSER`BindList\,(), \textsf{nil}\,),$
.4 $\qquad oh2 = TimeLookup\,(\textsf{mk-}TIMEPARSER`AllorExists\,(), \textsf{nil}\,) \textsf{ in}$
.5 $\quad [\textsf{mk-}INSTRTP`INCRTIME\,((oh1 \times numbinds) + oh2)];$

An exists unique expression is somewhat different because the time taken to check uniqueness is accounted for elsewhere.

1795.0 $TimeExistsUniqueExpr : () \rightarrow STKM`SubProgram$
.1 $TimeExistsUniqueExpr\,() \triangleq$
.2 $\quad \textsf{let } oh1 = TimeLookup\,(\textsf{mk-}TIMEPARSER`AddEnv\,(), \textsf{nil}\,),$
.3 $\qquad oh2 = TimeLookup\,(\textsf{mk-}TIMEPARSER`AllorExists\,(), \textsf{nil}\,) \textsf{ in}$
.4 $\quad [\textsf{mk-}INSTRTP`INCRTIME\,(oh1 + oh2)];$

The time taken to execute an iota expression consists of the constant time for an iota expression, and the time to add a new identifier to the environment.

1796.0 $\mathit{TimeIotaExpr} : () \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeIotaExpr}\,() \triangleq$
.2 $\quad$ let $oh1 = \mathit{TimeLookup}\,(\text{mk-}\mathit{TIMEPARSER}`\mathit{AddEnv}\,(), \text{nil}\,),$
.3 $\qquad oh2 = \mathit{TimeLookup}\,(\text{mk-}\mathit{TIMEPARSER}`\mathit{Iota}\,(), \text{nil}\,)$ in
.4 $\quad [\text{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(oh1 + oh2)];$

The following expressions (apply expression, field select expression and map inverse expression) each take a fixed overhead given by the constant for the current model.

1797.0 $\mathit{TimeApplyExpr} : () \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeApplyExpr}\,() \triangleq$
.2 $\quad$ let $oh = \mathit{TimeLookup}\,(\text{mk-}\mathit{TIMEPARSER}`\mathit{Apply}\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(oh)];$

1798.0 $\mathit{TimeFieldSelectExpr} : () \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeFieldSelectExpr}\,() \triangleq$
.2 $\quad$ let $oh = \mathit{TimeLookup}\,(\text{mk-}\mathit{TIMEPARSER}`\mathit{Fieldsel}\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(oh)];$

1799.0 $\mathit{TimeMapInverseExpr} : () \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeMapInverseExpr}\,() \triangleq$
.2 $\quad$ let $oh = \mathit{TimeLookup}\,(\text{mk-}\mathit{TIMEPARSER}`\mathit{MapInverse}\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(oh)];$

The time taken for prefix operators is either a constant for a given model, or depends on runtime values (and is thus dealt with elsewhere).

```
1800.0 TimePrefixExpr : AS‘PrefixExpr → STKM‘SubProgram
    .1 TimePrefixExpr (e) ≜
    .2   let time =
    .3          cases e.opr :
    .4            NUMPLUS → 0,
    .5            NUMMINUS → TimeLookup (mk-TIMEPARSER‘Minus (), GetType (e)),
    .6            NUMABS → TimeLookup (mk-TIMEPARSER‘Abs (), GetType (e)),
    .7            FLOOR → TimeLookup (mk-TIMEPARSER‘Floor (), GetType (e)),
    .8            NOT → TimeLookup (mk-TIMEPARSER‘Not (), nil ),
    .9            SETCARD → TimeLookup (mk-TIMEPARSER‘Setcard (), nil ),
   .10            SEQLEN → TimeLookup (mk-TIMEPARSER‘Seqlen (), nil ),
   .11            SEQHEAD → 0,
   .12            others → 0
   .13          end in
   .14   if time ≠ 0
   .15   then [mk-INSTRTP‘INCRTIME (time)]
   .16   else [];
```

Similarly, the time taken for binary operators is either a constant for a given model, or depends on runtime values (and is thus dealt with elsewhere).

```
1801.0 TimeBinaryExpr : AS‘BinaryExpr → STKM‘SubProgram
    .1 TimeBinaryExpr (e) ≜
    .2   let time =
    .3          cases e.opr :
    .4            NUMPLUS → TimeLookup (mk-TIMEPARSER‘Plus (), GetType (e)),
    .5            NUMMINUS → TimeLookup (mk-TIMEPARSER‘Minus (), GetType (e)),
    .6            NUMMULT → TimeLookup (mk-TIMEPARSER‘Mult (), GetType (e)),
    .7            NUMDIV → TimeLookup (mk-TIMEPARSER‘Div (), GetType (e)),
    .8            NUMREM → TimeLookup (mk-TIMEPARSER‘Rem (), GetType (e)),
    .9            NUMMOD → TimeLookup (mk-TIMEPARSER‘Mod (), GetType (e)),
   .10            INTDIV → TimeLookup (mk-TIMEPARSER‘Intdiv (), GetType (e)),
   .11            NUMLT → TimeLookup (mk-TIMEPARSER‘GreaterThan (), GetType (e)),
   .12            NUMLE → TimeLookup (mk-TIMEPARSER‘GreaterThanOrEqual (), GetType (e)),
   .13            NUMGT → TimeLookup (mk-TIMEPARSER‘GreaterThan (), GetType (e)),
   .14            NUMGE → TimeLookup (mk-TIMEPARSER‘GreaterThanOrEqual (), GetType (e)),
   .15            AND → TimeLookup (mk-TIMEPARSER‘And (), nil ),
   .16            OR → TimeLookup (mk-TIMEPARSER‘Or (), nil ),
   .17            IMPLY → TimeLookup (mk-TIMEPARSER‘Or (), nil ) +
   .18                    TimeLookup (mk-TIMEPARSER‘Not (), nil ),
   .19            EQUIV → TimeLookup (mk-TIMEPARSER‘Equal (), GetType (e)),
   .20            EQ → TimeLookup (mk-TIMEPARSER‘Equal (), GetType (e)),
   .21            NE → TimeLookup (mk-TIMEPARSER‘Equal (), GetType (e)),
   .22            others → 0
   .23          end in
   .24   if time ≠ 0
   .25   then [mk-INSTRTP‘INCRTIME (time)]
   .26   else [];
```

For set, sequence and map enumerations, the time taken depends on a constance multiplied by the number of elements in the enumeration.

1802.0 $\mathit{TimeSetEnumExpr} : \mathit{AS}`\mathit{SetEnumerationExpr} \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeSetEnumExpr}\ (\textsf{mk-}\mathit{AS}`\mathit{SetEnumerationExpr}\ (e, \text{-})) \triangleq$
.2 $\quad \textsf{let}\ \mathit{oh} = \mathit{TimeLookup}\ (\textsf{mk-}\mathit{TIMEPARSER}`\mathit{Addset}\ (), \textsf{nil}\ )\ \textsf{in}$
.3 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\ (\mathit{oh} \times \textsf{len}\ e)];$

1803.0 $\mathit{TimeSeqEnumExpr} : \mathit{AS}`\mathit{SeqEnumerationExpr} \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeSeqEnumExpr}\ (\textsf{mk-}\mathit{AS}`\mathit{SeqEnumerationExpr}\ (e, \text{-})) \triangleq$
.2 $\quad \textsf{let}\ \mathit{oh} = \mathit{TimeLookup}\ (\textsf{mk-}\mathit{TIMEPARSER}`\mathit{Appendseq}\ (), \textsf{nil}\ )\ \textsf{in}$
.3 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\ (\mathit{oh} \times \textsf{len}\ e)];$

1804.0 $\mathit{TimeMapEnumExpr} : \mathit{AS}`\mathit{MapEnumerationExpr} \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeMapEnumExpr}\ (\textsf{mk-}\mathit{AS}`\mathit{MapEnumerationExpr}\ (e, \text{-})) \triangleq$
.2 $\quad \textsf{let}\ \mathit{oh} = \mathit{TimeLookup}\ (\textsf{mk-}\mathit{TIMEPARSER}`\mathit{Appendmap}\ (), \textsf{nil}\ )\ \textsf{in}$
.3 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\ (\mathit{oh} \times \textsf{len}\ e)];$

For comprehension expressions, the time taken is handled both here and also in the corresponding function in CEXPR.

1805.0 $\mathit{TimeSetComprehensionExpr} : \mathit{AS}`\mathit{SetComprehensionExpr} \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeSetComprehensionExpr}\ (\textsf{mk-}\mathit{AS}`\mathit{SetComprehensionExpr}\ (\text{-}, \mathit{bind}, \text{-}, \text{-})) \triangleq$
.2 $\quad \textsf{let}\ \mathit{numbinds} = \textsf{len conc}\ [\mathit{bind}\ (i).\mathit{pat} \mid i \in \textsf{inds}\ \mathit{bind}],$
.3 $\qquad \mathit{oh} = \mathit{TimeLookup}\ (\textsf{mk-}\mathit{TIMEPARSER}`\mathit{BindList}\ (), \textsf{nil}\ )\ \textsf{in}$
.4 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\ (\mathit{oh} \times \mathit{numbinds})];$

1806.0 $\mathit{TimeSeqComprehensionExpr} : () \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeSeqComprehensionExpr}\ () \triangleq$
.2 $\quad \textsf{let}\ \mathit{oh} = \mathit{TimeLookup}\ (\textsf{mk-}\mathit{TIMEPARSER}`\mathit{BindList}\ (), \textsf{nil}\ )\ \textsf{in}$
.3 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\ (\mathit{oh})];$

1807.0 $\mathit{TimeMapComprehensionExpr} : \mathit{AS}`\mathit{MapComprehensionExpr} \rightarrow \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{TimeMapComprehensionExpr}\ (\textsf{mk-}\mathit{AS}`\mathit{MapComprehensionExpr}\ (\text{-}, \mathit{bind}, \text{-}, \text{-})) \triangleq$
.2 $\quad \textsf{let}\ \mathit{numbinds} = \textsf{len conc}\ [\mathit{bind}\ (i).\mathit{pat} \mid i \in \textsf{inds}\ \mathit{bind}],$
.3 $\qquad \mathit{oh} = \mathit{TimeLookup}\ (\textsf{mk-}\mathit{TIMEPARSER}`\mathit{BindList}\ (), \textsf{nil}\ )\ \textsf{in}$
.4 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\ (\mathit{oh} \times \mathit{numbinds})];$

The time to construct records or tuples depends on the number of fields in the record or tuple.

1808.0 $\mathit{TimeTupleConstructorExpr} : \mathit{AS}`\mathit{TupleConstructorExpr} \to \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeTupleConstructorExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{TupleConstructorExpr}\,(\mathit{fields}, \text{-})) \triangleq$

.2 $\quad \textsf{let } \mathit{oh} = \mathit{TimeLookup}\,(\textsf{mk-}\mathit{TIMEPARSER}`\mathit{Appendtup}\,(), \textsf{nil}\,) \textsf{ in}$

.3 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(\mathit{oh} \times \textsf{len } \mathit{fields})];$

1809.0 $\mathit{TimeRecordConstructorExpr} : \mathit{AS}`\mathit{RecordConstructorExpr} \to \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeRecordConstructorExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{RecordConstructorExpr}\,(\text{-}, \mathit{fields}, \text{-})) \triangleq$

.2 $\quad \textsf{let } \mathit{oh} = \mathit{TimeLookup}\,(\textsf{mk-}\mathit{TIMEPARSER}`\mathit{Reccons}\,(), \textsf{nil}\,) \textsf{ in}$

.3 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(\mathit{oh} \times \textsf{len } \mathit{fields})];$

Modifying a record depends on the number of fields in the record that are to be modified.

1810.0 $\mathit{TimeRecordModifierExpr} : \mathit{AS}`\mathit{RecordModifierExpr} \to \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeRecordModifierExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{RecordModifierExpr}\,(\text{-}, \mathit{mods}, \text{-})) \triangleq$

.2 $\quad \textsf{let } \mathit{oh} = \mathit{TimeLookup}\,(\textsf{mk-}\mathit{TIMEPARSER}`\mathit{Recmod}\,(), \textsf{nil}\,) \textsf{ in}$

.3 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(\mathit{oh} \times \textsf{len } \mathit{mods})];$

A lambda expression requires time dependent on the number of parameters to the lambda expression (which are added to the environment) and a model dependent overhead of lambda expressions.

1811.0 $\mathit{TimeLambdaExpr} : \mathit{AS}`\mathit{LambdaExpr} \to \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeLambdaExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{LambdaExpr}\,(\mathit{parm}, \text{-}, \text{-})) \triangleq$

.2 $\quad \textsf{let } \mathit{oh}1 = \mathit{TimeLookup}\,(\textsf{mk-}\mathit{TIMEPARSER}`\mathit{AddEnv}\,(), \textsf{nil}\,),$

.3 $\qquad \mathit{oh}2 = \mathit{TimeLookup}\,(\textsf{mk-}\mathit{TIMEPARSER}`\mathit{LambdaExpr}\,(), \textsf{nil}\,) \textsf{ in}$

.4 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,((\mathit{oh}1 \times \textsf{len } \mathit{parm}) + \mathit{oh}2)];$

To instantiate a function type, we require time for each of the arguments given.

1812.0 $\mathit{TimeFctTypeInstExpr} : \mathit{AS}`\mathit{FctTypeInstExpr} \to \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeFctTypeInstExpr}\,(\textsf{mk-}\mathit{AS}`\mathit{FctTypeInstExpr}\,(\text{-}, \mathit{inst}, \text{-})) \triangleq$

.2 $\quad \textsf{let } \mathit{oh} = \mathit{TimeLookup}\,(\textsf{mk-}\mathit{TIMEPARSER}`\mathit{Polyinst}\,(), \textsf{nil}\,) \textsf{ in}$

.3 $\quad [\textsf{mk-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(\mathit{oh} \times \textsf{len } \mathit{inst})];$

The time taken for an is expression depends on the type of the argument and the overhead for is expressions on the target model.

