Test Automation for Overture

February 9, 2009

Peter Gorm Larsen, Kenneth Lausdahl, Nuno Rodrigues, Carlos Vilhena and Adriana Santos
IHA, Minho University and Danske Bank

Contents

1 Introduction

This document contains the VDM++ model of the test autoimation feature for Overture enabling test case generation with combinational testing principles. This VDM++ is then automatically translated to Java using the Java code generator from VDMTools. The generated java code is then integrated into the Overture environment developed on top of the Eclipse platform. This feature is invoked from the Overture GUI from the Eclipse SDE.

The VDM++ model is structured such that it takes an OmlSpecifications AST and a set of the class names that test automation shall be done for. Global information is first extracted using the `ExpandSpec` operation (from the `Global` class) that extract inheritance information and global definitions that can be used in the traces that test cases shall be generated from. Ater this extraction the `ExpandSpecTraces` operation from the `Expanded` class. This yields a mapping from name of test case directory to a set of test cases (each test case being represented as a sequence of `OmlExpression` AST's). Each of the test cases then needs to be converted to their concrete syntax using the visitors from the `Oml2VppVisitor` class producing a string (in `Oml2VppVisitor.result`) that can be used as an argument to the VDMTools interpreter. This is done for each expression in each test case as a sequence of calls until a run-time error is returned from the interpreter or all expressions in the test case have been executed. All test cases consisting of a sequence of expressions to be executed after each other in a sequence can be displayed in the GUI along with the generated results from VDMTools' interpreter. In case a test case gives a run-time-error there is no reason to exetute other test cases that have the same prefix. These are filtered away using the `Filtering` class. The test cases that are filtered away are thus NOT executed by the VDMTools' interpreter. At the end of the execution statistics as well as logs from the entire execution of the whole test suite can be produced using the `ppTestCases` from the `Filtering` class.

This VDM++ model is structured into a number of classes and their role can be explained as:

CTesting: Currently the main class but this can probably be omitted entirely.

DEF: This maintains global information extracted from definitions in different classes that act as the context of the different traces. This information is about globally defined values and inheritance between classes.

Expanded: This is the class where the core of the generation of test cases is placed. This includes functionality for expanding all kinds of trace constructs and functionality for combining different kinds of collections.

Eval: This is the class that contains functionality for evaluation of sub expressions into semantic values.

SEM: This class provides an internal representation of semantic values that are returned from the evaluation of expressions. In addition this class also have functionality for moving semantic values into `OmlExpression` ASTs.

Filtering: This class is responsible for filtering the test cases that have the same prefix as other test cases that have returned a run-time error away, such that they don't need to executed.

Toolbox: This class is responsible for interfacing to VDMTools' interpreter using the CORBA interface. A part of this will only be implemented at the Java level.

Oml2VppVisitor: This class enables the conversion from AST's fom OML to the conceret syntax of VDM++ as a string of characters that can be saved in argument files and shown to the user.

StdLib: Standard library used in an Overture context.

2 Global Considerations

class *DEF*
instance variables

$valm : Identifier \xrightarrow{m} ValueMap := \{\mapsto\};$
$inherit : Identifier \xrightarrow{m} Identifier\text{-set} := \{\mapsto\};$
$recdefs : Name \xrightarrow{m} Identifier \xrightarrow{m} \mathbb{N}_1 := \{\mapsto\};$

types

public $ValueMap = Identifier \xrightarrow{m} IOmlExpression;$
public $Identifier = \text{char}^*;$

public

$Name :: clnm : [Identifier]$
$\quad tnm : Identifier$

operations

public

$DEF : IOmlSpecifications \xrightarrow{o} DEF$
$DEF\,(spec) \triangleq$
$\quad ExpandSpec(spec)\,;$

public

$ExpandSpec : IOmlSpecifications \xrightarrow{o} ()$
$ExpandSpec\,(spec) \triangleq$
$\quad$ for cl in $spec.getClassList\,()$
$\quad$ do let $id = cl.getIdentifier\,(),$
$\qquad super =$ if $cl.hasInheritanceClause\,()$
$\qquad\qquad$ then $cl.getInheritanceClause\,().getIdentifierList\,()$
$\qquad\qquad$ else $[],$
$\qquad body = cl.getClassBody\,()$ in
$\quad (\quad valm(id) := ExpandValueMap\,(body);$
$\qquad recdefs := recdefs \dagger ExpandRecTypeDefs\,(id, body);$
$\qquad inherit(id) :=$ elems $super$
$\quad)\,;$
$ExpandValueMap : IOmlDefinitionBlock^* \xrightarrow{o} ValueMap$
$ExpandValueMap\,(body\text{-}l) \triangleq$
$\quad (\quad$ dcl $v\text{-}m : ValueMap := \{\mapsto\};$
$\qquad$ for $body$ in $body\text{-}l$
$\qquad$ do if isofclass $(OmlValueDefinitions, body)$
$\qquad\quad$ then $v\text{-}m := v\text{-}m \dagger ExpandValueDef\,(body);$
$\qquad$ return $v\text{-}m$
$\quad);$
$ExpandValueDef : IOmlValueDefinitions \xrightarrow{o} ValueMap$
$ExpandValueDef\,(body) \triangleq$
$\quad (\quad$ dcl $v\text{-}m : ValueMap := \{\mapsto\};$
$\qquad$ for $vdef$ in $body.getValueList\,()$
$\qquad$ do let $shape = vdef.getShape\,(),$
$\qquad\qquad pat = shape.getPattern\,(),$
$\qquad\qquad expr = shape.getExpression\,()$ in
$\qquad\quad$ if isofclass $(OmlPatternIdentifier, pat)$
$\qquad\quad$ then $v\text{-}m := v\text{-}m \dagger MatchPatId2Expr\,(pat, expr)$
$\qquad\quad$ else error;
$\qquad$ return $v\text{-}m$
$\quad);$
$ExpandRecTypeDefs : Identifier \times IOmlDefinitionBlock^* \xrightarrow{o}$
$\qquad\qquad Name \xrightarrow{m} Identifier \xrightarrow{m} \mathbb{N}_1$
$ExpandRecTypeDefs\,(clnm, body\text{-}l) \triangleq$
$\quad (\quad$ dcl $r\text{-}m : Name \xrightarrow{m} Identifier \xrightarrow{m} \mathbb{N}_1 := \{\mapsto\};$
$\qquad$ for $body$ in $body\text{-}l$

```
            do if isofclass (OmlTypeDefinitions, body)
               then r-m := r-m † ExpandTypeDefs (clnm, body);
            return r-m
        );
    ExpandTypeDefs : Identifier × IOmlTypeDefinitions →o
                         Name →m Identifier →m ℕ1
    ExpandTypeDefs (clnm, body) ≜
        (   dcl r-m : Name →m Identifier →m ℕ1 := {↦};
            for tdef in body.getTypeList ()
            do let shape = tdef.getShape () in
               if isofclass (OmlComplexType, shape)
               then r-m := r-m † ExpandComplexTypeDef (clnm, shape);
            return r-m
        );
    ExpandComplexTypeDef : Identifier × IOmlComplexType →o
                               Name →m Identifier →m ℕ1
    ExpandComplexTypeDef (clnm, shape) ≜
        let id = shape.getIdentifier (),
            f-l = shape.getFieldList () in
        return {mk-Name (clnm, id) ↦ {f-l (i).getIdentifier () ↦ i |
                    i ∈ inds f-l}};
public
    LookUp : Identifier × Identifier →o IOmlExpression
    LookUp (clnm, defnm) ≜
        if clnm ∈ dom valm
        then let vm = valm (clnm) in
            if defnm ∈ dom vm
            then return vm (defnm)
            elseif clnm ∈ dom inherit
            then let supers = inherit (clnm) in
                (   for all sup ∈ supers
                    do if sup ∈ dom valm ∧ defnm ∈ dom (valm (sup))
                       then return valm (sup) (defnm) ;
                    error
                )
            else error
        else error;
public
    LookUpRecSel : SEM`REC × Identifier × IOmlFieldSelect →o SEM`VAL
    LookUpRecSel (recval, selid, expr) ≜
        let tag = recval.tag in
        if tag ∈ dom recdefs
        then let sel-m = recdefs (tag) in
            if selid ∈ dom sel-m
            then let num = sel-m (selid) in
                return (recval.v) (num)
            else (   RTERR`ReportError(expr, RTERR`RECORD-FIELD-ID-UNKNOWN);
                return mk-SEM`NUM (1)
            )
        else (   RTERR`ReportError(expr, RTERR`RECORD-FIELD-ID-UNKNOWN) ;
            return mk-SEM`NUM (1)
        )
functions
```

$MatchPatId2Expr : IOmlPatternIdentifier \times IOmlExpression \to ValueMap$
$MatchPatId2Expr\,(patid, expr) \triangleq$
let $id = patid.getIdentifier\,()$ in
$\{id \mapsto expr\}$
end *DEF*