1813.0 $TimeIsExpr : AS`IsExpr \rightarrow STKM`SubProgram$
.1 $TimeIsExpr\,(\mathsf{mk\text{-}}AS`IsExpr\,(\text{-}, arg, \text{-})) \triangleq$
.2 $\quad \mathsf{let}\ oh = TimeLookup\,(\mathsf{mk\text{-}}TIMEPARSER`Isexpr\,(), GetType\,(arg))\ \mathsf{in}$
.3 $\quad [\mathsf{mk\text{-}}INSTRTP`INCRTIME\,(oh)];$

To look up a component in a tuple requires a fixed overhead.

1814.0 $TimeTupleSelectExpr : () \rightarrow STKM`SubProgram$
.1 $TimeTupleSelectExpr\,() \triangleq$
.2 $\quad \mathsf{let}\ oh = TimeLookup\,(\mathsf{mk\text{-}}TIMEPARSER`Tupsel\,(), \mathsf{nil}\ )\ \mathsf{in}$
.3 $\quad [\mathsf{mk\text{-}}INSTRTP`INCRTIME\,(oh)];$

A type judgement is considered to be the same as an is expression.

1815.0 $TimeTypeJudgementExpr : AS`TypeJudgementExpr \rightarrow STKM`SubProgram$
.1 $TimeTypeJudgementExpr\,(\mathsf{mk\text{-}}AS`TypeJudgementExpr\,(e, \text{-}, \text{-})) \triangleq$
.2 $\quad \mathsf{let}\ oh = TimeLookup\,(\mathsf{mk\text{-}}TIMEPARSER`Isexpr\,(), GetType\,(e))\ \mathsf{in}$
.3 $\quad [\mathsf{mk\text{-}}INSTRTP`INCRTIME\,(oh)];$

To look up an identifier requires a fixed overhead for a particular model.

1816.0 $TimeNameLookUp : () \rightarrow STKM`SubProgram$
.1 $TimeNameLookUp\,() \triangleq$
.2 $\quad \mathsf{let}\ oh = TimeLookup\,(\mathsf{mk\text{-}}TIMEPARSER`Lookup\,(), \mathsf{nil}\ )\ \mathsf{in}$
.3 $\quad [\mathsf{mk\text{-}}INSTRTP`INCRTIME\,(oh)];$

For the following class-related expressions, the time taken to execute them will be a fixed overhead, specific to each kind of expression.

1817.0 $TimeSelfExpr : () \rightarrow STKM`SubProgram$
.1 $TimeSelfExpr\,() \triangleq$
.2 $\quad \mathsf{let}\ oh = TimeLookup\,(\mathsf{mk\text{-}}TIMEPARSER`Selfexpr\,(), \mathsf{nil}\ )\ \mathsf{in}$
.3 $\quad [\mathsf{mk\text{-}}INSTRTP`INCRTIME\,(oh)];$

1818.0 $TimeNewExpr : AS`NewExpr \rightarrow STKM`SubProgram$
.1 $TimeNewExpr\,(\mathsf{mk\text{-}}AS`NewExpr\,(nm, \text{-})) \triangleq$
.2 $\quad \mathsf{let}\ oh = TimeLookup\,(\mathsf{mk\text{-}}TIMEPARSER`Newobj\,(), \mathsf{nil}\ )\ \mathsf{in}$
.3 $\quad [\mathsf{mk\text{-}}INSTRTP`RUNTIME\text{-}INCRTIME\text{-}NEW\,(oh, nm)];$

1819.0 $TimeIsOfClassExpr : () \to STKM`SubProgram$

.1 $TimeIsOfClassExpr\ () \triangleq$

.2 $\quad$ let $oh = TimeLookup\ (\textsf{mk-}TIMEPARSER`Isofclass\ (), \textsf{nil}\ )$ in

.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\ (oh)];$

1820.0 $TimeIsOfBaseClassExpr : () \to STKM`SubProgram$

.1 $TimeIsOfBaseClassExpr\ () \triangleq$

.2 $\quad$ let $oh = TimeLookup\ (\textsf{mk-}TIMEPARSER`Isofbaseclass\ (), \textsf{nil}\ )$ in

.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\ (oh)];$

1821.0 $TimeSameBaseClassExpr : () \to STKM`SubProgram$

.1 $TimeSameBaseClassExpr\ () \triangleq$

.2 $\quad$ let $oh = TimeLookup\ (\textsf{mk-}TIMEPARSER`Samebaseclass\ (), \textsf{nil}\ )$ in

.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\ (oh)];$

1822.0 $TimeSameClassExpr : () \to STKM`SubProgram$

.1 $TimeSameClassExpr\ () \triangleq$

.2 $\quad$ let $oh = TimeLookup\ (\textsf{mk-}TIMEPARSER`Sameclass\ (), \textsf{nil}\ )$ in

.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\ (oh)];$

1823.0 $TimeThreadIdExpr : () \to STKM`SubProgram$

.1 $TimeThreadIdExpr\ () \triangleq$

.2 $\quad$ let $oh = TimeLookup\ (\textsf{mk-}TIMEPARSER`Threadid\ (), \textsf{nil}\ )$ in

.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\ (oh)];$

For history expressions, we have a fixed overhead regardless of the kind of history expression involved.

1824.0 $TimeHistoryExpr : () \to STKM`SubProgram$

.1 $TimeHistoryExpr\ () \triangleq$

.2 $\quad$ let $oh = TimeLookup\ (\textsf{mk-}TIMEPARSER`History\ (), \textsf{nil}\ )$ in

.3 $\quad [\textsf{mk-}INSTRTP`INCRTIME\ (oh)];$

1.30 Auxiliary Functions

The following functions are used in CEXPR to add time increment instructions for expressions not covered in the previous section.

```
1825.0 MkCbr : () → STKM`SubProgram
    .1 MkCbr () ≜
    .2   if ¬ GetCompilingTime ()
    .3   then []
    .4   else let oh = TimeLookup (mk-TIMEPARSER`Cbr (), nil ) in
    .5        [mk-INSTRTP`INCRTIME (oh)];
```

```
1826.0 MkBr : () → STKM`SubProgram
    .1 MkBr () ≜
    .2   if ¬ GetCompilingTime ()
    .3   then []
    .4   else let oh = TimeLookup (mk-TIMEPARSER`Br (), nil ) in
    .5        [mk-INSTRTP`INCRTIME (oh)];
```

```
1827.0 MkMatchPattern : () → STKM`SubProgram
    .1 MkMatchPattern () ≜
    .2   if ¬ GetCompilingTime ()
    .3   then []
    .4   else let oh = TimeLookup (mk-TIMEPARSER`MatchPattern (), nil ) in
    .5        [mk-INSTRTP`INCRTIME (oh)];
```

```
1828.0 MkSetCompInsert : () → STKM`SubProgram
    .1 MkSetCompInsert () ≜
    .2   if ¬ GetCompilingTime ()
    .3   then []
    .4   else let oh1 = TimeLookup (mk-TIMEPARSER`Addset (), nil ),
    .5            oh2 = TimeLookup (mk-TIMEPARSER`Cbr (), nil ) in
    .6        [mk-INSTRTP`INCRTIME (oh1 + oh2)];
```

```
1829.0 MkSeqCompInsert : () → STKM`SubProgram
    .1 MkSeqCompInsert () ≜
    .2   if ¬ GetCompilingTime ()
    .3   then []
    .4   else let oh1 = TimeLookup (mk-TIMEPARSER`Appendseq (), nil ),
    .5            oh2 = TimeLookup (mk-TIMEPARSER`Cbr (), nil ) in
    .6        [mk-INSTRTP`INCRTIME (oh1 + oh2)];
```

1830.0 $MkMapCompInsert : () \to STKM`SubProgram$
.1 $MkMapCompInsert\ () \triangleq$
.2 if $\neg$ $GetCompilingTime\ ()$
.3 then []
.4 else let $oh1 = TimeLookup\ (\text{mk-}TIMEPARSER`Appendmap\ (), \text{nil}\ ),$
.5 $oh2 = TimeLookup\ (\text{mk-}TIMEPARSER`Cbr\ (), \text{nil}\ )$ in
.6 $[\text{mk-}INSTRTP`INCRTIME\ (oh1 + oh2)];$

The function *TimeLookup* looks up in the time map the time defined for a particular instruction and type. Note that for any instruction i, $mk\_(i, nil)$ should always be in the domain of the time map.

1831.0 $TimeLookup : TIMEPARSER`Instruction \times [REP`BasicTypeRep] \to \mathbb{N}$
.1 $TimeLookup\ (instr, tp) \triangleq$
.2 let $m = CMPL`GetTM\ ()$ in
.3 if mk- $(instr, tp) \in$ dom m
.4 then $m\ (\text{mk-}\ (instr, tp))$
.5 else $m\ (\text{mk-}\ (instr, \text{nil}\ ));$

The function *GetType* corresponds to a reference to the type checker in the actual tool. As this can not be replicated at the specification level, we merely return the default in all cases.

1832.0 $GetType : AS`Expr \to [REP`BasicTypeRep]$
.1 $GetType\ (\text{-}) \triangleq$
.2 nil ;

1.31 Runtime Instructions

As stated above, some expressions can only be resolved at runtime. For such expressions, special runtime increment instructions are defined in *INSTRTP*. In this section we provide support functions for such runtime instructions. These are used by CEXPR, since these runtime instructions have to be embedded within the instructions for the corresponding expressions, so that when they are executed, the runtime values of the arguments can be accessed on the evaluation stack.

The function *IsRuntimePrefixOp* returns true iff the time taken to evaluate the unary operator given depends on the argument to the operator.

```
1833.0 IsRuntimePrefixOp : AS`UnaryOp → 𝔹
    .1 IsRuntimePrefixOp (op) △
    .2   op ∈
    .3   {SETDISTRUNION,
    .4    SETDISTRINTERSECT,
    .5    SETPOWER,
    .6    SEQDISTRCONC,
    .7    SEQELEMS,
    .8    SEQINDICES,
    .9    SEQTAIL,
   .10    MAPDOM,
   .11    MAPRNG,
   .12    MAPDISTRMERGE};
```

MkRuntimePrefixOp is the companion to *IsRuntimePrefixOp*, and generates the appropriate runtime increment instruction for the unary operator given.

```
1834.0 MkRuntimePrefixOp : AS`UnaryOp → STKM`SubProgram
    .1 MkRuntimePrefixOp (opr) △
    .2   if ¬ GetCompilingTime ()
    .3   then []
    .4   else let oh = cases opr :
    .5                   SETDISTRUNION → TimeLookup (mk-TIMEPARSER`Addset (), nil ),
    .6                   SETDISTRINTERSECT,
    .7                   SETPOWER → TimeLookup (mk-TIMEPARSER`Addset (), nil ),
    .8                   SEQDISTRCONC → TimeLookup (mk-TIMEPARSER`Appendseq (), nil ),
    .9                   SEQELEMS → TimeLookup (mk-TIMEPARSER`Addset (), nil ),
   .10                   SEQINDICES → TimeLookup (mk-TIMEPARSER`Addset (), nil ),
   .11                   SEQTAIL → TimeLookup (mk-TIMEPARSER`Appendseq (), nil ),
   .12                   MAPDOM → TimeLookup (mk-TIMEPARSER`Addset (), nil ),
   .13                   MAPRNG → TimeLookup (mk-TIMEPARSER`Addset (), nil ),
   .14                   MAPDISTRMERGE → TimeLookup (mk-TIMEPARSER`Appendmap (), nil )
   .15                 end in
   .16        [mk-INSTRTP`RUNTIME-INCRTIME-PREF (opr, oh)]
   .17 pre IsRuntimePrefixOp (opr) ;
```

IsRuntimeBinaryOp and *MkRuntimeBinaryOp* are the binary counterparts for *IsRuntimePrefixOp* and *MkRuntimePrefixOp*.

1835.0 $\mathit{IsRuntimeBinaryOp} : \mathit{AS}`\mathit{BinaryOp} \to \mathbb{B}$
.1 $\mathit{IsRuntimeBinaryOp}\,(op) \triangleq$
.2 $op \in$
.3 $\{$NUMEXP,
.4 SETUNION,
.5 SETINTERSECT,
.6 SETMINUS,
.7 SUBSET,
.8 PROPERSUBSET,
.9 INSET,
.10 NOTINSET,
.11 SEQCONC,
.12 MAPMERGE,
.13 MAPDOMRESTTO,
.14 MAPDOMRESTBY,
.15 MAPRNGRESTTO,
.16 MAPRNGRESTBY,
.17 COMPOSE$\}$;

1836.0 $\mathit{MkRuntimeBinaryOp} : \mathit{AS}`\mathit{BinaryOp} \to \mathit{STKM}`\mathit{SubProgram}$
.1 $\mathit{MkRuntimeBinaryOp}\,(opr) \triangleq$
.2 if $\neg$ $\mathit{GetCompilingTime}\,()$
.3 then $[]$
.4 else let mk-$(oh1, oh2) =$
.5 cases opr :
.6 NUMEXP $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Mult}\,(),$ nil $),$ nil $)$,
.7 SETUNION $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Addset}\,(),$ nil $),$ nil $)$,
.8 SETINTERSECT $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Addset}\,(),$ nil $),$ nil $)$,
.9 SETMINUS $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Addset}\,(),$ nil $),$ nil $)$,
.10 SUBSET $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Equal}\,(),$ nil $),$ nil $)$,
.11 PROPERSUBSET $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Equal}\,(),$ nil $),$ nil $)$,
.12 INSET $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Equal}\,(),$ nil $),$ nil $)$,
.13 NOTINSET $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Equal}\,(),$ nil $),$ nil $)$,
.14 SEQCONC $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Appendseq}\,(),$ nil $),$ nil $)$,
.15 MAPMERGE $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Appendmap}\,(),$ nil $)$,
.16 $\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Equal}\,(),$ nil $))$,
.17 MAPDOMRESTTO $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Appendmap}\,(),$ nil $),$ nil $)$,
.18 MAPDOMRESTBY $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Appendmap}\,(),$ nil $)$,
.19 $\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Equal}\,(),$ nil $))$,
.20 MAPRNGRESTTO $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Appendmap}\,(),$ nil $)$,
.21 $\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Equal}\,(),$ nil $))$,
.22 MAPRNGRESTBY $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Appendmap}\,(),$ nil $)$,
.23 $\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Equal}\,(),$ nil $))$,
.24 COMPOSE $\to$ mk-$(\mathit{TimeLookup}\,($mk-$\mathit{TIMEPARSER}`\mathit{Appendmap}\,(),$ nil $),$ nil $)$
.25 end in
.26 $[$mk-$\mathit{INSTRTP}`\mathit{RUNTIME\text{-}INCRTIME\text{-}BIN}\,(opr, oh1, oh2)]$;

The function *MkRuntimeSetSeqMap* generates a runtime increment instruction for set range expressions, subsequence expressions and seq/map modification expressions.

1837.0 $MkRuntimeSetSeqMap : AS`SetRangeExpr \mid AS`SubSequenceExpr \mid$
.1 $AS`SeqModifyMapOverrideExpr \rightarrow$
.2 $STKM`SubProgram$
.3 $MkRuntimeSetSeqMap\,(e) \triangleq$
.4 if $\neg\, GetCompilingTime\,()$
.5 then $[]$
.6 else let $oh =$
.7 cases true :
.8 (is-$AS`SetRangeExpr\,(e)) \rightarrow TimeLookup\,($mk-$TIMEPARSER`Addset\,()$, nil),
.9 (is-$AS`SubSequenceExpr\,(e)) \rightarrow TimeLookup\,($mk-$TIMEPARSER`Subseq\,()$, nil),
.10 (is-$AS`SeqModifyMapOverrideExpr\,(e)) \rightarrow$
.11 $TimeLookup\,($mk-$TIMEPARSER`Seqmapover\,()$, nil)
.12 end in
.13 [mk-$INSTRTP`RUNTIME\text{-}INCRTIME\text{-}SETSEQMAP\,(oh)$];

1.32 Statements

The approach for statements is similar to expressions. A difference is that only the startlist statement needs to be resolved at runtime. The remainder can be resolved statically, though if and cases statements are dealt with directly in CSTMT rather than here.

1838.0 $S2Time : AS`Stmt \to STKM`SubProgram$
.1 $S2Time\,(stmt) \triangleq$
.2 cases true :
.3 $(\text{is-}AS`DefStmt\,(stmt)) \to TimeDefStmt\,(stmt),$
.4 $(\text{is-}AS`LetStmt\,(stmt)) \to TimeLetStmt\,(stmt),$
.5 $(\text{is-}AS`LetBeSTStmt\,(stmt)) \to TimeLetBeSTStmt\,(stmt),$
.6 $(\text{is-}AS`AssignStmt\,(stmt)) \to TimeAssignStmt\,(stmt),$
.7 $(\text{is-}AS`AtomicAssignStmt\,(stmt)) \to TimeAtomicAssignStmt\,(stmt),$
.8 $(\text{is-}AS`SeqForLoopStmt\,(stmt)) \to TimeSeqForLoopStmt\,(stmt),$
.9 $(\text{is-}AS`SetForLoopStmt\,(stmt)) \to TimeSetForLoopStmt\,(stmt),$
.10 $(\text{is-}AS`IndexForLoopStmt\,(stmt)) \to TimeIndexForLoopStmt\,(stmt),$
.11 $(\text{is-}AS`WhileLoopStmt\,(stmt)) \to TimeWhileLoopStmt\,(stmt),$
.12 $(\text{is-}AS`CallStmt\,(stmt)) \to TimeCallStmt\,(stmt),$
.13 $(\text{is-}AS`ReturnStmt\,(stmt)) \to TimeReturnStmt\,(stmt),$
.14 $(\text{is-}AS`ExitStmt\,(stmt)) \to TimeExitStmt\,(stmt),$
.15 $(\text{is-}AS`AlwaysStmt\,(stmt)) \to TimeAlwaysStmt\,(stmt),$
.16 $(\text{is-}AS`TrapStmt\,(stmt)) \to TimeTrapStmt\,(stmt),$
.17 $(\text{is-}AS`RecTrapStmt\,(stmt)) \to TimeRecTrapStmt\,(stmt),$
.18 $(\text{is-}AS`StartStmt\,(stmt)) \to TimeStartStmt\,(stmt),$
.19 $(\text{is-}AS`BlockStmt\,(stmt)) \to TimeBlockStmt\,(stmt),$
.20 $(\text{is-}AS`NonDetStmt\,(stmt)) \to TimeNonDetStmt\,(stmt),$
.21 $(\text{is-}AS`DurationStmt\,(stmt)) \to [\,],$
.22 others $\to [\,]$
.23 end;

Statements for introducing local bindings are similar to the corresponding expressions, and need no further explanation.

1839.0 $TimeDefStmt : AS`DefStmt \to STKM`SubProgram$
.1 $TimeDefStmt\,(\text{mk-}AS`DefStmt\,(def\text{-}l, \text{-}, \text{-})) \triangleq$
.2 let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`AddEnv\,(), \text{nil}\,)$ in
.3 $[\text{mk-}INSTRTP`INCRTIME\,(oh \times \text{len}\ def\text{-}l)];$

1840.0 $TimeLetStmt : AS`LetStmt \to STKM`SubProgram$
.1 $TimeLetStmt\,(\text{mk-}AS`LetStmt\,(def\text{-}l, \text{-}, \text{-})) \triangleq$
.2 let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`AddEnv\,(), \text{nil}\,)$ in
.3 $[\text{mk-}INSTRTP`INCRTIME\,(oh \times \text{len}\ def\text{-}l)];$

1841.0 $TimeLetBeSTStmt : AS`LetBeSTStmt \to STKM`SubProgram$
.1 $TimeLetBeSTStmt\,(\text{-}) \triangleq$
.2 let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`LetBeST\,(), \text{nil}\,)$ in
.3 $[\text{mk-}INSTRTP`INCRTIME\,(oh)];$

An assignment statement requires time to update memory and the environment. For an atomic assignment statement, this overhead is multiplied by the number of component assignments.

1842.0 $TimeAssignStmt : AS`AssignStmt \rightarrow STKM`SubProgram$

.1 $TimeAssignStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh1 = TimeLookup\,(\text{mk-}TIMEPARSER`AddEnv\,(), \text{nil}\,),$
.3 $\qquad oh2 = TimeLookup\,(\text{mk-}TIMEPARSER`Update\,(), \text{nil}\,)$ in
.4 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh1 + oh2)];$

1843.0 $TimeAtomicAssignStmt : AS`AtomicAssignStmt \rightarrow STKM`SubProgram$

.1 $TimeAtomicAssignStmt\,(\text{mk-}AS`AtomicAssignStmt\,(assstmtl, \text{-})) \triangleq$
.2 $\quad$ let $oh1 = TimeLookup\,(\text{mk-}TIMEPARSER`AddEnv\,(), \text{nil}\,),$
.3 $\qquad oh2 = TimeLookup\,(\text{mk-}TIMEPARSER`Update\,(), \text{nil}\,)$ in
.4 $\quad [\text{mk-}INSTRTP`INCRTIME\,(\text{len } assstmtl \times (oh1 + oh2))];$

For loops, the time taken is a fixed overhead for the loop statement. In addition the time taken to bind the loop index identifier is accounted for directly in CSTMT by calls to *MkLoopBind*.

1844.0 $TimeSeqForLoopStmt : AS`SeqForLoopStmt \rightarrow STKM`SubProgram$

.1 $TimeSeqForLoopStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Loop\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

1845.0 $TimeSetForLoopStmt : AS`SetForLoopStmt \rightarrow STKM`SubProgram$

.1 $TimeSetForLoopStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Loop\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

1846.0 $TimeIndexForLoopStmt : AS`IndexForLoopStmt \rightarrow STKM`SubProgram$

.1 $TimeIndexForLoopStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Loop\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

1847.0 $TimeWhileLoopStmt : AS`WhileLoopStmt \rightarrow STKM`SubProgram$

.1 $TimeWhileLoopStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Loop\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

1848.0 $MkLoopBind : \mathbb{N} \rightarrow STKM`SubProgram$
.1 $MkLoopBind\,(n) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`AddEnv\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(n \times oh)];$

For call and return statements, the time taken is a fixed overhead for a given model.

1849.0 $TimeCallStmt : AS`CallStmt \rightarrow STKM`SubProgram$
.1 $TimeCallStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Call\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

1850.0 $TimeReturnStmt : AS`ReturnStmt \rightarrow STKM`SubProgram$
.1 $TimeReturnStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Return\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

An exit statement corresponds in terms of timing to an unconditional branch.

1851.0 $TimeExitStmt : AS`ExitStmt \rightarrow STKM`SubProgram$
.1 $TimeExitStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Br\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

For exception handling, the time taken depends on fixed overheads for the model.

1852.0 $TimeAlwaysStmt : AS`AlwaysStmt \rightarrow STKM`SubProgram$
.1 $TimeAlwaysStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Always\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

1853.0 $TimeTrapStmt : AS`TrapStmt \rightarrow STKM`SubProgram$
.1 $TimeTrapStmt\,(\text{-}) \triangleq$
.2 $\quad$ let $oh = TimeLookup\,(\text{mk-}TIMEPARSER`Trap\,(), \text{nil}\,)$ in
.3 $\quad [\text{mk-}INSTRTP`INCRTIME\,(oh)];$

1854.0 $\mathit{TimeRecTrapStmt} : \mathit{AS}`\mathit{RecTrapStmt} \rightarrow \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeRecTrapStmt}\,(\text{-}) \triangleq$

.2 $\quad \mathsf{let}\ \mathit{oh} = \mathit{TimeLookup}\,(\mathsf{mk}\text{-}\mathit{TIMEPARSER}`\mathit{RecTrap}\,(), \mathsf{nil}\,)\ \mathsf{in}$

.3 $\quad [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(\mathit{oh})];$

To start a thread requires a fixed overhead for a given model.

1855.0 $\mathit{TimeStartStmt} : \mathit{AS}`\mathit{StartStmt} \rightarrow \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeStartStmt}\,(\text{-}) \triangleq$

.2 $\quad \mathsf{let}\ \mathit{oh} = \mathit{TimeLookup}\,(\mathsf{mk}\text{-}\mathit{TIMEPARSER}`\mathit{Start}\,(), \mathsf{nil}\,)\ \mathsf{in}$

.3 $\quad [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(\mathit{oh})];$

The time taken to execute a block statement consists of the fixed overhead for block statements, together with the time taken to add any new local state variables.

1856.0 $\mathit{TimeBlockStmt} : \mathit{AS}`\mathit{BlockStmt} \rightarrow \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeBlockStmt}\,(\mathsf{mk}\text{-}\mathit{AS}`\mathit{BlockStmt}\,(\mathit{dcl\text{-}l}, \text{-}, \text{-})) \triangleq$

.2 $\quad \mathsf{let}\ \mathit{oh}1 = \mathit{TimeLookup}\,(\mathsf{mk}\text{-}\mathit{TIMEPARSER}`\mathit{AddEnv}\,(), \mathsf{nil}\,),$

.3 $\qquad \mathit{oh}2 = \mathit{TimeLookup}\,(\mathsf{mk}\text{-}\mathit{TIMEPARSER}`\mathit{Update}\,(), \mathsf{nil}\,),$

.4 $\qquad \mathit{oh}3 = \mathit{TimeLookup}\,(\mathsf{mk}\text{-}\mathit{TIMEPARSER}`\mathit{BlockStmt}\,(), \mathsf{nil}\,)\ \mathsf{in}$

.5 $\quad [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{INCRTIME}\,((\mathsf{len}\ \mathit{dcl\text{-}l} \times (\mathit{oh}1 + \mathit{oh}2)) + \mathit{oh}3)];$

For a non-deterministic statement, the time taken is a fixed overhead.

1857.0 $\mathit{TimeNonDetStmt} : \mathit{AS}`\mathit{NonDetStmt} \rightarrow \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{TimeNonDetStmt}\,(\text{-}) \triangleq$

.2 $\quad \mathsf{let}\ \mathit{oh} = \mathit{TimeLookup}\,(\mathsf{mk}\text{-}\mathit{TIMEPARSER}`\mathit{NonDet}\,(), \mathsf{nil}\,)\ \mathsf{in}$

.3 $\quad [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{INCRTIME}\,(\mathit{oh})];$

The function *MkRuntimeStartList* generates the runtime increment instruction which multiplies the overhead of starting a thread by the number of threads in the argument list.