Test Suite : vdm.tc
Class : DEF

Name	#Calls	Coverage
DEF‘DEF	31	$\surd$
DEF‘LookUp	9	94%
DEF‘ExpandSpec	31	$\surd$
DEF‘LookUpRecSel	0	0%
DEF‘ExpandTypeDefs	7	$\surd$
DEF‘ExpandValueDef	5	96%
DEF‘ExpandValueMap	74	$\surd$
DEF‘MatchPatId2Expr	6	$\surd$
DEF‘ExpandRecTypeDefs	74	$\surd$
DEF‘ExpandComplexTypeDef	6	$\surd$
Total Coverage		**81%**

3 Expansion of Traces

The `Expanded` class is responsible for expanding trace definitions to collections of sequences of expressions. This class contains functionality for expansion of traces and uses evaluation of expressions that occur inside such traces in the `Eval` class. Finally the class contains functionality for combining collections of different kinds of structures.

class *Expanded*
instance variables
private $zeroOrMoreMax : \mathbb{N} := 3$;
private $oneOrMoreMax : \mathbb{N} := 3$;
public $seqOfNames : (\text{char}^*)^* := []$;
static $curcl : Global`Identifier := \text{""}$;
$cxt\text{-}s\text{-}stack : Eval`Context\text{-set}^* := []$;

3.1 Expansion of Trace ASTs

Traces in VDM++ can be expressed in using different kinds of constructs for choices, bindings and repetitions. Common to all of these are that the expansion functions/operations yield a mapping from the name of a trace (the name is a pair of the name of the class and the name of the trace) to a set of test cases. Each test case can be seen as a sequnce of expressions that needs to be executed after each other to conduct the required test. The expressions are represented in the AST form and then later these are transformed to a sequence of chars such that these can be executed by an interpreter.

operations
public
$ExpandSpecTraces : IOmlSpecifications \times Global`Identifier\text{-set} \xrightarrow{o}$
$Global`Name \xrightarrow{m} IOmlExpression^*\text{-set}$
$ExpandSpecTraces\,(spec, cl\text{-}s) \triangleq$
(dcl $res\text{-}m : Global`Name \xrightarrow{m} IOmlExpression^*\text{-set} := \{\mapsto\}$;
for cl in $spec.getClassList\,()$

```
            do let id = cl.getIdentifier () in
               if id ∈ cl-s
               then res-m := res-m † ExpandClassTraces (cl);
            return res-m
         );
public
      ExpandClassTraces : IOmlClass ->o
                          Global`Name ->m IOmlExpression*-set
      ExpandClassTraces (cl) ≜
         (  dcl res-m : Global`Name ->m IOmlExpression*-set := {|->};
            let clid = cl.getIdentifier (),
                body = cl.getClassBody () in
            (  curcl := clid;
               for def-l in body
               do if isofclass (IOmlTraceDefinitions, def-l)
                  then res-m := res-m † ExpandTraceDefs (clid, def-l)
            );
            return res-m
         );
public
      ExpandTraceDefs : Global`Identifier × IOmlTraceDefinitions ->o
                        Global`Name ->m IOmlExpression*-set
      ExpandTraceDefs (clid, def-l) ≜
         (  dcl res-m : Global`Name ->m IOmlExpression*-set := {|->};
            for ntrace : OmlNamedTrace in def-l.getTraces ()
            do let name = ntrace.getName (),
                   defs = ntrace.getDefs () in
               res-m := res-m † {mk-Global`Name (clid, name) |->
                                 ExpandTraceDef (defs, {{|->}})};
            return res-m
         );
      ExpandTraceDef : IOmlTraceDefinition × Eval`Context-set ->o
                       IOmlExpression*-set
      ExpandTraceDef (tdef, ctx-s) ≜
         if isofclass (IOmlTraceDefinitionItem, tdef)
         then ExpandTraceDefItem(tdef, ctx-s)
         elseif isofclass (IOmlTraceSequenceDefinition, tdef)
         then ExpandTraceSeqDef(tdef, ctx-s)
         else ExpandTraceChoiceDef(tdef, ctx-s) ;
```

A `TraceDefinitionItem` is composed of a collection of bindings, a trace core definition and a trace repeat pattern (enabling repetition of the trace core expressions).

In the `overture.ast` file it is defined as:

```
TraceDefinitionItem ::
  bind    : seq of TraceBinding
  test    : TraceCoreDefinition
  regexpr : [TraceRepeatPattern];
```

At the concrete syntax level that corresponds to:

```
trace definition = trace core definition
                 | trace bindings, trace core definition
                 | trace core definition, trace repeat pattern
```

| trace bindings, trace core definition, trace repeat pattern ;

trace core definition = trace apply expression
| trace bracketed expression ;

trace apply expression = identifier, '.', Identifier, '(', expression list, ')' ;

trace repeat pattern = '*'
| '+'
| '?'
| '{', numeric literal, '}'
| '{', numeric literal, ',' numeric literal, '}' ;

trace bracketed expression = '(', trace definition list, ')' ;

trace bindings = { trace binding } ;

trace binding = 'let', list of local definitions, 'in'
| 'let', bind, 'in'
| 'let', bind, 'be', 'st', expression, 'in' ;

$ExpandTraceDefItem : IOmlTraceDefinitionItem \times Eval`Context\text{-set} \xrightarrow{o} IOmlExpression^*\text{-set}$
$ExpandTraceDefItem\,(tdef, ctx\text{-}s) \triangleq$
let $bind = tdef.getBind\,()$,
$trace = tdef.getTest\,()$,
$regexpr =$ if $tdef.hasRegexpr\,()$
then $tdef.getRegexpr\,()$
else nil ,
$new\text{-}ctx\text{-}s = ExpandTraceBinds\,(bind, ctx\text{-}s)$,
$expr\text{-}l\text{-}s = ExpandTraceCoreDef\,(trace, new\text{-}ctx\text{-}s)$ in
return if $regexpr =$ nil
then $expr\text{-}l\text{-}s$
else $ExpandTraceRepeatPat\,(regexpr, expr\text{-}l\text{-}s)$;
$ExpandTraceSeqDef : IOmlTraceSequenceDefinition \times Eval`Context\text{-set} \xrightarrow{o} IOmlExpression^*\text{-set}$
$ExpandTraceSeqDef\,(tdef, ctx\text{-}s) \triangleq$
(dcl $expr\text{-}l\text{-}s : IOmlExpression^*\text{-set} := \{[]\}$;
for td in $tdef.getDefs\,()$
do let $e\text{-}l\text{-}s = ExpandTraceDef\,(td, ctx\text{-}s)$ in
$expr\text{-}l\text{-}s := CombineTraces\,(expr\text{-}l\text{-}s, e\text{-}l\text{-}s)$;
return $expr\text{-}l\text{-}s$
);
$ExpandTraceChoiceDef : IOmlTraceChoiceDefinition \times Eval`Context\text{-set} \xrightarrow{o} IOmlExpression^*\text{-set}$
$ExpandTraceChoiceDef\,(tdef, ctx\text{-}s) \triangleq$
(dcl $expr\text{-}l\text{-}s : IOmlExpression^*\text{-set} := \{\}$;
for td in $tdef.getDefs\,()$
do let $e\text{-}l\text{-}s = ExpandTraceDef\,(td, ctx\text{-}s)$ in
$expr\text{-}l\text{-}s := expr\text{-}l\text{-}s \cup e\text{-}l\text{-}s$;
return $expr\text{-}l\text{-}s$
);
$ExpandTraceBinds : IOmlTraceBinding^* \times Eval`Context\text{-set} \xrightarrow{o} Eval`Context\text{-set}$
$ExpandTraceBinds\,(bind\text{-}l, cxt\text{-}s) \triangleq$
(dcl $c\text{-}s : Eval`Context\text{-set} := cxt\text{-}s$;

```
        for bind in bind-l
        do let c-s2 = if isofclass (IOmlTraceLetBinding, bind)
                      then ExtractLetBinding (bind, c-s)
                      else ExtractLetBeBinding (bind, c-s) in
           c-s := Eval`CombineContexts (c-s, c-s2);
        return c-s
    )
functions
    ExpandTraceCoreDef : IOmlTraceCoreDefinition × Eval`Context-set →
                         IOmlExpression*-set
    ExpandTraceCoreDef (tdef, cxt-s) ≜
      if isofclass (IOmlTraceMethodApply, tdef)
      then ExpandTraceMethodApply (tdef, cxt-s)
      else ExpandBracketedTraceDef (tdef, cxt-s)
```

Bracketed trace definitions actually give raise to an extra level of complexity. The reason for this is that one needs to be able to distingush cases such as:

```
PushBeforePop : let x in set {1,3}
                in
                  let y in set {3,8}
                  in
                    (s.Push(x); s.Push(y))
```

from

```
PushBeforePop : let x in set {1,3}
                in
                  let y in set {3,8}
                  in
                    s.Push(x);
                let x in set {1,3}
                in
                  let y in set {3,8}
                  in
                    s.Push(y)
```