1858.0 $\mathit{MkRuntimeStartList} : () \rightarrow \mathit{STKM}`\mathit{SubProgram}$

.1 $\mathit{MkRuntimeStartList}\,() \triangleq$

.2 $\quad \mathsf{let}\ \mathit{oh} = \mathit{TimeLookup}\,(\mathsf{mk}\text{-}\mathit{TIMEPARSER}`\mathit{Start}\,(), \mathsf{nil}\,)\ \mathsf{in}$

.3 $\quad [\mathsf{mk}\text{-}\mathit{INSTRTP}`\mathit{RUNTIME\text{-}INCRTIME\text{-}STARTLIST}\,(\mathit{oh})]$

end *TIME*

Test Suite :	rtinfo.ast
Module :	TIME

Name	#Calls	Coverage
TIME‘MkBr	**undefined**	**undefined**
TIME‘MkCbr	**undefined**	**undefined**
TIME‘E2Time	**undefined**	**undefined**
TIME‘S2Time	**undefined**	**undefined**
TIME‘GetType	**undefined**	**undefined**
TIME‘MkLoopBind	**undefined**	**undefined**
TIME‘TimeIsExpr	**undefined**	**undefined**
TIME‘TimeLookup	**undefined**	**undefined**
TIME‘TimeDefExpr	**undefined**	**undefined**
TIME‘TimeDefStmt	**undefined**	**undefined**
TIME‘TimeLetExpr	**undefined**	**undefined**
TIME‘TimeLetStmt	**undefined**	**undefined**
TIME‘TimeNewExpr	**undefined**	**undefined**
TIME‘TimeCallStmt	**undefined**	**undefined**
TIME‘TimeExitStmt	**undefined**	**undefined**
TIME‘TimeIotaExpr	**undefined**	**undefined**
TIME‘TimeSelfExpr	**undefined**	**undefined**
TIME‘TimeTrapStmt	**undefined**	**undefined**
TIME‘TimeApplyExpr	**undefined**	**undefined**
TIME‘TimeBlockStmt	**undefined**	**undefined**
TIME‘TimeStartStmt	**undefined**	**undefined**
TIME‘MkMatchPattern	**undefined**	**undefined**
TIME‘TimeAlwaysStmt	**undefined**	**undefined**
TIME‘TimeAssignStmt	**undefined**	**undefined**
TIME‘TimeBinaryExpr	**undefined**	**undefined**
TIME‘TimeLambdaExpr	**undefined**	**undefined**
TIME‘TimeNameLookUp	**undefined**	**undefined**
TIME‘TimeNonDetStmt	**undefined**	**undefined**
TIME‘TimePrefixExpr	**undefined**	**undefined**
TIME‘TimeReturnStmt	**undefined**	**undefined**
TIME‘MkMapCompInsert	**undefined**	**undefined**
TIME‘MkSeqCompInsert	**undefined**	**undefined**
TIME‘MkSetCompInsert	**undefined**	**undefined**
TIME‘TimeHistoryExpr	**undefined**	**undefined**
TIME‘TimeLetBeSTExpr	**undefined**	**undefined**
TIME‘TimeLetBeSTStmt	**undefined**	**undefined**
TIME‘TimeMapEnumExpr	**undefined**	**undefined**
TIME‘TimeRecTrapStmt	**undefined**	**undefined**
TIME‘TimeSeqEnumExpr	**undefined**	**undefined**
TIME‘TimeSetEnumExpr	**undefined**	**undefined**
TIME‘GetCompilingTime	**undefined**	**undefined**
TIME‘SetCompilingTime	**undefined**	**undefined**
TIME‘TimeThreadIdExpr	**undefined**	**undefined**
TIME‘IsRuntimeBinaryOp	**undefined**	**undefined**
TIME‘IsRuntimePrefixOp	**undefined**	**undefined**
TIME‘MkRuntimeBinaryOp	**undefined**	**undefined**
TIME‘MkRuntimePrefixOp	**undefined**	**undefined**

Name	#Calls	Coverage
TIME‘TimeIsOfClassExpr	**undefined**	**undefined**
TIME‘TimeSameClassExpr	**undefined**	**undefined**
TIME‘TimeWhileLoopStmt	**undefined**	**undefined**
TIME‘MkRuntimeSetSeqMap	**undefined**	**undefined**
TIME‘MkRuntimeStartList	**undefined**	**undefined**
TIME‘TimeMapInverseExpr	**undefined**	**undefined**
TIME‘TimeSeqForLoopStmt	**undefined**	**undefined**
TIME‘TimeSetForLoopStmt	**undefined**	**undefined**
TIME‘TimeAllOrExistsExpr	**undefined**	**undefined**
TIME‘TimeFctTypeInstExpr	**undefined**	**undefined**
TIME‘TimeFieldSelectExpr	**undefined**	**undefined**
TIME‘TimeTupleSelectExpr	**undefined**	**undefined**
TIME‘TimeAtomicAssignStmt	**undefined**	**undefined**
TIME‘TimeExistsUniqueExpr	**undefined**	**undefined**
TIME‘TimeIndexForLoopStmt	**undefined**	**undefined**
TIME‘TimeIsOfBaseClassExpr	**undefined**	**undefined**
TIME‘TimeSameBaseClassExpr	**undefined**	**undefined**
TIME‘TimeTypeJudgementExpr	**undefined**	**undefined**
TIME‘TimeRecordModifierExpr	**undefined**	**undefined**
TIME‘TimeMapComprehensionExpr	**undefined**	**undefined**
TIME‘TimeSeqComprehensionExpr	**undefined**	**undefined**
TIME‘TimeSetComprehensionExpr	**undefined**	**undefined**
TIME‘TimeTupleConstructorExpr	**undefined**	**undefined**
TIME‘TimeRecordConstructorExpr	**undefined**	**undefined**
Total Coverage		**0%**

1.33 Introduction

This module contains the specification of the time trace file. This file contains information about runtime events such as swapping of threads, and requests, activations and completions of operations.

module *TIMETRACETP*

imports

1859.0 from *AS*

1860.0 types *Ids*;

.1 *Name* ,

1861.0 from *SEM*

1862.0 types *OBJ-Ref* ,

1863.0 from *SCHDTP*

1864.0 types *ThreadId* ,

1865.0 from $INSTRTP$

1866.0 types act;
.1 fin;
.2 req;
.3 $active$;
.4 $waiting$;
.5 $HistoryKind$

exports all

definitions

A trace is a sequence of trace items. A trace item is a time stamped trace event, so the overall trace is monotonically increasing with respect to the time of events.

types

1867.0 $Trace = TraceItem^*$

.1 inv $t \triangleq \forall\, i, j \in \mathsf{inds}\ t \cdot$
.2 $i < j \;\Rightarrow\; t\,(i).time \leq t\,(j).time$;

1868.0 $TraceItem :: event : TraceEvent$
.1 $time : \mathbb{N}$;

A trace event may be swapping in or out of a thread, or a request, activation or completion of an operation.

1869.0 $TraceEvent = ThreadSwapIn \mid ThreadSwapOut \mid DelayedThreadSwapIn \mid$
.1 $OpRequest \mid OpActivate \mid OpCompleted$;

A thread swap in event consists solely of a thread id. Similarly for a swap out event.

1870.0 $ThreadSwapIn :: id : SCHDTP`ThreadId$
.1 $objref : [SEM`OBJ\text{-}Ref]$
.2 $clnm : [AS`Ids]$;

1871.0 $DelayedThreadSwapIn :: id : SCHDTP`ThreadId$
.1 $objref : [SEM`OBJ\text{-}Ref]$
.2 $clnm : [AS`Ids]$
.3 $delay : \mathbb{R}$;

1872.0 $ThreadSwapOut :: id : SCHDTP`ThreadId$
.1 $objref : [SEM`OBJ\text{-}Ref]$
.2 $clnm : [AS`Ids]$;

An operation request consists of the name of the operation requested, and the object from which the request originated. Note that the name is fully qualified, and there includes the class in which the operation is defined. Operation activations and completions are similar.

1873.0 $OpRequest :: opname : AS`Ids$
.1 $objref : SEM`OBJ\text{-}Ref$
.2 $clnm : AS`Ids;$

1874.0 $OpActivate :: opname : AS`Ids$
.1 $objref : SEM`OBJ\text{-}Ref$
.2 $clnm : AS`Ids;$

1875.0 $OpCompleted :: opname : AS`Ids$
.1 $objref : SEM`OBJ\text{-}Ref$
.2 $clnm : AS`Ids$

end *TIMETRACETP*

Test Suite : rtinfo.ast
Module : TIMETRACETP

Name	**#Calls**	**Coverage**
Total Coverage		**undefined**

module *TIMETRACE*

imports

1876.0 from *AS*
1877.0 types *Id*;
.1 *Ids*;
.2 *Name* ,
1878.0 from *CI*
1879.0 types *ContextId* ,
1880.0 from *IO* all ,
1881.0 from *SEM*
1882.0 types *OBJ-Ref* ,
1883.0 from *STKM*
1884.0 operations $GetTime : () \xrightarrow{o} \mathbb{R}$,
1885.0 from *SCHDTP*
1886.0 types *ThreadId* ,
1887.0 from *INSTRTP*
1888.0 types *act*;
.1 *fin*;
.2 *req*;
.3 *active*;
.4 *waiting*;
.5 *HistoryKind* ,

1889.0 from $\mathit{TIMETRACETP}$ all

exports

1890.0 operations $\mathit{GetLogfile} : () \xrightarrow{o}$ char*;
.1 $\mathit{SetLogfile}$: char* $\xrightarrow{o} ()$;
.2 $\mathit{LogHistEvent}{:}\mathit{AS}`\mathit{Name}\times\mathit{AS}`\mathit{Ids}\times\mathit{INSTRTP}`\mathit{HistoryKind}\times\mathit{SEM}`\mathit{OBJ\text{-}Ref} \xrightarrow{o}$
$()$;
.3 $\mathit{LogThreadSwapIn}{:}\mathit{SCHDTP}`\mathit{ThreadId}\times[\mathit{SEM}`\mathit{OBJ\text{-}Ref}]\times[\mathit{AS}`\mathit{Name}] \xrightarrow{o}$
$()$;
.4 $\mathit{LogThreadSwapOut}{:}\mathit{SCHDTP}`\mathit{ThreadId}\times[\mathit{SEM}`\mathit{OBJ\text{-}Ref}]\times[\mathit{AS}`\mathit{Name}] \xrightarrow{o}$
$()$;
.5 $\mathit{LogDelayedThreadSwapIn}{:}\mathit{SCHDTP}`\mathit{ThreadId}\times[\mathit{SEM}`\mathit{OBJ\text{-}Ref}]\times[\mathit{AS}`\mathit{Name}]\times$
$\mathbb{R} \xrightarrow{o} ()$

definitions

The trace state consists of a trace, and the name of the file to which trace output should be written. Note that events are recorded both in the state, and also in the log file.

The operation *LogHistEvent* is called from STATE‘UpdateHistCount. It logs events relating to operation requests, activations and completions. Events are logged with the current time obtained from the stack machine.

1891.0 state $\mathit{TraceState}$ of
.1 $\mathit{new\text{-}log} : \mathbb{B}$
.2 $\mathit{logfile}$: char*
.3 init $\mathit{tst} \triangleq \mathit{tst} =$ mk-$\mathit{TraceState}$ (true, "$\mathit{logfile}$")
.4 end

operations

1892.0 $\mathit{LogHistEvent} : \mathit{AS}`\mathit{Name} \times \mathit{AS}`\mathit{Ids} \times \mathit{INSTRTP}`\mathit{HistoryKind} \times \mathit{SEM}`\mathit{OBJ\text{-}Ref} \xrightarrow{o} ()$
.1 $\mathit{LogHistEvent}\,(\mathit{clnm}, \mathit{opname}, \mathit{kind}, \mathit{objref}) \triangleq$
.2 let $\mathit{new\text{-}event} =$
.3 cases true :
.4 (is-$\mathit{INSTRTP}`\mathit{req}\,(\mathit{kind})) \rightarrow$
.5 mk-$\mathit{TIMETRACETP}`\mathit{OpRequest}\,(\mathit{opname}, \mathit{objref}, \mathit{clnm.ids})$,
.6 (is-$\mathit{INSTRTP}`\mathit{act}\,(\mathit{kind})) \rightarrow$
.7 mk-$\mathit{TIMETRACETP}`\mathit{OpActivate}\,(\mathit{opname}, \mathit{objref}, \mathit{clnm.ids})$,
.8 (is-$\mathit{INSTRTP}`\mathit{fin}\,(\mathit{kind})) \rightarrow$
.9 mk-$\mathit{TIMETRACETP}`\mathit{OpCompleted}\,(\mathit{opname}, \mathit{objref}, \mathit{clnm.ids})$,
.10 others $\rightarrow$ undefined
.11 end in
.12 def $\mathit{curtime} = \mathit{STKM}`\mathit{GetTime}\,()$ in
.13 $\mathit{AddToLogfile}($mk-$\mathit{TIMETRACETP}`\mathit{TraceItem}\,(\mathit{new\text{-}event}, \mathit{curtime}))$;

Similarly thread events are logged with the current time. These operations are called from SCHD‘SelAndRunThread.

1893.0 $LogThreadSwapIn : SCHDTP`ThreadId \times [SEM`OBJ\text{-}Ref] \times [AS`Name] \xrightarrow{o} ()$

.1 $LogThreadSwapIn\,(threadid, objRef, clnm) \triangleq$
.2 let $ids =$ if $clnm =$ nil
.3 then nil
.4 else $clnm.ids$ in
.5 def $curtime = STKM`GetTime\,()$ in
.6 $AddToLogfile($mk-$TIMETRACETP`TraceItem$
.7 (
.8 mk-$TIMETRACETP`ThreadSwapIn\,(threadid, objRef, ids),$
.9 $curtime))$;

1894.0 $LogDelayedThreadSwapIn : SCHDTP`ThreadId \times [SEM`OBJ\text{-}Ref] \times [AS`Name] \times \mathbb{R} \xrightarrow{o} ()$

.1 $LogDelayedThreadSwapIn\,(threadid, objRef, clnm, delay) \triangleq$
.2 let $ids =$ if $clnm =$ nil
.3 then nil
.4 else $clnm.ids$ in
.5 def $curtime = STKM`GetTime\,()$ in
.6 $AddToLogfile($mk-$TIMETRACETP`TraceItem$
.7 (
.8 mk-$TIMETRACETP`DelayedThreadSwapIn\,(threadid, objRef,$
.9 $ids, delay),$
.10 $curtime))$;

1895.0 $LogThreadSwapOut : SCHDTP`ThreadId \times [SEM`OBJ\text{-}Ref] \times [AS`Name] \xrightarrow{o} ()$

.1 $LogThreadSwapOut\,(threadid, objRef, clnm) \triangleq$
.2 let $ids =$ if $clnm =$ nil
.3 then nil
.4 else $clnm.ids$ in
.5 def $curtime = STKM`GetTime\,()$ in
.6 $AddToLogfile($mk-$TIMETRACETP`TraceItem$
.7 (
.8 mk-$TIMETRACETP`ThreadSwapOut\,(threadid, objRef, ids),$
.9 $curtime))$;

The operations *SetLogFile* and *GetLogFile* update the name of the logfile. Note that no check is made to ensure that logging has not begun, so this could lead to unpredictable results if called in the middle of execution.

1896.0 $SetLogfile :$ char$^* \xrightarrow{o} ()$

.1 $SetLogfile\,(newlogfile) \triangleq$
.2 $logfile := newlogfile;$

1897.0 $GetLogfile : () \xrightarrow{o} \mathsf{char}^*$

.1 $GetLogfile\ () \triangleq$

.2 $\quad \mathsf{return}\ logfile$

The following functions are used to pretty-print TraceItems. This leads to more efficient use of the logfile.

functions

```
1898.0 TraceItem2String : TIMETRACETP`TraceItem → char*
.1  TraceItem2String (ti) △
.2    cases ti.event :
.3      mk-TIMETRACETP`ThreadSwapIn (id, objref, clnm) →
.4          "ThreadSwapIn- > " ⁀ "ThreadId : " ⁀ Num2String (id) ⁀
.5          "Obj : " ⁀
.6          Objref2String (objref) ⁀
.7          "Class : " ⁀
.8          Clnm2String (clnm) ⁀
.9          "@" ⁀
.10         Num2String (ti.time),
.11     mk-TIMETRACETP`ThreadSwapOut (id, objref, clnm) →
.12         "ThreadSwapOut- > " ⁀ "ThreadId : " ⁀ Num2String (id) ⁀
.13         "Obj : " ⁀
.14         Objref2String (objref) ⁀
.15         "Class : " ⁀
.16         Clnm2String (clnm) ⁀
.17         "@" ⁀
.18         Num2String (ti.time),
.19     mk-TIMETRACETP`DelayedThreadSwapIn (id, objref, clnm, delay) →
.20         "DelayedThreadSwapIn- > " ⁀ "ThreadId : " ⁀ Num2String (id) ⁀
.21         "Obj : " ⁀
.22         Objref2String (objref) ⁀
.23         "Class : " ⁀
.24         Clnm2String (clnm) ⁀
.25         "@" ⁀
.26         Num2String (ti.time) ⁀
.27         "Delayed : " ⁀
.28         Num2String (delay),
.29     mk-TIMETRACETP`OpRequest (opname, objref, clnm) →
.30         "req- > " ⁀ "Op : " ⁀ Ids2String (opname) ⁀
.31         "Obj : " ⁀
.32         Objref2String (objref) ⁀
.33         "Class : " ⁀
.34         Clnm2String (clnm) ⁀
.35         "@" ⁀
.36         Num2String (ti.time),
.37     mk-TIMETRACETP`OpActivate (opname, objref, clnm) →
.38         "act- > " ⁀ "Op : " ⁀ Ids2String (opname) ⁀
.39         "Obj : " ⁀
.40         Objref2String (objref) ⁀
.41         "Class : " ⁀
.42         Clnm2String (clnm) ⁀
.43         "@" ⁀
.44         Num2String (ti.time),
.45     mk-TIMETRACETP`OpCompleted (opname, objref, clnm) →
.46         "fin- > " ⁀ "Op : " ⁀ Ids2String (opname) ⁀
.47         "Obj : " ⁀
.48         Objref2String (objref) ⁀
.49         "Class : " ⁀
.50         Clnm2String (clnm) ⁀
.51         "@" ⁀
.52         Num2String (ti.time)
.53   end;
```

1899.0 $Num2String : \mathbb{R} \rightarrow \mathsf{char}^*$
.1 $Num2String\,(r) \triangleq$
.2 $\quad Int2String\,(\mathsf{floor}\ r);$

1900.0 $Int2String : \mathbb{Z} \rightarrow \mathsf{char}^*$
.1 $Int2String\,(i) \triangleq$
.2 $\quad \mathsf{if}\ i = 0$
.3 $\quad \mathsf{then}\ \texttt{"}0\texttt{"}$
.4 $\quad \mathsf{else\ let}\ i\text{-}digits = Digits\,(i)\ \mathsf{in}$
.5 $\qquad \mathsf{conc}\ [Int2Char\,(i\text{-}digits\,(i)) \mid i \in \mathsf{inds}\ i\text{-}digits];$

1901.0 $Digits : \mathbb{Z} \rightarrow \mathbb{Z}^*$
.1 $Digits\,(i) \triangleq$
.2 $\quad \mathsf{if}\ i = 0$
.3 $\quad \mathsf{then}\ []$
.4 $\quad \mathsf{else}\ Digits\,(i\ \mathsf{div}\ 10) \curvearrowright [i\ \mathsf{mod}\ 10];$

1902.0 $Clnm2String : [AS`Ids] \rightarrow \mathsf{char}^*$
.1 $Clnm2String\,(ids) \triangleq$
.2 $\quad \mathsf{if}\ ids = \mathsf{nil}$
.3 $\quad \mathsf{then}\ \texttt{"}nil\texttt{"}$
.4 $\quad \mathsf{else}\ Ids2String\,(ids);$

1903.0 $Ids2String : AS`Ids \rightarrow \mathsf{char}^*$
.1 $Ids2String\,(ids) \triangleq$
.2 $\quad \mathsf{if\ len}\ ids = 1$
.3 $\quad \mathsf{then\ hd}\ ids$
.4 $\quad \mathsf{else}\ (\mathsf{hd}\ ids) \curvearrowright \texttt{"`"} \curvearrowright Ids2String\,(\mathsf{tl}\ ids);$

```
1904.0  Int2Char : ℤ → char*
    .1  Int2Char (i) △
    .2    cases i :
    .3      0 → "0",
    .4      1 → "1",
    .5      2 → "2",
    .6      3 → "3",
    .7      4 → "4",
    .8      5 → "5",
    .9      6 → "6",
   .10      7 → "7",
   .11      8 → "8",
   .12      9 → "9",
   .13      others → []
   .14    end;
```

```
1905.0  Objref2String : [SEM`OBJ-Ref] → char*
    .1  Objref2String (objref) △
    .2    if objref = nil
    .3    then "nil"
    .4    else let mk-SEM`OBJ-Ref (n) = objref in
    .5          Int2String (n)
```

The operation AddToLogfile is used within the module to add trace items to the log file. It uses the IO module to do this. If the trace is currently empty, a new trace file is begun, otherwise items are appended to the end of the file. This could lead to unexpected behaviour if the name of the log file is changed midway through execution.

operations

```
1906.0  AddToLogfile : TIMETRACETP`TraceItem →o ()
    .1  AddToLogfile (te) △
    .2    let directive = if new-log
    .3                    then START
    .4                    else APPEND in
    .5    let - = IO`fecho (logfile, TraceItem2String (te) ⁀ "\n", directive) in
    .6    if new-log
    .7    then new-log := false
```

end *TIMETRACE*

Test Suite : rtinfo.ast
Module : TIMETRACE

Name	#Calls	Coverage
TIMETRACE`Digits	**undefined**	**undefined**

Name	#Calls	Coverage
TIMETRACE‘Int2Char	**undefined**	**undefined**
TIMETRACE‘GetLogfile	**undefined**	**undefined**
TIMETRACE‘Ids2String	**undefined**	**undefined**
TIMETRACE‘Int2String	**undefined**	**undefined**
TIMETRACE‘Num2String	**undefined**	**undefined**
TIMETRACE‘SetLogfile	**undefined**	**undefined**
TIMETRACE‘Clnm2String	**undefined**	**undefined**
TIMETRACE‘AddToLogfile	**undefined**	**undefined**
TIMETRACE‘LogHistEvent	**undefined**	**undefined**
TIMETRACE‘Objref2String	**undefined**	**undefined**
TIMETRACE‘LogThreadSwapIn	**undefined**	**undefined**
TIMETRACE‘LogThreadSwapOut	**undefined**	**undefined**
TIMETRACE‘TraceItem2String	**undefined**	**undefined**
TIMETRACE‘LogDelayedThreadSwapIn	**undefined**	**undefined**
Total Coverage		**0%**

– CI‘HasTypeInfo(n.cid)

.1 VDM++ Abstract Syntax

module AS

imports

1907.0 from CI all ,

1908.0 from REP all

exports all

definitions

types

1909.0 $VdmFiles = CI`FileId \xrightarrow{m} SpecFile;$

1910.0 $SpecFile :: name : [FileName]$
.1 $vdm : Document$
.2 $tokenci : CI`TokenContextInfo$
.3 $timestamp : [TimeStamp];$

1911.0 $TimeStamp =$ token;

1912.0 $FileName =$ token;

1913.0 $Document = Class^{+};$

1914.0 $\mathit{Class} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{supercls} : \mathit{Name}^*$
.2 $\mathit{defs} : [\mathit{Definitions}]$
.3 $\mathit{useslib} : [\mathit{TextLit}]$
.4 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$

.5 $\mathsf{inv}\ \mathit{cls} \triangleq \mathsf{len}\ \mathit{cls.nm.ids} = 1;$

.1.1 Instance Variables

1915.0 $\mathit{InstanceVarDef} = \mathit{InstAssignDef} \mid \mathit{InstanceInv};$

1916.0 $\mathit{InstAssignDef} :: \mathit{ad} : \mathit{AssignDef}$
.1 $\mathit{access} : \mathit{Access}$
.2 $\mathit{static} : \mathbb{B}$
.3 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1917.0 $\mathit{InstanceInv} :: \mathit{expr} : \mathit{Expr}$
.1 $\mathit{access} : \mathit{Access}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

.1.2 Synchronisation Definitions

1918.0 $\mathit{SyncDef} = \mathit{DeclarativeSync};$

1919.0 $\mathit{DeclarativeSync} = \mathit{Permission} \mid \mathit{Mutex};$

1920.0 $\mathit{Permission} :: \mathit{spec} : \mathit{Name}$
.1 $\mathit{guard} : \mathit{Expr}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1921.0 $\mathit{Mutex} :: \mathit{ops} : [\mathit{NameList}]$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

.1.3 Threads

1922.0 $\mathit{ThreadDef} = \mathit{PerObl} \mid \mathit{Stmt};$

1923.0 $\mathit{PerObl} :: \mathit{dur} : \mathit{NumLit}$
.1 $\mathit{mtd} : \mathit{Name}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

.1.4 Time Variables

1924.0 $\mathit{TimeVarDef} = \mathit{TimeVarDecl} \mid \mathit{Assumption} \mid \mathit{Effect};$

1925.0 $\mathit{TimeVarDecl} :: i : [\mathrm{INPUT}]$
.1 $\mathit{nms} : \mathit{Name}^{+}$
.2 $\mathit{tp} : \mathit{Type}$
.3 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId};$

1926.0 $\mathit{Assumption} :: \mathit{nms} : \mathit{Name}^{+}$
.1 $\mathit{expr} : \mathit{Expr}$
.2 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId};$

1927.0 $\mathit{Effect} :: \mathit{nms} : \mathit{Name}^{+}$
.1 $\mathit{expr} : \mathit{Expr}$
.2 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId};$

1928.0 $\mathit{Access} = \mathrm{DEFAULT\_AS} \mid \mathrm{PRIVATE\_AS} \mid \mathrm{PROTECTED\_AS} \mid \mathrm{PUBLIC\_AS} \mid$
.1 $\mathrm{NOT\_INITIALISED\_AS};$

1929.0 $\mathit{Definitions} :: \mathit{typem} : \mathit{Name} \xrightarrow{m} \mathit{TypeDef}$
.1 $\mathit{valuem} : \mathit{ValueDef}^{*}$
.2 $\mathit{fnm} : \mathit{Name} \xrightarrow{m} \mathit{FnDef}$
.3 $\mathit{opm} : \mathit{Name} \xrightarrow{m} \mathit{OpDef}$
.4 $\mathit{instvars} : \mathit{InstanceVarDef}^{*}$
.5 $\mathit{timevars} : \mathit{TimeVarDef}^{*}$
.6 $\mathit{syncs} : \mathit{SyncDef}^{*}$
.7 $\mathit{thread} : [\mathit{ThreadDef}]$
.8 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId};$

1930.0 $\mathit{StateDef} :: \mathit{tp} : \mathit{CompositeType}$
.1 $\mathit{Inv} : [\mathit{Invariant}]$
.2 $\mathit{Init} : [\mathit{StateInit}]$
.3 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId};$

1931.0 $\mathit{StateInit} :: \mathit{pat} : \mathit{Pattern}$
.1 $\mathit{expr} : \mathit{Expr}$
.2 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId};$

The *StateDef* type definition is really not part of the VDM++ language, however, the type definition is included in order to avoid too many ifdefs in the specification of the dynamic semantics.