In the first case it is essential when combining contexts that the variables `x` and `y` are mapped to the same value for each subexpression inside the trace definition enclosed inside the brackets. In the second case where no brackets are used it is essential to include all possible combinations in the resulting collection of test cases.

```
operations
    ExpandBracketedTraceDef : IOmlTraceBracketedDefinition × Eval`Context-set ^o→
                              IOmlExpression*-set
    ExpandBracketedTraceDef (tdef, cxt-s) ≜
      (   PushCxt(cxt-s);
          let e-l-s = ExpandTraceDef (tdef.getDefinition (), {{↦}}),
              e-l-s2 = {AddContextToExprList (e-l, hd cxt-s-stack) |
                        e-l ∈ e-l-s} in
          (   PopCxt();
              return ⋃ e-l-s2
          )
      )
```

functions

$ExpandTraceMethodApply : IOmlTraceMethodApply \times Eval`Context\text{-set} \rightarrow IOmlExpression^*\text{-set}$

$ExpandTraceMethodApply\,(tdef, cxt\text{-}s) \triangleq$
  let $var = tdef.getVariableName\,()$,
    $met\text{-}nm = tdef.getMethodName\,()$,
    $args = tdef.getArgs\,()$,
    $fieldsel =$ new $OmlFieldSelect\,($new $OmlName\,($nil , $var)$,
                                  new $OmlName\,($nil , $met\text{-}nm))$,
    $expr =$ new $OmlApplyExpression\,(fieldsel, args)$ in
  $AddContextToExpr\,(expr, cxt\text{-}s)$

operations

$ExpandTraceRepeatPat : IOmlTraceRepeatPattern \times IOmlExpression^*\text{-set} \xrightarrow{o} IOmlExpression^*\text{-set}$

$ExpandTraceRepeatPat\,(regexpr, expr\text{-}l\text{-}s) \triangleq$
  cases true:
    (isofclass $(IOmlTraceZeroOrMore, regexpr)) \rightarrow$
      let $rep = RepeatCombine\,(expr\text{-}l\text{-}s, 1, oneOrMoreMax)$ in
      return $\{[]\} \cup rep$,
    (isofclass $(IOmlTraceOneOrMore, regexpr)) \rightarrow$
      return $RepeatCombine\,(expr\text{-}l\text{-}s, 1, oneOrMoreMax)$,
    (isofclass $(IOmlTraceZeroOrOne, regexpr)) \rightarrow$
      return $expr\text{-}l\text{-}s \cup \{[]\}$,
    (isofclass $(IOmlTraceRange, regexpr)) \rightarrow$
      return $ExpandTraceRange\,(regexpr, expr\text{-}l\text{-}s)$
  end;

$ExpandTraceRange : IOmlTraceRange \times IOmlExpression^*\text{-set} \xrightarrow{o} IOmlExpression^*\text{-set}$

$ExpandTraceRange\,(regexpr, expr\text{-}l\text{-}s) \triangleq$
  let $low = regexpr.getLower\,().getVal\,()$,
    $high =$ if $regexpr.hasUpper\,()$
        then $regexpr.getUpper\,().getVal\,()$
        else low,
    $l =$ if $low = 0$
      then 1
      else low,
    $no =$ if $low = 0$
       then $\{[]\}$
       else $\{\}$,
    $rep = RepeatCombine\,(expr\text{-}l\text{-}s, l, high)$ in
  return $no \cup rep$;

There are a number of functions/operations for combining different kinds of collections in different ways.

$RepeatCombine : (IOmlExpression^*\text{-set}) \times \mathbb{N}_1 \times \mathbb{N}_1 \xrightarrow{o} IOmlExpression^*\text{-set}$

$RepeatCombine\,(expr\text{-}l\text{-}s, low, high) \triangleq$
  (dcl $acc\text{-}e\text{-}l\text{-}s : IOmlExpression^*\text{-set} := \{\}$,
    $ith\text{-}e\text{-}l\text{-}s : IOmlExpression^*\text{-set} := \{[]\}$,
    $i : \mathbb{N}_1 := 1$;
   while $i \leq high$

```
            do let oldith = ith-e-l-s in
               (   ith-e-l-s := {e-l1 ⌢ e-l2 |
                                    e-l1 ∈ oldith, e-l2 ∈ expr-l-s};
                   if i ≥ low
                   then acc-e-l-s := acc-e-l-s ∪ ith-e-l-s;
                   i := i + 1
               );
            return acc-e-l-s
         )
functions
      AddContextToExpr : IOmlExpression × Eval`Context-set →
                                 IOmlExpression*-set
      AddContextToExpr (expr, cxt-s) ≜
        if cxt-s = {}
        then {[expr]}
        else {let def-l = Context2ValShapeL (cxt) in
              if def-l = []
              then [expr]
              else [new OmlLetExpression (def-l, expr)] |
                   cxt ∈ cxt-s};
      AddContextToExprList : IOmlExpression* × Eval`Context-set →
                                    IOmlExpression*-set
      AddContextToExprList (e-l, cxt-s) ≜
        if cxt-s = {}
        then {e-l}
        else {let def-l = Context2ValShapeL (cxt) in
              if def-l = []
              then e-l
              else [new OmlLetExpression (def-l, e-l (i)) |
                        i ∈ inds e-l] |
                   cxt ∈ cxt-s};
      Context2ValShapeL : Eval`Context → IOmlValueShape*
      Context2ValShapeL (cxt) ≜
        if cxt = {↦}
        then []
        else let id ∈ dom cxt in
             let pat = new OmlPatternIdentifier (id),
                 val = SEM`VAL2IOmlExpr (cxt (id)),
                 valshape = new OmlValueShape (pat, nil , val),
                 rest = Context2ValShapeL ({id} ⩤ cxt) in
             [valshape] ⌢ rest;
      SmallerContext : Eval`Context → ℕ
      SmallerContext (cxt) ≜
        card dom cxt;
      CombineTraces : (IOmlExpression*-set) × (IOmlExpression*-set) →
                           (IOmlExpression*-set)
      CombineTraces (e-l-s1, e-l-s2) ≜
        {e-l1 ⌢ e-l2 |
             e-l1 ∈ e-l-s1, e-l2 ∈ e-l-s2}
operations
```

```
        ExtractLetBinding : IOmlTraceLetBinding × Eval`Context-set ^o→
                                Eval`Context-set
        ExtractLetBinding (bind, cxt-s) ≜
          let def-l = bind.getDefinitionList () in
          (   dcl c-s : Eval`Context-set := cxt-s;
              for valshape in def-l
              do let pat = valshape.getPattern (),
                     expr = valshape.getExpression (),
                     val-s = {Eval`evaluateExpression (expr, cxt) | cxt ∈ c-s},
                     newc-s = {Eval`PatternMatch (pat, val) | val ∈ val-s} in
                 c-s := Eval`CombineContexts (c-s, newc-s);
              return c-s
          )
functions
        ExtractLetBeBinding : IOmlTraceLetBeBinding × Eval`Context-set →
                                Eval`Context-set
        ExtractLetBeBinding (lbbind, cxt-s) ≜
          let bind = lbbind.getBind (),
              best = if lbbind.hasBest ()
                     then lbbind.getBest ()
                     else nil  in
          if isofclass (IOmlSetBind, bind)
          then let bestex = if best = nil
                            then new OmlSymbolicLiteralExpression (new OmlBooleanLiteral (true))
                            else best in
               ExtractLetBeSetBinding (bind, bestex, cxt-s)
          else undefined;
        ExtractLetBeSetBinding : IOmlSetBind × IOmlExpression × Eval`Context-set →
                                 Eval`Context-set
        ExtractLetBeSetBinding (bind, best, cxt-s) ≜
          let p-l = bind.getPattern (),
              expr = bind.getExpression () in
          ⋃{let val = Eval`evaluateExpression (expr, cxt),
               c-l-s = if is-SEM`SET (val)
                       then {[Eval`PatternMatch (p-l (i), v) |
                                  i ∈ inds p-l] |
                                 v ∈ val.v}
                       else undefined in
            {Eval`MergeContextList (c-l) |
                 c-l ∈ c-l-s ·
                     let c = Eval`MergeContextList ([cxt] ⌢ c-l) in
                     Eval`evaluateExpression (best, c) = mk-SEM`BOOL (true)} |
              cxt ∈ cxt-s}
operations
public
        expandRegexpr : IOmlTraceDefinitionItem ^o→
                            (IOmlTraceDefinitionItem*)*
        expandRegexpr (i) ≜
          return expandRegexprChoose (i);
public
```

```
        expandRegexprChoose : IOmlTraceDefinitionItem ^o→
                                  (IOmlTraceDefinitionItem*)*
        expandRegexprChoose (i) △
          let r = i.getRegexpr () in
          if i.hasRegexpr ()
          then return expandSymbol (i, r)
          else return [[i]] ;
public
        expandN2M : ℕ × ℕ × IOmlTraceDefinitionItem × IOmlTraceDefinitionItem* ^o→
                        (IOmlTraceDefinitionItem*)*
        expandN2M (n, m, s, o) △
          if n ≠ m
          then return [o] ↷ expandN2M (n, m − 1, s, o ↷ [s])
          else if n = 0
               then return [[new OmlTraceDefinitionItem ([],
                                                         new OmlTraceMethodApply ([], [], []), nil )]]
               else return [o] ;
public
        expandSymbol : IOmlTraceDefinitionItem × IOmlTraceRepeatPattern ^o→
                           (IOmlTraceDefinitionItem*)*
        expandSymbol (s, r) △
          cases true:
            (isofclass (IOmlTraceZeroOrMore, r)) →
                return expandSymbolZeroOrMore (s, r),
            (isofclass (IOmlTraceOneOrMore, r)) →
                return expandSymbolOneOrMore (s, r),
            (isofclass (IOmlTraceZeroOrOne, r)) →
                return expandSymbolZeroOrOne (s, r),
            (isofclass (IOmlTraceRange, r)) →
                return expandSymbolRange (s, r),
            others → return []
          end;
public
        expandSymbolZeroOrMore : IOmlTraceDefinitionItem × IOmlTraceZeroOrMore ^o→
                                     (IOmlTraceDefinitionItem*)*
        expandSymbolZeroOrMore (s, -) △
          return expandN2M (0, zeroOrMoreMax, s, [s]);
public
        expandSymbolOneOrMore : IOmlTraceDefinitionItem × IOmlTraceOneOrMore ^o→
                                    (IOmlTraceDefinitionItem*)*
        expandSymbolOneOrMore (s, -) △
          return expandN2M (1, oneOrMoreMax, s, [s]);
public
        expandSymbolZeroOrOne : IOmlTraceDefinitionItem × IOmlTraceZeroOrOne ^o→
                                    (IOmlTraceDefinitionItem*)*
        expandSymbolZeroOrOne (s, -) △
          return expandN2M (0, 1, s, [s]);
public
        expandSymbolRange : IOmlTraceDefinitionItem × IOmlTraceRange ^o→
                                (IOmlTraceDefinitionItem*)*
        expandSymbolRange (s, t) △
          let min = t.getLower ().getVal (),
```

```
        max = getVal (min, t.getUpper ()) in
    return expandN2M (min, max, s, [s]);
public
    getVal : ℕ × [IOmlNumericLiteral] →o
             ℕ
    getVal (min, n) △
      if n = nil
      then return min
      else return n.getVal () ;
public
    getLetBeInfo : IOmlTraceLetBeBinding →o
                   char* →m IOmlExpression
    getLetBeInfo (b) △
      return {extractBindingVariable (b) ↦
              extractBindingExpression (b)}
    pre  isOfTypeSB (b) ;
public
    isOfTypeSB : IOmlTraceLetBeBinding →o
                 ℬ
    isOfTypeSB (b) △
      let bind = b.getBind () in
      if isofclass (IOmlSetBind, bind)
      then return isOfTypePattern (bind)
      else return false ;
public
    isOfTypePattern : IOmlSetBind →o
                      ℬ
    isOfTypePattern (s) △
      let p = s.getPattern (),
          v = p (1) in
      return isofclass (IOmlPatternIdentifier, v)
    pre  len (s.getPattern ()) = 1 ;
public
    extractBindingVariable : IOmlTraceLetBeBinding →o
                             char*
    extractBindingVariable (b) △
      let bind = b.getBind () in
      return getVariable (bind)
    pre  is_(b.getBind (), IOmlSetBind) ;
public
    getVariable : IOmlSetBind →o
                  char*
    getVariable (b) △
      let p = b.getPattern (),
          v = p (1) in
      return getVariableName (v)
    pre  len (b.getPattern ()) = 1 ∧
         isofclass (IOmlPatternIdentifier, b.getPattern () (1)) ;
public
    getVariableName : IOmlPatternIdentifier →o
                      char*
    getVariableName (pi) △
      return pi.getIdentifier ();
```

```
public
        extractBindingExpression : IOmlTraceLetBeBinding ->o
                                   IOmlExpression
        extractBindingExpression (b) ≜
          let bind = b.getBind () in
          return getExpression (bind)
        pre isofclass (IOmlSetBind, b.getBind ()) ;
public
        getExpression : IOmlSetBind ->o
                        IOmlExpression
        getExpression (b) ≜
          return b.getExpression ();
public
        getConstraints : IOmlTraceLetBeBinding ->o
                         [IOmlExpression]
        getConstraints (b) ≜
          return b.getBest ();
public
        getLetInfo : IOmlTraceLetBinding ->o
                     char* ->m IOmlExpression
        getLetInfo (b) ≜
          let def-list = b.getDefinitionList () in
          return {getPatternId (def-list (e).getPattern ()) ↦
                  def-list (e).getExpression () |
                    e ∈ inds def-list ·
                      isofclass (IOmlPatternIdentifier, def-list (e).getPattern ())};
public
        getPatternId : IOmlPatternIdentifier ->o
                       char*
        getPatternId (p) ≜
          return p.getIdentifier ();
```