.1.5 Types

1932.0 $\mathit{TypeDef} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{shape} : \mathit{Type}$
.2 $\mathit{Inv} : [\mathit{Invariant}]$
.3 $\mathit{access} : \mathit{Access}$
.4 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.5 inv $td \triangleq$ true;

1933.0 $\mathit{Invariant} :: \mathit{pat} : \mathit{Pattern}$
.1 $\mathit{expr} : \mathit{Expr}$
.2 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$;

1934.0 $\mathit{Type} = \mathit{BasicType} \mid \mathit{QuoteType} \mid \mathit{CompositeType} \mid \mathit{UnionType} \mid$
.1 $\mathit{ProductType} \mid \mathit{OptionalType} \mid \mathit{SetType} \mid \mathit{SeqType} \mid$
.2 $\mathit{MapType} \mid \mathit{TypeName} \mid \mathit{FnType} \mid \mathit{BracketedType} \mid$
.3 $\mathit{OpType} \mid \mathit{TypeVar}$;

1935.0 $\mathit{BracketedType} :: \mathit{tp} : \mathit{Type}$
.1 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$;

1936.0 $\mathit{BasicType} ::$ BOOLEAN | NATONE | NAT | INTEGER | RAT | REAL | .1 TOKEN
.2 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$;

1937.0 $\mathit{QuoteType} :: \mathit{lit} : \mathit{QuoteLit}$
.1 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$;

1938.0 $\mathit{CompositeType} :: \mathit{name} : \mathit{Name}$
.1 $\mathit{fields} : \mathit{Field}^*$
.2 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.3 inv $ct \triangleq$ true;

The entry dc in the *Field* type describes if the field is an abstract field.

1939.0 $\mathit{Field} :: \mathit{sel} : [\mathit{Name}]$
.1 $\mathit{type} : \mathit{Type}$
.2 $\mathit{dc} : \mathbb{B}$
.3 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.4 inv $\mathit{field} \triangleq$ if $\mathit{field}.\mathit{sel} \neq$ nil
.5 then len $\mathit{field}.\mathit{sel}.\mathit{ids} = 1$
.6 else true;

1940.0 $\mathit{UnionType} :: \quad \mathit{Type}^*$
.1 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$;

1941.0 $\mathit{ProductType} :: \mathit{tps} : \mathit{Type}^*$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1942.0 $\mathit{OptionalType} :: \quad \mathit{Type}$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1943.0 $\mathit{SetType} :: \mathit{elemtp} : \mathit{Type}$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1944.0 $\mathit{SeqType} = \mathit{Seq0Type} \mid \mathit{Seq1Type};$

1945.0 $\mathit{Seq0Type} :: \mathit{elemtp} : \mathit{Type}$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1946.0 $\mathit{Seq1Type} :: \mathit{elemtp} : \mathit{Type}$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1947.0 $\mathit{MapType} = \mathit{GeneralMapType} \mid \mathit{InjectiveMapType};$

1948.0 $\mathit{GeneralMapType} :: \mathit{mapdom} : \mathit{Type}$
.1 $\mathit{maprng} : \mathit{Type}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1949.0 $\mathit{InjectiveMapType} :: \mathit{mapdom} : \mathit{Type}$
.1 $\mathit{maprng} : \mathit{Type}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1950.0 $\mathit{TypeName} :: \mathit{name} : \mathit{Name}$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1951.0 $\mathit{DynFnType} = \mathit{FnType} \mid \mathit{LambdaFnType};$

1952.0 $\mathit{LambdaFnType} :: \mathit{fndom} : \mathit{DiscretionaryType}$
.1 $\mathit{fnrng} : \mathit{Type} \mid \mathit{AllType};$

1953.0 $\mathit{FnType} = \mathit{PartialFnType} \mid \mathit{TotalFnType};$

1954.0 $\mathit{PartialFnType} :: \mathit{fndom} : \mathit{DiscretionaryType}$
.1 $\mathit{fnrng} : \mathit{Type}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1955.0 $\mathit{TotalFnType} :: \mathit{fndom} : \mathit{DiscretionaryType}$
.1 $\mathit{fnrng} : \mathit{Type}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId};$

1956.0 $\mathit{DiscretionaryType} = \mathit{Type}^*;$

1957.0 $\mathit{TypeVar} :: \quad \mathit{Name}$
.1 $\quad \mathit{cid} :- \mathit{CI}`\mathit{ContextId};$

1958.0 $\mathit{AllType} :: \quad ;$

.1.6 Functions

1959.0 $\mathit{FnDef} = \mathit{ExplFnDef} \mid \mathit{ImplFnDef} \mid \mathit{ExtExplFnDef};$

1960.0 $\mathit{ExplFnDef} :: \mathit{nm} : \mathit{Name}$
.1 $\quad \mathit{tpparms} : \mathit{TypeVarList}$
.2 $\quad \mathit{tp} : \mathit{FnType}$
.3 $\quad \mathit{parms} : \mathit{ParametersList}$
.4 $\quad \mathit{body} : \mathit{FnBody}$
.5 $\quad \mathit{fnpre} : [\mathit{Expr}]$
.6 $\quad \mathit{fnpost} : [\mathit{Expr}]$
.7 $\quad \mathit{access} : \mathit{Access}$
.8 $\quad \mathit{static} : \mathbb{B}$
.9 $\quad \mathit{cid} :- \mathit{CI}`\mathit{ContextId}$
.10 $\mathsf{inv}\ \mathit{fndef} \triangleq \mathsf{len}\ \mathit{fndef.nm.ids} = 1;$

1961.0 $\mathit{ImplFnDef} :: \mathit{nm} : \mathit{Name}$
.1 $\quad \mathit{params} : \mathit{TypeVarList}$
.2 $\quad \mathit{partps} : \mathit{ParameterTypes}$
.3 $\quad \mathit{resnmtps} : \mathit{NameType}^*$
.4 $\quad \mathit{fnpre} : [\mathit{Expr}]$
.5 $\quad \mathit{fnpost} : \mathit{Expr}$
.6 $\quad \mathit{access} : \mathit{Access}$
.7 $\quad \mathit{static} : \mathbb{B}$
.8 $\quad \mathit{cid} :- \mathit{CI}`\mathit{ContextId}$
.9 $\mathsf{inv}\ \mathit{fndef} \triangleq \mathsf{len}\ \mathit{fndef.nm.ids} = 1;$

1962.0 $\mathit{ExtExplFnDef} :: \mathit{nm} : \mathit{Name}$
.1 $\quad \mathit{params} : \mathit{TypeVarList}$
.2 $\quad \mathit{partps} : \mathit{ParameterTypes}$
.3 $\quad \mathit{resnmtps} : \mathit{NameType}^*$
.4 $\quad \mathit{body} : \mathit{FnBody}$
.5 $\quad \mathit{fnpre} : [\mathit{Expr}]$
.6 $\quad \mathit{fnpost} : [\mathit{Expr}]$
.7 $\quad \mathit{access} : \mathit{Access}$
.8 $\quad \mathit{static} : \mathbb{B}$
.9 $\quad \mathit{cid} :- \mathit{CI}`\mathit{ContextId};$

1963.0 $\mathit{TypeVarList} = \mathit{TypeVar}^*;$

1964.0 $\mathit{ParametersList} = \mathit{Parameters}^*;$

1965.0 $Parameters = Pattern^*;$

1966.0 $ParameterTypes = PatTypePair^*;$

1967.0 $NameType :: nm : Name$
.1 $tp : Type$
.2 cid :- $CI`ContextId$
.3 inv $nt \triangleq$ len $nt.nm.ids = 1;$

1968.0 $PatTypePair :: pats : Pattern^*$
.1 $tp : Type$
.2 cid :- $CI`ContextId;$

1969.0 $dummy =$ NOTYETSPEC;

1970.0 $FnBody :: body : Expr$ | NOTYETSPEC | .1 SUBRESP
.2 cid :- $CI`ContextId;$

.1.7 Operations

1971.0 $OpDef = ExplOpDef \mid ImplOpDef \mid ExtExplOpDef;$

A constructor is an operation which (1) has the same name as the current class name; (2) has no return type.

1972.0 $ExplOpDef :: nm : Name$
.1 $tp : OpType$
.2 $parms : Parameters$
.3 $body : OpBody$
.4 $oppre : [Expr]$
.5 $oppost : [Expr]$
.6 $access : Access$
.7 $static : \mathbb{B}$
.8 $constr : \mathbb{B}$
.9 cid :- $CI`ContextId;$

The field *timepost* in *ExplOpDef* will always be nil in VDM-SL specifications.

1973.0 $\mathit{ImplOpDef} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{partps} : \mathit{ParameterTypes}$
.2 $\mathit{resnmtps} : \mathit{NameType}^*$
.3 $\mathit{opext} : \mathit{ExtInf}^*$
.4 $\mathit{oppre} : [\mathit{Expr}]$
.5 $\mathit{oppost} : \mathit{Expr}$
.6 $\mathit{excps} : \mathit{Error}^*$
.7 $\mathit{access} : \mathit{Access}$
.8 $\mathit{static} : \mathbb{B}$
.9 $\mathit{constr} : \mathbb{B}$
.10 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId};$

1974.0 $\mathit{ExtExplOpDef} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{partps} : \mathit{ParameterTypes}$
.2 $\mathit{resnmtps} : \mathit{NameType}^*$
.3 $\mathit{body} : \mathit{OpBody}$
.4 $\mathit{opext} : \mathit{ExtInf}^*$
.5 $\mathit{oppre} : [\mathit{Expr}]$
.6 $\mathit{oppost} : [\mathit{Expr}]$
.7 $\mathit{excps} : \mathit{Error}^*$
.8 $\mathit{access} : \mathit{Access}$
.9 $\mathit{static} : \mathbb{B}$
.10 $\mathit{constr} : \mathbb{B}$
.11 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId};$

The field *timepost* in *ExtExplOpDef* will always be nil in VDM-SL specifications.

1975.0 $\mathit{OpType} :: \mathit{opdom} : \mathit{Type}^*$
.1 $\mathit{oprng} : [\mathit{Type}]$
.2 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId};$

1976.0 $\mathit{ExtInf} :: \mathit{mode} : \mathit{Mode}$
.1 $\mathit{vars} : \mathit{Name}^*$
.2 $\mathit{tp} : [\mathit{Type}]$
.3 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId};$

1977.0 $\mathit{Mode} = \mathrm{READ} \mid \mathrm{READWRITE};$

1978.0 $\mathit{OpBody} :: \mathit{body} : \mathit{Stmt} \mid \mathrm{NOTYETSPEC} \mid$ .1 $\mathrm{SUBRESP}$
.2 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId};$

.1.8 Values

1979.0 $\mathit{ValueDef} :: \mathit{pat} : \mathit{Pattern}$
.1 $\mathit{tp} : [\mathit{Type}]$
.2 $\mathit{val} : \mathit{Expr}$
.3 $\mathit{access} : \mathit{Access}$
.4 $\mathit{static} : \mathbb{B}$
.5 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

.1.9 Expressions

1980.0 $\mathit{Expr} = \mathit{BracketedExpr} \mid \mathit{DefExpr} \mid \mathit{LetExpr} \mid \mathit{LetBeSTExpr} \mid \mathit{IfExpr} \mid \mathit{CasesExpr} \mid$
.1 $\mathit{UnaryExpr} \mid \mathit{BinaryExpr} \mid \mathit{QuantExpr} \mid \mathit{IotaExpr} \mid$
.2 $\mathit{SetEnumerationExpr} \mid \mathit{SetComprehensionExpr} \mid \mathit{SetRangeExpr} \mid$
.3 $\mathit{SeqEnumerationExpr} \mid \mathit{SeqComprehensionExpr} \mid$
.4 $\mathit{SubSequenceExpr} \mid \mathit{SeqModifyMapOverrideExpr} \mid$
.5 $\mathit{MapEnumerationExpr} \mid \mathit{MapComprehensionExpr} \mid$
.6 $\mathit{TupleConstructorExpr} \mid$
.7 $\mathit{RecordConstructorExpr} \mid \mathit{RecordModifierExpr} \mid$
.8 $\mathit{FieldSelectExpr} \mid \mathit{ApplyExpr} \mid$
.9 $\mathit{IsExpr} \mid \mathit{Literal} \mid \mathit{Name} \mid \mathit{OldName} \mid \mathit{UndefinedExpr} \mid$
.10 $\mathit{TupleSelectExpr} \mid \mathit{TypeJudgementExpr} \mid \mathit{PreConditionApplyExpr} \mid$
.11 $\mathit{SelfExpr} \mid \mathit{NewExpr} \mid \mathit{IsOfClassExpr} \mid \mathit{IsOfBaseClassExpr} \mid$
.12 $\mathit{SameBaseClassExpr} \mid \mathit{SameClassExpr} \mid$
.13 $\mathit{ActExpr} \mid \mathit{FinExpr} \mid \mathit{ActiveExpr} \mid \mathit{WaitingExpr} \mid \mathit{ReqExpr} \mid$
.14 $\mathit{ThreadIdExpr} \mid \mathit{GuardExpr} \mid$
.15 $\mathit{TokenConstructorExpr} \mid \mathit{FctTypeInstExpr} \mid \mathit{LambdaExpr} \mid \mathit{LastRes};$

1981.0 $\mathit{GuardExpr} :: \mathit{AS}`\mathit{Expr}$
.1 $\mathit{AS}`\mathit{Name};$

1982.0 $\mathit{BracketedExpr} :: \mathit{expr} : \mathit{Expr}$
.1 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.2 inv $n \triangleq$ true;

1983.0 $\mathit{DefExpr} :: \mathit{Def} : (\mathit{PatternBind} \times \mathit{Expr})^*$
.1 $\mathit{In} : \mathit{Expr}$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

1984.0 $\mathit{LetExpr} :: \mathit{localdef} : \mathit{LocalDef}^*$
.1 $\mathit{body} : \mathit{Expr}$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

1985.0 $\mathit{LocalDef} = \mathit{FnDef} \mid \mathit{ValueDef};$

1986.0 $LetBeSTExpr :: lhs : Bind$
.1 $St : [Expr]$
.2 $In : Expr$
.3 $cid :\text{-} CI`ContextId$

.4 inv $n \triangleq$ true;

1987.0 $IfExpr :: test : Expr$
.1 $cons : Expr$
.2 $elsif : ElseifExpr^*$
.3 $altn : Expr$
.4 $cid :\text{-} CI`ContextId$

.5 inv $n \triangleq$ true;

1988.0 $ElseifExpr :: test : Expr$
.1 $cons : Expr$
.2 $cid :\text{-} CI`ContextId;$

1989.0 $CasesExpr :: sel : Expr$
.1 $altns : CaseAltn^*$
.2 $Others : [Expr]$
.3 $cid :\text{-} CI`ContextId$

.4 inv $n \triangleq$ true;

1990.0 $CaseAltn :: match : Pattern^*$
.1 $body : Expr$
.2 $cid :\text{-} CI`ContextId;$

1991.0 $UnaryExpr = PrefixExpr \mid MapInverseExpr;$

1992.0 $PrefixExpr :: opr : UnaryOp$
.1 $arg : Expr$
.2 $cid :\text{-} CI`ContextId$

.3 inv $n \triangleq$ true;

1993.0 $UnaryOp =$ NUMPLUS | NUMMINUS | NUMABS | FLOOR |
.1 NOT |
.2 SETCARD | SETDISTRUNION | SETDISTRINTERSECT |
.3 SETPOWER |
.4 SEQLEN | SEQDISTRCONC | SEQELEMS | SEQINDICES |
.5 SEQTAIL | SEQHEAD |
.6 MAPDOM | MAPRNG | MAPDISTRMERGE;

1994.0 $MapInverseExpr :: op : Expr$
.1 $cid :\text{-} CI`ContextId$

.2 inv $n \triangleq$ true;

1995.0 $\mathit{BinaryExpr} :: \mathit{left} : \mathit{Expr}$
.1 $\mathit{opr} : \mathit{BinaryOp}$
.2 $\mathit{right} : \mathit{Expr}$
.3 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.4 inv $n \triangleq$ true;

1996.0 $\mathit{BinaryOp} =$ NUMPLUS | NUMMINUS | NUMMULT | NUMDIV |
.1 NUMREM | NUMMOD | NUMEXP | INTDIV |
.2 NUMLT | NUMLE | NUMGT | NUMGE |
.3 AND | OR | IMPLY | EQUIV |
.4 EQ | NE |
.5 SETUNION | SETINTERSECT | SETMINUS | SUBSET |
.6 PROPERSUBSET | INSET | NOTINSET |
.7 SEQCONC |
.8 MAPMERGE | MAPDOMRESTTO | MAPDOMRESTBY |
.9 MAPRNGRESTTO | MAPRNGRESTBY |
.10 COMPOSE;

1997.0 $\mathit{QuantExpr} = \mathit{AllOrExistsExpr} \mid \mathit{ExistsUniqueExpr};$

1998.0 $\mathit{AllOrExistsExpr} :: \mathit{quant} : \mathit{AllOrExistsQuantifier}$
.1 $\mathit{bind} : \mathit{BindList}$
.2 $\mathit{pred} : \mathit{Expr}$
.3 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.4 inv $n \triangleq$ true;

1999.0 $\mathit{ExistsUniqueExpr} :: \mathit{bind} : \mathit{Bind}$
.1 $\mathit{pred} : \mathit{Expr}$
.2 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

2000.0 $\mathit{AllOrExistsQuantifier} =$ ALL | EXISTS;

2001.0 $\mathit{IotaExpr} :: \mathit{bind} : \mathit{Bind}$
.1 $\mathit{pred} : \mathit{Expr}$
.2 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

2002.0 $\mathit{SetEnumerationExpr} :: \mathit{els} : \mathit{Expr}^*$
.1 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.2 inv $n \triangleq$ true;

2003.0 $\mathit{SetComprehensionExpr} :: \mathit{elem} : \mathit{Expr}$
.1 $\mathit{bind} : \mathit{BindList}$
.2 $\mathit{pred} : [\mathit{Expr}]$
.3 $\mathit{cid} : \text{-} \mathit{CI}`\mathit{ContextId}$

.4 inv $n \triangleq$ true;

2004.0 $SetRangeExpr :: lb : Expr$
.1 $ub : Expr$
.2 cid :- $CI`ContextId$

.3 inv $n \triangleq$ true;

2005.0 $SeqEnumerationExpr :: els : Expr^*$
.1 cid :- $CI`ContextId$

.2 inv $n \triangleq$ true;

2006.0 $SeqComprehensionExpr :: elem : Expr$
.1 $bind : SetBind$
.2 $pred : [Expr]$
.3 cid :- $CI`ContextId$

.4 inv $n \triangleq$ true;

2007.0 $SubSequenceExpr :: sequence : Expr$
.1 $frompos : Expr$
.2 $topos : Expr$
.3 cid :- $CI`ContextId$

.4 inv $n \triangleq$ true;

2008.0 $SeqModifyMapOverrideExpr :: seqmap : Expr$
.1 $mapexp : Expr$
.2 cid :- $CI`ContextId$

.3 inv $n \triangleq$ true;

2009.0 $MapEnumerationExpr :: els : Maplet^*$
.1 cid :- $CI`ContextId$

.2 inv $n \triangleq$ true;

2010.0 $Maplet :: mapdom : Expr$
.1 $maprng : Expr$
.2 cid :- $CI`ContextId$;

2011.0 $MapComprehensionExpr :: elem : Maplet$
.1 $bind : BindList$
.2 $pred : [Expr]$
.3 cid :- $CI`ContextId$

.4 inv $n \triangleq$ true;

2012.0 $TupleConstructorExpr :: fields : Expr^*$
.1 cid :- $CI`ContextId$

.2 inv $n \triangleq$ true;

2013.0 $TokenConstructorExpr :: field : Expr$
.1 $cid :\text{-}\ CI`ContextId$

.2 inv $n \triangleq$ true;

2014.0 $RecordConstructorExpr :: tag : Name$
.1 $fields : Expr^*$
.2 $cid :\text{-}\ CI`ContextId$

.3 inv $n \triangleq$ true;

2015.0 $RecordModifierExpr :: rec : Expr$
.1 $modifiers : RecordModification^+$
.2 $cid :\text{-}\ CI`ContextId$

.3 inv $n \triangleq$ true;

2016.0 $RecordModification :: field : Name$
.1 $new : Expr$
.2 $cid :\text{-}\ CI`ContextId$

.3 inv $rm \triangleq$ len $rm.field.ids = 1$;

2017.0 $FieldSelectExpr :: rec : Expr$
.1 $nm : Name \mid FctTypeInstExpr$
.2 $cid :\text{-}\ CI`ContextId$

.3 inv $n \triangleq$
.4 $(($is-$Name\,(n.nm) \wedge$ len $n.nm.ids \leq 2) \vee$
.5 $($is-$FctTypeInstExpr\,(n.nm) \wedge$ len $n.nm.polyfct.ids \leq 2)) \wedge$
.6 true;

2018.0 $FctTypeInstExpr :: polyfct : Name$
.1 $inst : Type^*$
.2 $cid :\text{-}\ CI`ContextId$

.3 inv $n \triangleq$ true;

2019.0 $LambdaExpr :: parm : TypeBind^*$
.1 $body : Expr$
.2 $cid :\text{-}\ CI`ContextId$

.3 inv $n \triangleq$ true;

2020.0 $ApplyExpr :: fct : Expr$
.1 $arg : Expr^*$
.2 $cid :\text{-}\ CI`ContextId$

.3 inv $n \triangleq$ true;

2021.0 $\mathit{IsExpr} :: \mathit{type} : \mathit{BasicType} \mid \mathit{Name}$
.1 $\mathit{arg} : \mathit{Expr}$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

2022.0 $\mathit{UndefinedExpr} :: \mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

2023.0 $\mathit{TupleSelectExpr} :: \mathit{tuple} : \mathit{Expr}$
.1 $\mathit{no} : \mathit{RealLit}$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

2024.0 $\mathit{TypeJudgementExpr} :: \mathit{expr} : \mathit{Expr}$
.1 $\mathit{type} : \mathit{Type}$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

2025.0 $\mathit{PreConditionApplyExpr} :: \mathit{fct} : \mathit{Expr}$
.1 $\mathit{arg} : \mathit{Expr}^*$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

2026.0 $\mathit{SelfExpr} :: \mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.1 inv $n \triangleq$ true;

2027.0 $\mathit{ThreadIdExpr} :: \mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId};$

2028.0 $\mathit{NewExpr} :: \mathit{cls} : \mathit{Name}$
.1 $\mathit{args} : \mathit{Expr}^*$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.3 inv $ns \triangleq$ len $ns.cls.ids = 1;$

2029.0 $\mathit{IsOfClassExpr} :: \mathit{class} : \mathit{Name}$
.1 $\mathit{arg} : \mathit{Expr}$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ len $n.class.ids = 1 \wedge$ true;

2030.0 $\mathit{IsOfBaseClassExpr} :: \mathit{class} : \mathit{Name}$
.1 $\mathit{arg} : \mathit{Expr}$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ len $n.class.ids = 1 \wedge$ true;

2031.0 $\mathit{SameBaseClassExpr} :: \mathit{expr}1 : \mathit{Expr}$
.1 $\mathit{expr}2 : \mathit{Expr}$
.2 $\mathit{cid} : \text{-}\mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

2032.0 $SameClassExpr :: expr1 : Expr$
.1 $expr2 : Expr$
.2 cid : -CI‘$ContextId$

.3 inv $n \triangleq$ true;

2033.0 $ActExpr :: mthd : NameList$
.1 cid : -CI‘$ContextId$;

2034.0 $FinExpr :: mthd : NameList$
.1 cid : -CI‘$ContextId$;

2035.0 $ActiveExpr :: mthd : NameList$
.1 cid : -CI‘$ContextId$;

2036.0 $WaitingExpr :: mthd : NameList$
.1 cid : -CI‘$ContextId$;

2037.0 $ReqExpr :: mthd : NameList$
.1 cid : -CI‘$ContextId$;

.1.10 Names

2038.0 $Name :: ids : Ids$
.1 cid : -CI‘$ContextId$

.2 inv $n \triangleq$ true;

2039.0 $Ids = Id^*$;

2040.0 $Id = \mathsf{char}^*$;

2041.0 $NameList = Name^+$;

2042.0 $OldName :: ids : Id^*$
.1 cid : -CI‘$ContextId$

.2 inv $n \triangleq$ true;

.1.11 Statements

2043.0 $Stmt = DefStmt \mid LetStmt \mid LetBeSTStmt \mid$
.1 $AssignStmt \mid SeqForLoopStmt \mid AtomicAssignStmt \mid$
.2 $SetForLoopStmt \mid IndexForLoopStmt \mid WhileLoopStmt \mid$
.3 $ReturnStmt \mid IfStmt \mid$
.4 $CasesStmt \mid BlockStmt \mid IdentStmt \mid$
.5 $ErrorStmt \mid AlwaysStmt \mid CallStmt \mid SpecificationStmt \mid$
.6 $StartStmt \mid StartListStmt \mid$
.7 $DurationStmt \mid$
.8 $ExitStmt \mid TrapStmt \mid RecTrapStmt \mid NonDetStmt;$

2044.0 $DefStmt :: value : (PatternBind \times Expr)^*$
.1 $In : Stmt$
.2 $cid :\text{-}\ CI`ContextId;$

2045.0 $LetStmt :: localdef : LocalDef^*$
.1 $In : Stmt$
.2 $cid :\text{-}\ CI`ContextId;$

2046.0 $LetBeSTStmt :: bind : Bind$
.1 $St : [Expr]$
.2 $In : Stmt$
.3 $cid :\text{-}\ CI`ContextId;$

2047.0 $BlockStmt :: dcls : AssignDef^*$
.1 $stmts : Stmt^*$
.2 $cid :\text{-}\ CI`ContextId;$

2048.0 $AssignDef :: var : Name$
.1 $tp : Type$
.2 $dclinit : [Expr]$
.3 $cid :\text{-}\ CI`ContextId;$

2049.0 $AtomicAssignStmt :: atm : AssignStmt^*$
.1 $cid :\text{-}\ CI`ContextId;$

2050.0 $AssignStmt :: lhs : StateDesignator$
.1 $rhs : Expr$
.2 $cid :\text{-}\ CI`ContextId;$

2051.0 $StateDesignator = Name \mid FieldRef \mid MapOrSeqRef;$

2052.0 $FieldRef :: var : StateDesignator$
.1 $sel : Name$
.2 $cid :\text{-}\ CI`ContextId;$

2053.0 *MapOrSeqRef* :: *var* : *StateDesignator*
.1 *arg* : *Expr*
.2 *cid* :- *CI*‘*ContextId*;

2054.0 *SeqForLoopStmt* :: *cv* : *PatternBind*
.1 *dirn* : [REVERSE]
.2 *fseq* : *Expr*
.3 *body* : *Stmt*
.4 *cid* :- *CI*‘*ContextId*;

2055.0 *SetForLoopStmt* :: *cv* : *Pattern*
.1 *fset* : *Expr*
.2 *body* : *Stmt*
.3 *cid* :- *CI*‘*ContextId*;

2056.0 *IndexForLoopStmt* :: *cv* : *Name*
.1 *lb* : *Expr*
.2 *ub* : *Expr*
.3 *By* : [*Expr*]
.4 *body* : *Stmt*
.5 *cid* :- *CI*‘*ContextId*

.6 inv *ifls* $\triangleq$ len *ifls.cv.ids* $= 1$;

2057.0 *WhileLoopStmt* :: *test* : *Expr*
.1 *body* : *Stmt*
.2 *cid* :- *CI*‘*ContextId*;

2058.0 *ReturnStmt* :: *val* : [*Expr*]
.1 *cid* :- *CI*‘*ContextId*;