3.2 Popping and Pushing the Context Stack

This is needed to be able to handle bracketed traces appropriately.

```
        PushCxt : Eval`Context-set ->o ()
        PushCxt (cxt-s) ≜
          cxt-s-stack := [cxt-s] ↷ cxt-s-stack;
        PopCxt : () ->o ()
        PopCxt () ≜
          cxt-s-stack := tl cxt-s-stack
        pre ¬ CxtStackIsEmpty () ;
        CxtStackIsEmpty : () ->o B
        CxtStackIsEmpty () ≜
          return cxt-s-stack = [];
public static
        GetCurClass : () ->o Global`Identifier
        GetCurClass () ≜
          return curcl
end Expanded
```

Test Suite : vdm.tc

Class : Expanded

Name	#Calls	Coverage
Expanded‘PopCxt	21	50%
Expanded‘getVal	0	0%
Expanded‘PushCxt	21	√
Expanded‘expandN2M	0	0%
Expanded‘getLetInfo	0	0%
Expanded‘isOfTypeSB	0	0%
Expanded‘GetCurClass	0	0%
Expanded‘getVariable	0	0%
Expanded‘expandSymbol	0	0%
Expanded‘getLetBeInfo	0	0%
Expanded‘getPatternId	0	0%
Expanded‘CombineTraces	77	√
Expanded‘RepeatCombine	38	√
Expanded‘expandRegexpr	0	0%
Expanded‘getExpression	0	0%
Expanded‘ExpandTraceDef	157	√
Expanded‘SmallerContext	0	0%
Expanded‘getConstraints	0	0%
Expanded‘CxtStackIsEmpty	0	0%
Expanded‘ExpandTraceDefs	28	√
Expanded‘getVariableName	0	0%
Expanded‘isOfTypePattern	0	0%
Expanded‘AddContextToExpr	98	86%
Expanded‘ExpandSpecTraces	28	√
Expanded‘ExpandTraceBinds	119	√
Expanded‘ExpandTraceRange	30	√
Expanded‘Context2ValShapeL	245	√
Expanded‘ExpandClassTraces	67	√
Expanded‘ExpandTraceSeqDef	31	√
Expanded‘ExtractLetBinding	8	√
Expanded‘expandSymbolRange	0	0%
Expanded‘ExpandTraceCoreDef	119	√
Expanded‘ExpandTraceDefItem	119	√
Expanded‘ExtractLetBeBinding	21	√
Expanded‘expandRegexprChoose	0	0%
Expanded‘AddContextToExprList	36	90%
Expanded‘ExpandTraceChoiceDef	7	√
Expanded‘ExpandTraceRepeatPat	40	√
Expanded‘expandSymbolOneOrMore	0	0%
Expanded‘expandSymbolZeroOrOne	0	0%
Expanded‘ExpandTraceMethodApply	98	√
Expanded‘ExtractLetBeSetBinding	21	√
Expanded‘expandSymbolZeroOrMore	0	0%
Expanded‘extractBindingVariable	0	0%
Expanded‘ExpandBracketedTraceDef	21	√
Expanded‘extractBindingExpression	0	0%
Total Coverage		**65%**

4 Evaluation of Expressions

Inside traces it is possible to use expressions and these expressions are used to identify numerous test cases. Thus these needs to be evaluated to determine the required looseness in the test cases. The evaluation operations/functions takes an AST for an expression and a context and yields a semantic value. The context is a mapping from identifier to a semantic value. Only a small subset of expressions are covered right now.

class $Eval$

types

public $Context = DEF`Identifier \xrightarrow{m} SEM`VAL$

instance variables

$specdefs : DEF;$
$curcl : DEF`Identifier;$

operations

public

$Eval : DEF`Identifier \times DEF \xrightarrow{o} Eval$
$Eval\,(clid, defs) \triangleq$
($curcl := clid;$
$specdefs := defs$
);

public

$evaluateExpression : IOmlExpression \times Context \xrightarrow{o} SEM`VAL$
$evaluateExpression\,(expr, cxt) \triangleq$
cases true:
(isofclass $(IOmlSymbolicLiteralExpression, expr)) \rightarrow$
return $getValueOfSymLit\,(expr)$,
(isofclass $(IOmlSequenceEnumeration, expr)) \rightarrow$
return $evaluateSeqEnumeration\,(expr, cxt)$,
(isofclass $(IOmlSetEnumeration, expr)) \rightarrow$
return $evaluateSetEnumeration\,(expr, cxt)$,
(isofclass $(IOmlMapEnumeration, expr)) \rightarrow$
return $evaluateMapEnumeration\,(expr, cxt)$,
(isofclass $(IOmlSetRangeExpression, expr)) \rightarrow$
return $evaluateSetRange\,(expr, cxt)$,
(isofclass $(IOmlName, expr)) \rightarrow$
return $evaluateName\,(expr, cxt)$,
(isofclass $(IOmlBinaryExpression, expr)) \rightarrow$
return $evaluateBinary\,(expr, cxt)$,
(isofclass $(IOmlUnaryExpression, expr)) \rightarrow$
return $evaluateUnary\,(expr, cxt)$,
(isofclass $(IOmlBracketedExpression, expr)) \rightarrow$
return $evaluateBracketedExpression\,(expr, cxt)$,
(isofclass $(IOmlSetComprehension, expr)) \rightarrow$
return $evaluateSetComprehension\,(expr, cxt)$,
(isofclass $(IOmlNewExpression, expr)) \rightarrow$
return $evaluateNewExpression\,(expr, cxt)$,
(isofclass $(IOmlIfExpression, expr)) \rightarrow$
return $evaluateIfExpression\,(expr, cxt)$,
(isofclass $(IOmlLetExpression, expr)) \rightarrow$
return $evaluateLetExpression\,(expr, cxt)$,
(isofclass $(IOmlFieldSelect, expr)) \rightarrow$
return $evaluateFieldSelect\,(expr, cxt)$,

```
    (isofclass (IOmlRecordConstructor, expr)) →
        return evaluateRecordConstructor (expr, cxt),
    (isofclass (IOmlTokenExpression, expr)) →
        return evaluateTokenExpression (expr, cxt),
    others → error
end;
```

4.1 Sequence Enumeration

Evaluation of a sequence enumeration expression is simply an evaluation of the element expressions and then placing all of them in a semantic sequence.

public

```
evaluateSeqEnumeration : IOmlSequenceEnumeration × Context →o
                         SEM`VAL
evaluateSeqEnumeration (expr, cxt) ≜
  let s = expr.getExpressionList () in
  return mk-SEM`SEQ ([evaluateExpression (s (i), cxt) | i ∈ inds s]);
```

4.2 Bracketed Expression

Basically the expression inside the brackets simply needs to be evaluated.

public

```
evaluateBracketedExpression : IOmlBracketedExpression × Context →o
                              SEM`VAL
evaluateBracketedExpression (expr, cxt) ≜
  let e = expr.getExpression () in
  evaluateExpression(e, cxt) ;
```

4.3 Set Enumeration

Evaluation of a set enumeration expression is simply an evaluation of the element expressions and then placing all of them in a semantic set.

public

```
evaluateSetEnumeration : IOmlSetEnumeration × Context →o
                         SEM`VAL
evaluateSetEnumeration (expr, cxt) ≜
  let s = expr.getExpressionList () in
  return mk-SEM`SET ({evaluateExpression (s (i), cxt) | i ∈ inds s});
```

4.4 Map Enumeration

Evaluation of a map enumeration expression is simply an evaluation of the element maplet expressions and then placing all of them in a semantic map.

public

```
evaluateMapEnumeration : IOmlMapEnumeration × Context →o
                         SEM`VAL
evaluateMapEnumeration (expr, cxt) ≜
  let s = expr.getMapletList () in
  return mk-SEM`MAP ({evaluateExpression (s (i).getDomExpression (), cxt) ↦
                      evaluateExpression (s (i).getRngExpression (), cxt) |
                        i ∈ inds s});
```

4.5 Set Range Expression

Evaluating a set range expression is done by evaluation of the lower and upper ranges. In case these are both numbers the resulting set is the integers between these numbers. Otherwise an error must be returned.

public

$$
\begin{array}{l}
evaluateSetRange : IOmlSetRangeExpression \times Context \xrightarrow{o} \\
\qquad SEM`VAL \\
evaluateSetRange\,(expr, cxt) \triangleq \\
\quad \textsf{let } l = evaluateExpression\,(expr.getLower\,(), cxt), \\
\qquad u = evaluateExpression\,(expr.getUpper\,(), cxt), \\
\qquad s : \mathbb{N}\textsf{-set} = \textsf{if is-}SEM`NUM\,(l) \wedge \textsf{is-}SEM`NUM\,(u) \\
\qquad\qquad \textsf{then } \{l.v, \ldots, u.v\} \\
\qquad\qquad \textsf{else } \{\} \textsf{ in} \\
\quad (\quad \textsf{if } \neg \textsf{is-}SEM`NUM\,(l) \\
\qquad \textsf{then } RTERR`ReportError(expr, RTERR`LOWER\text{-}BOUND\text{-}NOT\text{-}A\text{-}NUMBER); \\
\qquad \textsf{if } \neg \textsf{is-}SEM`NUM\,(u) \\
\qquad \textsf{then } RTERR`ReportError(expr, RTERR`UPPER\text{-}BOUND\text{-}NOT\text{-}A\text{-}NUMBER); \\
\qquad \textsf{return mk-}SEM`SET\,(\{\textsf{mk-}SEM`NUM\,(i) \mid i \in s\}) \\
\quad);
\end{array}
$$

4.6 Set Comprehension

Evaluation of set comprehension expressions is done by running over the possible bindings made by the syntactic binding and then in all the cases where the guard evaluates to true the element expression is evaluated and each of these are placed in the resulting set.

public

$$
\begin{array}{l}
evaluateSetComprehension : IOmlSetComprehension \times Context \xrightarrow{o} \\
\qquad SEM`VAL \\
evaluateSetComprehension\,(expr, cxt) \triangleq \\
\quad \textsf{let } elem = expr.getExpression\,(), \\
\qquad bind\text{-}l = expr.getBindList\,(), \\
\qquad guard = \textsf{if } expr.hasGuard\,() \\
\qquad\qquad \textsf{then } expr.getGuard\,() \\
\qquad\qquad \textsf{else nil}, \\
\qquad cxt\text{-}s = evalBindList\,(bind\text{-}l, cxt) \textsf{ in} \\
\quad \textsf{return mk-}SEM`SET \\
\qquad\qquad (\\
\qquad\qquad \{\textsf{let } cxt3 = cxt \dagger cxt2 \textsf{ in} \\
\qquad\qquad\ \textsf{if } guard = \textsf{nil} \ \vee \\
\qquad\qquad\quad evaluateExpression\,(guard, cxt3) = \textsf{mk-}SEM`BOOL\,(\textsf{true}) \\
\qquad\qquad\ \textsf{then } evaluateExpression\,(elem, cxt3) \\
\qquad\qquad\ \textsf{else NoVAL} \mid \\
\qquad\qquad\quad cxt2 \in cxt\text{-}s\} \setminus \\
\qquad\qquad \{\text{NoVAL}\});
\end{array}
$$

4.7 Binary Expression

public

$$
\begin{array}{l}
evaluateBinary : IOmlBinaryExpression \times Context \xrightarrow{o} \\
\qquad SEM`VAL \\
evaluateBinary\,(expr, cxt) \triangleq \\
\quad \textsf{let } l\text{-}expr = expr.getLhsExpression\,(), \\
\qquad operat = expr.getOperator\,(),
\end{array}
$$

```
    r-expr = expr.getRhsExpression () in
  let l-v = evaluateExpression (l-expr, cxt),
      op = operat.getValue () in
  if (op = OmlBinaryOperatorQuotes`IQAND ∧
      l-v = mk-SEM`BOOL (false))
  then return mk-SEM`BOOL (false)
  elseif (op = OmlBinaryOperatorQuotes`IQOR ∧
          l-v = mk-SEM`BOOL (true)) ∨
         (op = OmlBinaryOperatorQuotes`IQIMPLY ∧
          l-v = mk-SEM`BOOL (false))
  then return mk-SEM`BOOL (true)
  else let r-v = evaluateExpression (r-expr, cxt) in
       return BinOpApply (l-v, op, r-v, expr) ;
BinOpApply : SEM`VAL × ℕ × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
BinOpApply (l-v, op, r-v, expr) ≜
  return cases op :
           (OmlBinaryOperatorQuotes`IQMODIFY) → EvalModify (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQGE) → EvalGE (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQLT) → EvalLT (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQPSUBSET) → EvalPSubset (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQMOD) → EvalMod (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQMAPDOMRESBY) → EvalDomResBy (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQINTER) → EvalInter (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQCOMP) → EvalComp (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQMINUS) → EvalMinus (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQREM) → EvalRem (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQAND) → EvalAnd (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQUNION) → EvalUnion (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQINSET) → EvalInSet (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQEQUIV) → EvalEquiv (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQMAPRNGRESTO) → EvalMapRngResTo (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQITERATE) → EvalIterate (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQSUBSET) → EvalSubset (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQMAPRNGRESBY) → EvalMapRngResBy (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQTUPSEL) → EvalTupSel (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQNOTINSET) → EvalNotInSet (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQMULTIPLY) → EvalMult (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQIMPLY) → EvalImply (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQOR) → EvalOr (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQGT) → EvalGt (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQPLUS) → EvalPlus (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQMUNION) → EvalMUnion (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQMAPDOMRESTO) → EvalMapDomResTo (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQEQ) → EvalEq (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQDIV) → EvalDiv (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQDIFFERENCE) → EvalDifference (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQCONC) → EvalConc (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQLE) → EvalLE (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQDIVIDE) → EvalDivide (l-v, r-v, expr),
           (OmlBinaryOperatorQuotes`IQNE) → EvalNE (l-v, r-v, expr)
         end;
```