2059.0 *IfStmt* :: *test* : *Expr*
.1 *cons* : *Stmt*
.2 *elsif* : *ElseifStmt**
.3 *altn* : [*Stmt*]
.4 *cid* :- *CI*‘*ContextId*;

2060.0 *ElseifStmt* :: *test* : *Expr*
.1 *cons* : *Stmt*
.2 *cid* :- *CI*‘*ContextId*;

2061.0 *Error* :: *nm* : *Name*
.1 *cond* : *Expr*
.2 *action* : *Expr*
.3 *cid* :- *CI*‘*ContextId*;

2062.0 *ErrorStmt* :: *cid* :- *CI*‘*ContextId*;

2063.0 $AlwaysStmt :: Post : Stmt$
.1 $body : Stmt$
.2 cid :- $CI`ContextId$;

2064.0 $TrapStmt :: pat : PatternBind$
.1 $Post : Stmt$
.2 $body : Stmt$
.3 cid :- $CI`ContextId$;

2065.0 $RecTrapStmt :: traps : Trap^*$
.1 $body : Stmt$
.2 cid :- $CI`ContextId$;

2066.0 $Trap :: match : PatternBind$
.1 $trappost : Stmt$
.2 cid :- $CI`ContextId$;

2067.0 $ExitStmt :: expr : [Expr]$
.1 cid :- $CI`ContextId$;

2068.0 $NonDetStmt :: stmts : Stmt^*$
.1 cid :- $CI`ContextId$;

2069.0 $CallStmt :: obj : [Expr]$
.1 $oprt : Name$
.2 $args : Expr^*$
.3 cid :- $CI`ContextId$
.4 inv $n \triangleq$ true;

2070.0 $CasesStmt :: sel : Expr$
.1 $altns : CasesStmtAltn^*$
.2 $Others : [Stmt]$
.3 cid :- $CI`ContextId$;

2071.0 $CasesStmtAltn :: match : Pattern^*$
.1 $body : Stmt$
.2 cid :- $CI`ContextId$;

2072.0 $IdentStmt :: cid$:- $CI`ContextId$;

2073.0 $SpecificationStmt :: opext : ExtInf^*$
.1 $oppre : [Expr]$
.2 $oppost : Expr$
.3 $excps : Error^*$
.4 cid :- $CI`ContextId$;

2074.0 $StartStmt :: expr : Expr$
.1 cid : -CI‘$ContextId$;

2075.0 $StartListStmt :: expr : Expr$
.1 cid : -CI‘$ContextId$;

2076.0 $DurationStmt :: num : Expr$
.1 $stmt : Stmt$
.2 cid : -CI‘$ContextId$;

.1.12 Patterns

2077.0 $Pattern = PatternName \mid MatchVal \mid SetPattern \mid SeqPattern \mid$
.1 $TuplePattern \mid RecordPattern$;

2078.0 $PatternName :: nm : [Name] \mid OldName$
.1 cid : -CI‘$ContextId$
.2 inv $n \triangleq$ true;

The *OldName* type in *PatternName* is used by the code generator, in order to code generate post conditions on operations. Note, that old names can not appear in pattern names in the concrete syntax.

2079.0 $MatchVal :: val : Expr$
.1 cid : -CI‘$ContextId$;

2080.0 $SetPattern = SetEnumPattern \mid SetUnionPattern$;

The empty set is represented using *MatchVal* and not *SetEnumPattern*. Consequently *SetEnumPattern* contains $Pattern^+$ instead of $Pattern^*$.

2081.0 $SetEnumPattern :: Elems : Pattern^+$
.1 cid : -CI‘$ContextId$
.2 inv $n \triangleq$ true;

2082.0 $SetUnionPattern :: lp : Pattern$
.1 $rp : Pattern$
.2 cid : -CI‘$ContextId$;

2083.0 $SeqPattern = SeqEnumPattern \mid SeqConcPattern$;

The empty sequence is represented using *MatchVal* and not *SeqEnumPattern*. Consequently *SeqEnumPattern* contains $Pattern^+$ instead of $Pattern^*$.

2084.0 $SeqEnumPattern :: els : Pattern^{+}$
.1 $cid : \text{-} CI`ContextId$

.2 inv $n \triangleq$ true;

2085.0 $SeqConcPattern :: lp : Pattern$
.1 $rp : Pattern$
.2 $cid : \text{-} CI`ContextId;$

2086.0 $TuplePattern :: fields : Pattern^{*}$
.1 $cid : \text{-} CI`ContextId$

.2 inv $n \triangleq$ true;

2087.0 $RecordPattern :: nm : Name$
.1 $fields : Pattern^{*}$
.2 $cid : \text{-} CI`ContextId$

.3 inv $n \triangleq$ true;

.1.13 Bindings

2088.0 $PatternBind = Pattern \mid Bind;$

2089.0 $Bind = SetBind \mid TypeBind;$

2090.0 $SetBind :: pat : Pattern$
.1 $Set : Expr$
.2 $cid : \text{-} CI`ContextId;$

2091.0 $TypeBind :: pat : Pattern$
.1 $tp : Type$
.2 $cid : \text{-} CI`ContextId;$

2092.0 $BindList = MultBind^{*};$

2093.0 $MultBind = MultSetBind \mid MultTypeBind;$

2094.0 $MultSetBind :: pat : Pattern^{*}$
.1 $Set : Expr$
.2 $cid : \text{-} CI`ContextId;$

2095.0 $MultTypeBind :: pat : Pattern^{*}$
.1 $tp : Type$
.2 $cid : \text{-} CI`ContextId;$

.1.14 Literals

2096.0 $Literal = BoolLit \mid NilLit \mid NumLit \mid RealLit \mid CharLit \mid TextLit \mid QuoteLit;$

2097.0 $BoolLit :: val : \mathbb{B}$
.1 $cid :\text{-} CI`ContextId;$

2098.0 $NilLit :: cid :\text{-} CI`ContextId;$

2099.0 $RealLit :: val : \mathbb{R}$
.1 $cid :\text{-} CI`ContextId;$

2100.0 $NumLit = RealLit$
.1 $\mathsf{inv}\ n \triangleq \mathsf{is}\text{-}\mathbb{Z}\,(n.val);$

2101.0 $CharLit :: val : \mathsf{char}$
.1 $cid :\text{-} CI`ContextId;$

2102.0 $TextLit :: val : \mathsf{char}^*$
.1 $cid :\text{-} CI`ContextId;$

2103.0 $QuoteLit :: val : \mathsf{char}^*$
.1 $cid :\text{-} CI`ContextId;$

.1.15 Structure combining the AST and the ContextInfo

AstCI is a structure that is not directly part of the AST. It combines a *Document* or *Expr*, a *CI‘TokenContextInfo* and a *CI‘ FileIdMap* into a compound structure that is used by all the test environments.

2104.0 $AstCI :: ast : Document \mid Expr$
.1 $tllci : CI`TokenContextInfo^*$
.2 $fidm : CI`FileIdMap;$

.1.16 Debugger Constructs

LastRes is not directly part of the AST but is so tightly related to it that is has been defined as an AST node anyway. It is the node corresponding to the $$ command in the debugger.

2105.0 $LastRes :: \ ;$

2106.0 $\mathit{ClassStructure} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{inheritance} : \mathit{Inherit}^*$
.2 $\mathit{instance\text{-}vars} : \mathit{InstanceVarDef}^*$
.3 $\mathit{inst\text{-}init} : \mathit{InstanceInit}^*$
.4 $\mathit{inst\text{-}inv} : \mathit{InstanceInv}^*$
.5 $\mathit{methods} : \mathit{MethodDef}^*$
.6 $\mathit{tps} : \mathit{TypeDef}^*$
.7 $\mathit{fcts} : \mathit{FnDef}^*$
.8 $\mathit{vals} : \mathit{ValueDef}^*$
.9 $\mathit{sync} : \mathit{Sync}$
.10 $\mathit{thread} : \mathit{Thread}$
.11 $\mathit{timevars} : \mathit{TimeVar}^*$
.12 $\mathit{version} : \mathbb{N}$
.13 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId}$

.14 inv $\mathit{clstrct} \triangleq$ len $\mathit{clstrct.nm.ids} = 1$;

2107.0 $\mathit{Inherit} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{labels} : \mathit{LabelNumber}^*$
.2 $\mathit{methods} : [\mathit{InheritMethods}]$
.3 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId}$

.4 inv $\mathit{inh} \triangleq$ len $\mathit{inh.nm.ids} = 1$;

2108.0 $\mathit{LabelNumber} = \mathit{BoolLit} \mid \mathit{QuoteLit} \mid \mathit{NumLit} \mid \mathit{Range}$;

2109.0 $\mathit{Range} :: l : \mathit{NumLit}$
.1 $r : \mathit{NumLit}$
.2 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId}$;

2110.0 $\mathit{InheritMethods} :: \mathit{kind} : \mathrm{ALL} \mid \mathrm{ALLSUPER} \mid \mathit{Name}^*$
.1 $\mathit{cid} : \mathit{CI}`\mathit{ContextId}$;

2111.0 $\mathit{InstanceInit} :: \mathit{nms} : \mathit{Name}^* \mid \mathrm{OBJECTSTATE}$
.1 $\mathit{exp} : \mathit{Stmt}$
.2 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

2112.0 $\mathit{MethodDef} = \mathit{PrelimMethodDef} \mid \mathit{FullMethodDef}$;

2113.0 $\mathit{PrelimMethodDef} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{params} : \mathit{NameTypePair}^*$
.2 $\mathit{valtp} : [\mathit{OptNameType}]$
.3 $\mathit{body} : \mathrm{NOTYETSPEC} \mid \mathrm{SUBRESP}$
.4 $\mathit{cid} :\text{-} \mathit{CI}`\mathit{ContextId}$

.5 inv $n \triangleq$ len $n.\mathit{nm.ids} = 1 \wedge$ true;

2114.0 $\mathit{FullMethodDef} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{params} : \mathit{NameTypePair}^*$
.2 $\mathit{valtp} : [\mathit{OptNameType}]$
.3 $\mathit{mtpre} : [\mathit{DecoratedExpr}]$
.4 $\mathit{timep} : [\mathit{DecoratedExpr}]$
.5 $\mathit{mtbody} : \mathit{Stmt}$
.6 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$

.7 inv $n \triangleq$ len $n.\mathit{nm}.\mathit{ids} = 1 \land$ true;

2115.0 $\mathit{OptNameType} :: \mathit{nm} : [\mathit{Name}]$
.1 $\mathit{tp} : \mathit{Type}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$

.3 inv $\mathit{ont} \triangleq$ len $\mathit{ont}.\mathit{nm}.\mathit{ids} = 1$;

2116.0 $\mathit{NameTypePair} :: \mathit{pats} : \mathit{Name}^*$
.1 $\mathit{tp} : \mathit{Type}$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$

.3 inv $\mathit{ntp} \triangleq \forall\, p \in$ elems $\mathit{ntp}.\mathit{pats} \cdot$
.4 len $p.\mathit{ids} = 1$;

2117.0 $\mathit{DecoratedExpr} :: \mathit{expr} : \mathit{Expr}$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$

.2 inv $n \triangleq$ true;

2118.0 $\mathit{TimeVar} :: \mathit{vars} : \mathit{TimeVarDecl}^*$
.1 $\mathit{ass} : \mathit{Assumption}^*$
.2 $\mathit{effs} : \mathit{Effect}^*$
.3 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$;

2119.0 $\mathit{ObjRefType} :: \mathit{nm} : \mathit{Name}$
.1 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$

.2 inv $\mathit{objreftype} \triangleq$
.3 len $\mathit{objreftype}.\mathit{nm}.\mathit{ids} = 1$;

2120.0 $\mathit{InvokeStmt} :: \mathit{inst} : [\mathit{Name}] \mid \mathit{NewStmt}$
.1 $\mathit{mis} : \mathit{MethodInvoke}^+$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

2121.0 $\mathit{MethodInvoke} :: \mathit{mthd} : \mathit{Name}$
.1 $\mathit{arg} : \mathit{Expr}^*$
.2 $\mathit{cid} :\text{-}\, \mathit{CI}`\mathit{ContextId}$

.3 inv $n \triangleq$ true;

2122.0 $NewStmt :: inst : Name$
.1 $cid : \text{-} CI`ContextId$
.2 inv $ns \triangleq$ len $ns.inst.ids = 1;$

2123.0 $TopologyStmt :: stmt : SpecificationStmt$
.1 $cid : \text{-} CI`ContextId;$

2124.0 $Sync = PermissionStmt^*;$

2125.0 $PermissionStmt :: spec : Name$
.1 $guard : Expr$
.2 $cid : \text{-} CI`ContextId;$

2126.0 $Thread :: \ (PerObl \mid Stmt)^*$
.1 $cid : \text{-} CI`ContextId;$

2127.0 $FieldOrObjSelectExpr :: objrec : Expr$
.1 $select : Expr$
.2 $cid : \text{-} CI`ContextId$

end AS

Appendix A

References

[Bruin93] Cornelis de Bruin. *Towards an Interpreter for Full VDM-SL: Extending the IFAD VDM-SL Toolbox, including Modules, Higher Order Functions and Exception Handling.* Master's thesis, Delft University of Technology, June 1993. 107 pages.

[BSIVDM92] *VDM Specification Language – Proto-Standard.* Technical Report, British Standards Institution, August 1992. 403 pages. BSI IST/5/19 N-231.

[Lassen&91] Poul Bøgh Lassen and Peter Gorm Larsen. An Abstract State Machine Semantics for the IPTES Meta-IV Subset. October 1991.
IPTES Doc.id : IPTES-IFAD-12-V2.0.

[Lassen92] Poul Bøgh Lassen. *Differences between IPTES VDM-SL and BSI/VDM-SL.* Technical Report, IFAD, March 1992.
IPTES Doc.id. : IPTES-IFAD-137-V1.0.

Index