$EvalGE : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalGE\ (l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`NUM\ (l\text{-}v) \wedge$ is-$SEM`NUM\ (r\text{-}v)$
  then return mk-$SEM`BOOL\ (l\text{-}v.v \geq r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`NUM\text{-}EXPECTED)$;
    return mk-$SEM`BOOL$ (true)
  );
$EvalModify : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalModify\ (l\text{-}v, r\text{-}v, expr) \triangleq$
  error;
$EvalLT : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalLT\ (l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`NUM\ (l\text{-}v) \wedge$ is-$SEM`NUM\ (r\text{-}v)$
  then return mk-$SEM`BOOL\ (l\text{-}v.v < r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`NUM\text{-}EXPECTED)$;
    return mk-$SEM`BOOL$ (true)
  );
$EvalPSubset : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalPSubset\ (l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`SET\ (l\text{-}v) \wedge$ is-$SEM`SET\ (r\text{-}v)$
  then return mk-$SEM`BOOL\ (l\text{-}v.v \subset r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`SET\text{-}EXPECTED)$;
    return mk-$SEM`BOOL$ (true)
) ;
$EvalMod : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalMod\ (l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`NUM\ (l\text{-}v) \wedge$ is-$SEM`NUM\ (r\text{-}v)$
  then return mk-$SEM`NUM\ (l\text{-}v.v$ mod $r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`NUM\text{-}EXPECTED)$;
    return mk-$SEM`NUM\ (0)$
) ;
$EvalDomResBy : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalDomResBy\ (l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`SET\ (l\text{-}v) \wedge$ is-$SEM`MAP\ (r\text{-}v)$
  then return mk-$SEM`MAP\ (l\text{-}v.v \lhd r\text{-}v.v)$
  elseif $\neg$ is-$SEM`SET\ (l\text{-}v)$
  then ($RTERR`ReportError(expr, RTERR`SET\text{-}EXPECTED)$;
    return mk-$SEM`MAP\ (\{\mapsto\})$
)
  else ($RTERR`ReportError(expr, RTERR`MAP\text{-}EXPECTED)$;
    return mk-$SEM`MAP\ (\{\mapsto\})$
) ;
$EvalInter : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalInter\ (l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`SET\ (l\text{-}v) \wedge$ is-$SEM`SET\ (r\text{-}v)$
  then return mk-$SEM`SET\ (l\text{-}v.v \cap r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`SET\text{-}EXPECTED)$;
    return mk-$SEM`SET\ (\{\})$
  );
$EvalComp : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalComp\ (l\text{-}v, r\text{-}v, expr) \triangleq$
  error;

```
EvalMinus : SEM`VAL × SEM`VAL × IOmlBinaryExpression →^o SEM`VAL
EvalMinus (l-v, r-v, expr) ≜
  if is-SEM`NUM (l-v) ∧ is-SEM`NUM (r-v)
  then return mk-SEM`NUM (l-v.v − r-v.v)
  else (   RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
           return mk-SEM`NUM (0)
       );
EvalRem : SEM`VAL × SEM`VAL × IOmlBinaryExpression →^o SEM`VAL
EvalRem (l-v, r-v, expr) ≜
  if is-SEM`NUM (l-v) ∧ is-SEM`NUM (r-v)
  then return mk-SEM`NUM (l-v.v rem r-v.v)
  else (   RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
           return mk-SEM`NUM (0)
       ) ;
EvalAnd : SEM`VAL × SEM`VAL × IOmlBinaryExpression →^o SEM`VAL
EvalAnd (l-v, r-v, expr) ≜
  if is-SEM`BOOL (l-v) ∧ is-SEM`BOOL (r-v)
  then return mk-SEM`BOOL (l-v.v ∧ r-v.v)
  else (   RTERR`ReportError(expr, RTERR`BOOL-EXPECTED) ;
           return mk-SEM`BOOL (true)
       );
EvalUnion : SEM`VAL × SEM`VAL × IOmlBinaryExpression →^o SEM`VAL
EvalUnion (l-v, r-v, expr) ≜
  if is-SEM`SET (l-v) ∧ is-SEM`SET (r-v)
  then return mk-SEM`SET (l-v.v ∪ r-v.v)
  else (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`SET ({})
       ) ;
EvalInSet : SEM`VAL × SEM`VAL × IOmlBinaryExpression →^o SEM`VAL
EvalInSet (l-v, r-v, expr) ≜
  if is-SEM`SET (r-v)
  then return mk-SEM`BOOL (l-v ∈ r-v.v)
  else (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`BOOL (true)
       );
EvalEquiv : SEM`VAL × SEM`VAL × IOmlBinaryExpression →^o SEM`VAL
EvalEquiv (l-v, r-v, expr) ≜
  error;
EvalMapRngResTo : SEM`VAL × SEM`VAL × IOmlBinaryExpression →^o SEM`VAL
EvalMapRngResTo (l-v, r-v, expr) ≜
  if is-SEM`MAP (l-v) ∧ is-SEM`SET (r-v)
  then return mk-SEM`MAP (l-v.v ⊳ r-v.v)
  elseif is-SEM`MAP (l-v)
  then (   RTERR`ReportError(expr, RTERR`MAP-EXPECTED) ;
           return mk-SEM`MAP ({↦})
       )
  else (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`MAP ({↦})
       ) ;
EvalIterate : SEM`VAL × SEM`VAL × IOmlBinaryExpression →^o SEM`VAL
EvalIterate (l-v, r-v, expr) ≜
  error;
```

```
EvalSubset : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
EvalSubset (l-v, r-v, expr) ≜
  if is-SEM`SET (l-v) ∧ is-SEM`SET (r-v)
  then return mk-SEM`BOOL (l-v.v ⊆ r-v.v)
  else (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`BOOL (true)
       ) ;
EvalMapRngResBy : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
EvalMapRngResBy (l-v, r-v, expr) ≜
  if is-SEM`MAP (l-v) ∧ is-SEM`SET (r-v)
  then return mk-SEM`MAP (l-v.v ⩥ r-v.v)
  elseif ¬ is-SEM`MAP (l-v)
  then (   RTERR`ReportError(expr, RTERR`MAP-EXPECTED) ;
           return mk-SEM`MAP ({↦})
       )
  else (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`MAP ({↦})
       ) ;
EvalTupSel : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
EvalTupSel (l-v, r-v, expr) ≜
  if is-SEM`TUPLE (l-v) ∧ is-SEM`NUM (r-v) ∧
    r-v.v ∈ inds l-v.v
  then return l-v.v (r-v.v)
  elseif ¬ is-SEM`TUPLE (l-v)
  then (   RTERR`ReportError(expr, RTERR`TUPLE-EXPECTED) ;
           return mk-SEM`NUM (0)
       )
  else (   RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
           return mk-SEM`NUM (0)
       ) ;
EvalNotInSet : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
EvalNotInSet (l-v, r-v, expr) ≜
  if is-SEM`SET (r-v)
  then return mk-SEM`BOOL (l-v ∉ r-v.v)
  else (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`BOOL (false)
       ) ;
EvalMult : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
EvalMult (l-v, r-v, expr) ≜
  if is-SEM`NUM (l-v) ∧ is-SEM`NUM (r-v)
  then return mk-SEM`NUM (l-v.v × r-v.v)
  else (   RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
           return mk-SEM`NUM (1)
       );
EvalImply : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
EvalImply (l-v, r-v, expr) ≜
  if is-SEM`BOOL (l-v) ∧ is-SEM`BOOL (r-v)
  then return mk-SEM`BOOL (l-v.v  ⇒  r-v.v)
  else (   RTERR`ReportError(expr, RTERR`BOOL-EXPECTED) ;
           return mk-SEM`BOOL (true)
       ) ;
```

$EvalOr : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalOr\,(l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`BOOL\,(l\text{-}v) \wedge$ is-$SEM`BOOL\,(r\text{-}v)$
  then return mk-$SEM`BOOL\,(l\text{-}v.v \vee r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`BOOL\text{-}EXPECTED)$;
    return mk-$SEM`BOOL\,(\text{true})$
) ;
$EvalGt : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalGt\,(l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`NUM\,(l\text{-}v) \wedge$ is-$SEM`NUM\,(r\text{-}v)$
  then return mk-$SEM`BOOL\,(l\text{-}v.v > r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`NUM\text{-}EXPECTED)$;
    return mk-$SEM`BOOL\,(\text{true})$
) ;
$EvalPlus : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalPlus\,(l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`NUM\,(l\text{-}v) \wedge$ is-$SEM`NUM\,(r\text{-}v)$
  then return mk-$SEM`NUM\,(l\text{-}v.v + r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`NUM\text{-}EXPECTED)$;
    return mk-$SEM`NUM\,(0)$
) ;
$EvalMUnion : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalMUnion\,(l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`MAP\,(l\text{-}v) \wedge$ is-$SEM`MAP\,(r\text{-}v) \wedge$
    let $tmpLv : SEM`MAP = l\text{-}v$,
      $tmpRv : SEM`MAP = r\text{-}v$ in
    let $lv : SEM`VAL \xrightarrow{m} SEM`VAL = tmpLv.v$,
      $rv : SEM`VAL \xrightarrow{m} SEM`VAL = tmpRv.v$,
      $lvD : SEM`VAL\text{-set} = \text{dom } lv$,
      $rvD : SEM`VAL\text{-set} = \text{dom } rv$,
      $s = lvD \cap rvD$ in
    $\forall\, e \in s \cdot lv\,(e) = rv\,(e)$
  then let $tmpLv : SEM`MAP = l\text{-}v$,
      $tmpRv : SEM`MAP = r\text{-}v$ in
    let $lv : SEM`VAL \xrightarrow{m} SEM`VAL = tmpLv.v$,
      $rv : SEM`VAL \xrightarrow{m} SEM`VAL = tmpRv.v$ in
    return mk-$SEM`MAP\,(lv \cup rv)$
  else ($RTERR`ReportError(expr, RTERR`MAP\text{-}EXPECTED)$;
    return mk-$SEM`MAP\,(\{\mapsto\})$
) ;
$EvalMapDomResTo : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalMapDomResTo\,(l\text{-}v, r\text{-}v, expr) \triangleq$
  if is-$SEM`SET\,(l\text{-}v) \wedge$ is-$SEM`MAP\,(r\text{-}v)$
  then return mk-$SEM`MAP\,(l\text{-}v.v \lhd r\text{-}v.v)$
  else ($RTERR`ReportError(expr, RTERR`MAP\text{-}EXPECTED)$;
    return mk-$SEM`MAP\,(\{\mapsto\})$
) ;
$EvalEq : SEM`VAL \times SEM`VAL \times IOmlBinaryExpression \xrightarrow{o} SEM`VAL$
$EvalEq\,(l\text{-}v, r\text{-}v, \text{-}) \triangleq$
  return mk-$SEM`BOOL\,(l\text{-}v = r\text{-}v)$;

```
        EvalDiv : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
        EvalDiv (l-v, r-v, expr) ≜
          if is-SEM`NUM (l-v) ∧ is-SEM`NUM (r-v)
          then return mk-SEM`NUM (l-v.v div r-v.v)
          else (    RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
                    return mk-SEM`BOOL (true)
               );
        EvalDifference : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
        EvalDifference (l-v, r-v, expr) ≜
          if is-SEM`SET (l-v) ∧ is-SEM`SET (r-v)
          then return mk-SEM`SET (l-v.v \ r-v.v)
          else (    RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
                    return mk-SEM`SET ({})
               );
        EvalConc : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
        EvalConc (l-v, r-v, expr) ≜
          if is-SEM`SEQ (l-v) ∧ is-SEM`SEQ (r-v)
          then return mk-SEM`SEQ (l-v.v ⌢ r-v.v)
          else (    RTERR`ReportError(expr, RTERR`SEQ-EXPECTED) ;
                    return mk-SEM`SEQ ([])
               ) ;
        EvalLE : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
        EvalLE (l-v, r-v, expr) ≜
          if is-SEM`NUM (l-v) ∧ is-SEM`NUM (r-v)
          then return mk-SEM`BOOL (l-v.v ≤ r-v.v)
          else (    RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
                    return mk-SEM`BOOL (true)
               ) ;
        EvalDivide : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
        EvalDivide (l-v, r-v, expr) ≜
          if is-SEM`NUM (l-v) ∧ is-SEM`NUM (r-v)
          then return mk-SEM`NUM (l-v.v/r-v.v)
          else (    RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
                    return mk-SEM`BOOL (true)
               ) ;
        EvalNE : SEM`VAL × SEM`VAL × IOmlBinaryExpression →o SEM`VAL
        EvalNE (l-v, r-v, -) ≜
          return mk-SEM`BOOL (l-v ≠ r-v)
functions
public
        evaluateUnary : IOmlUnaryExpression × Context →
                        SEM`VAL
        evaluateUnary (expr, cxt) ≜
          let exprarg = expr.getExpression (),
              operat = expr.getOperator () in
          let v = evaluateExpression (exprarg, cxt),
              op = operat.getValue () in
           UnOpApply (v, op, expr)
operations
        UnOpApply : SEM`VAL × ℕ × IOmlUnaryExpression →o SEM`VAL
        UnOpApply (val, op, expr) ≜
          cases op:
            (OmlUnaryOperatorQuotes`IQABS) → EvalAbs(val, expr) ,
            (OmlUnaryOperatorQuotes`IQINVERSE) → EvalInverse(val, expr) ,
```

```
    (OmlUnaryOperatorQuotes`IQHD) → EvalHd(val, expr),
    (OmlUnaryOperatorQuotes`IQELEMS) → EvalElems(val, expr),
    (OmlUnaryOperatorQuotes`IQINDS) → EvalInds(val, expr),
    (OmlUnaryOperatorQuotes`IQTL) → EvalTl(val, expr),
    (OmlUnaryOperatorQuotes`IQCARD) → EvalCard(val, expr),
    (OmlUnaryOperatorQuotes`IQDUNION) → EvalDUnion(val, expr),
    (OmlUnaryOperatorQuotes`IQPOWER) → EvalPower(val, expr),
    (OmlUnaryOperatorQuotes`IQLEN) → EvalLen(val, expr),
    (OmlUnaryOperatorQuotes`IQPLUS) → EvalUPlus(val, expr),
    (OmlUnaryOperatorQuotes`IQDOM) → EvalDom(val, expr),
    (OmlUnaryOperatorQuotes`IQDMERGE) → EvalMerge(val, expr),
    (OmlUnaryOperatorQuotes`IQDINTER) → EvalDInter(val, expr),
    (OmlUnaryOperatorQuotes`IQNOT) → EvalNot(val, expr),
    (OmlUnaryOperatorQuotes`IQMINUS) → EvalUMinus(val, expr),
    (OmlUnaryOperatorQuotes`IQRNG) → EvalRng(val, expr),
    (OmlUnaryOperatorQuotes`IQFLOOR) → EvalFloor(val, expr),
    (OmlUnaryOperatorQuotes`IQDCONC) → EvalDConc(val, expr)
  end;
EvalAbs : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalAbs (val, expr) △
  if is-SEM`NUM (val)
  then return mk-SEM`NUM (abs val.v)
  else (  RTERR`ReportError(expr, RTERR`NUM-EXPECTED);
          return mk-SEM`NUM (1)
       );
EvalInverse : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalInverse (val, expr) △
  if is-SEM`MAP (val)
  then let tmp : SEM`MAP = val in
       return mk-SEM`MAP (tmp.v-1)
  else (  RTERR`ReportError(expr, RTERR`MAP-EXPECTED);
          return mk-SEM`MAP ({↦})
       );
EvalHd : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalHd (val, expr) △
  if is-SEM`SEQ (val) ∧ val.v ≠ []
  then return hd val.v
  elseif is-SEM`SEQ (val)
  then (  RTERR`ReportError(expr, RTERR`NONEMPTY-SEQ-EXPECTED);
          return mk-SEM`NUM (1)
       )
  else (  RTERR`ReportError(expr, RTERR`SEQ-EXPECTED);
          return mk-SEM`NUM (1)
       );
EvalElems : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalElems (val, expr) △
  if is-SEM`SEQ (val)
  then return mk-SEM`SET (elems val.v)
  else (  RTERR`ReportError(expr, RTERR`SEQ-EXPECTED);
          return mk-SEM`SET ({})
       );
```

```
EvalInds : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalInds (val, expr) ≜
  if is-SEM`SEQ (val)
  then return mk-SEM`SET ({mk-SEM`NUM (i) | i ∈ inds val.v})
  else (   RTERR`ReportError(expr, RTERR`SEQ-EXPECTED) ;
           return mk-SEM`SET ({})
       ) ;
EvalTl : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalTl (val, expr) ≜
  if is-SEM`SEQ (val) ∧ val.v ≠ []
  then return mk-SEM`SEQ (tl val.v)
  elseif is-SEM`SEQ (val)
  then (   RTERR`ReportError(expr, RTERR`NONEMPTY-SEQ-EXPECTED) ;
           return mk-SEM`NUM (1)
       )
  else (   RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
           return mk-SEM`BOOL (true)
       ) ;
EvalCard : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalCard (val, expr) ≜
  if is-SEM`SET (val)
  then return mk-SEM`NUM (card val.v)
  else (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`NUM (0)
       ) ;
EvalDUnion : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalDUnion (val, expr) ≜
  if is-SEM`SET (val) ∧ ∀ s ∈ val.v · is-SEM`SET (s)
  then return mk-SEM`SET (⋃ {s.v | s ∈ val.v})
  elseif is-SEM`SET (val)
  then (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`NUM (1)
       )
  else (   RTERR`ReportError(expr, RTERR`ALL-SETS-EXPECTED) ;
           return mk-SEM`NUM (1)
       ) ;
EvalPower : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalPower (val, expr) ≜
  if is-SEM`SET (val)
  then return mk-SEM`SET ({mk-SEM`SET (s) | s ∈ F (val.v)})
  else (   RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
           return mk-SEM`SET ({})
       ) ;
EvalLen : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalLen (val, expr) ≜
  if is-SEM`SEQ (val)
  then return mk-SEM`NUM (len val.v)
  else (   RTERR`ReportError(expr, RTERR`SEQ-EXPECTED) ;
           return mk-SEM`NUM (1)
       ) ;
EvalUPlus : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalUPlus (val, expr) ≜
  if is-SEM`NUM (val)
  then return val
```

```
    else (    RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
              return mk-SEM`NUM (1)
         ) ;
EvalDom : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalDom (val, expr) ≜
    if is-SEM`MAP (val)
    then return mk-SEM`SET (dom val.v)
    else (    RTERR`ReportError(expr, RTERR`MAP-EXPECTED) ;
              return mk-SEM`MAP ({↦})
         ) ;
EvalMerge : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalMerge (val, expr) ≜
    if is-SEM`SET (val) ∧ ∀ m ∈ val.v · is-SEM`MAP (m)
    then return mk-SEM`MAP (merge {m.v | m ∈ val.v})
    elseif is-SEM`SET (val)
    then (    RTERR`ReportError(expr, RTERR`MAP-EXPECTED) ;
              return mk-SEM`MAP ({↦})
         )
    else (    RTERR`ReportError(expr, RTERR`ALL-MAPS-EXPECTED) ;
              return mk-SEM`MAP ({↦})
         ) ;
EvalDInter : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalDInter (val, expr) ≜
    if is-SEM`SET (val) ∧ ∀ s ∈ val.v · is-SEM`SET (s)
    then return mk-SEM`SET (⋂ {s.v | s ∈ val.v})
    elseif is-SEM`SET (val)
    then (    RTERR`ReportError(expr, RTERR`SET-EXPECTED) ;
              return mk-SEM`SET ({})
         )
    else (    RTERR`ReportError(expr, RTERR`ALL-SETS-EXPECTED) ;
              return mk-SEM`SET ({})
         ) ;
EvalNot : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalNot (val, expr) ≜
    if is-SEM`BOOL (val)
    then return mk-SEM`BOOL (¬ val.v)
    else (    RTERR`ReportError(expr, RTERR`BOOL-EXPECTED) ;
              return mk-SEM`BOOL (true)
         ) ;
EvalUMinus : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalUMinus (val, expr) ≜
    if is-SEM`NUM (val)
    then return mk-SEM`NUM (− val.v)
    else (    RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
              return mk-SEM`NUM (1)
         ) ;
EvalRng : SEM`VAL × IOmlUnaryExpression →o SEM`VAL
EvalRng (val, expr) ≜
    if is-SEM`MAP (val)
    then return mk-SEM`SET (rng val.v)
    else (    RTERR`ReportError(expr, RTERR`MAP-EXPECTED) ;
              return mk-SEM`SET ({})
         ) ;
```

```
    EvalFloor : SEM`VAL × IOmlUnaryExpression ^o→ SEM`VAL
    EvalFloor (val, expr) ≜
      if is-SEM`NUM (val)
      then return mk-SEM`NUM (floor val.v)
      else (    RTERR`ReportError(expr, RTERR`NUM-EXPECTED) ;
                return mk-SEM`NUM (1)
           ) ;
    EvalDConc : SEM`VAL × IOmlUnaryExpression ^o→ SEM`VAL
    EvalDConc (val, expr) ≜
      if is-SEM`SEQ (val) ∧ ∀ s ∈ elems let tmp1 : SEM`SEQ = val in
        tmp1.v · is-SEM`SEQ (s)
      then let tmp : SEM`SEQ = val,
               l : SEM`SEQ* = tmp.v in
           return mk-SEM`SEQ (conc [let tmp2 : SEM`SEQ = l (i) in
                                    tmp2.v |
                                        i ∈ inds l])
      else (    RTERR`ReportError(expr, RTERR`ALL-SEQS-EXPECTED) ;
                return mk-SEM`SEQ ([])
           ) ;
public
    evaluateNewExpression : IOmlNewExpression × Context ^o→
                                SEM`VAL
    evaluateNewExpression (expr, cxt) ≜
      let clnm = expr.getName (),
          par-l = expr.getExpressionList (),
          v-l = [evaluateExpression (par-l (i), cxt) | i ∈ inds par-l] in
      return mk-SEM`OBJ (clnm.getIdentifier (), v-l);
public
    evaluateIfExpression : IOmlIfExpression × Context ^o→
                               SEM`VAL
    evaluateIfExpression (expr, cxt) ≜
      let testexpr = expr.getIfExpression (),
          thenexpr = expr.getThenExpression (),
          elseifexpr = expr.getElseifExpressionList (),
          elseexpr = expr.getElseExpression (),
          testval = evaluateExpression (testexpr, cxt) in
      if ¬ is-SEM`BOOL (testval)
      then error
      elseif testval = mk-SEM`BOOL (true)
      then evaluateExpression(thenexpr, cxt)
      elseif elseifexpr = []
      then evaluateExpression(elseexpr, cxt)
      else evaluateElseIfExpression(elseifexpr, elseexpr, cxt) ;
public
    evaluateElseIfExpression : IOmlElseIfExpression^+ × IOmlIfExpression × Context ^o→ SEM`VAL
    evaluateElseIfExpression (elseifexpr-l, elseexpr, cxt) ≜
      let first = hd elseifexpr-l,
          testexpr = first.getElseifExpression (),
          thenexpr = first.getThenExpression (),
          testval = evaluateExpression (testexpr, cxt) in
      if ¬ is-SEM`BOOL (testval)
      then error
      elseif testval = mk-SEM`BOOL (true)
      then evaluateExpression(thenexpr, cxt)
```

```
    elseif len elseifexpr-l = 1
    then evaluateExpression(elseexpr, cxt)
    else evaluateElseIfExpression(tl elseifexpr-l, elseexpr, cxt) ;
public
  evaluateLetExpression : IOmlLetExpression × Context ^o→
                          SEM`VAL
  evaluateLetExpression (expr, cxt) ≜
    let def-l = expr.getDefinitionList (),
        inexpr = expr.getExpression (),
        cxt2 = EvalDefList (def-l, cxt) in
    evaluateExpression(inexpr, cxt2) ;
  EvalDefList : IOmlValueShape* × Context ^o→ Context
  EvalDefList (defi-l, cxt) ≜
    (   dcl cxt2 : Context := cxt;
        for defi in defi-l
        do let pat = defi.getPattern (),
              expr = defi.getExpression (),
              val = evaluateExpression (expr, cxt2) in
           cxt2 := cxt2 † PatternMatch (pat, val);
        return cxt2
    );
public
  evaluateFieldSelect : IOmlFieldSelect × Context ^o→
                        SEM`VAL
  evaluateFieldSelect (expr, cxt) ≜
    let argexpr = expr.getExpression (),
        name = expr.getName (),
        val = evaluateExpression (argexpr, cxt),
        id = name.getIdentifier () in
    if is-SEM`REC (val)
    then specdefs.LookUpRecSel(val, id, expr)
    elseif is-SEM`OBJ (val)
    then let clnm = val.nm,
             cexpr = specdefs.LookUp (clnm, id) in
         evaluateExpression(cexpr, cxt)
    else error;
public
  evaluateRecordConstructor : IOmlRecordConstructor × Context ^o→
                              SEM`VAL
  evaluateRecordConstructor (expr, cxt) ≜
    let name = expr.getName (),
        e-l = expr.getExpressionList (),
        clnm = if name.hasClassIdentifier ()
               then name.getClassIdentifier ()
               else curcl,
        id = name.getIdentifier (),
        v-l = [evaluateExpression (e-l (i), cxt) | i ∈ inds e-l] in
    return mk-SEM`REC (mk-DEF`Name (clnm, id), v-l);
public
  evaluateTokenExpression : IOmlTokenExpression × Context ^o→
                            SEM`VAL
  evaluateTokenExpression (texpr, cxt) ≜
    let expr = texpr.getExpression (),
```

```
        val = evaluateExpression (expr, cxt) in
    return mk-SEM`TOKEN (val);
public
  evaluateName : IOmlName × Context →o
                 SEM`VAL
  evaluateName (expr, cxt) △
    let nm = expr.getIdentifier () in
    if nm ∈ dom cxt
    then return cxt (nm)
    else let hascl = expr.hasClassIdentifier (),
             clnm = if hascl
                    then expr.getClassIdentifier ()
                    else curcl,
             expr2 = specdefs.LookUp (clnm, nm) in
         evaluateExpression(expr2, cxt) ;
public
  getValueOfSymLit : IOmlSymbolicLiteralExpression →o
                     SEM`VAL
  getValueOfSymLit (expr) △
    let val = expr.getLiteral () in
    return getValue (val);
public
  getValue : IOmlLiteral →o SEM`VAL
  getValue (lit) △
     cases true:
       (isofclass (IOmlNumericLiteral, lit)) →
            return getValueNumeric (lit),
       (isofclass (IOmlRealLiteral, lit)) →
            return getValueReal (lit),
       (isofclass (IOmlBooleanLiteral, lit)) →
            return getValueBoolean (lit),
       (isofclass (IOmlCharacterLiteral, lit)) →
            return getValueChar (lit),
       (isofclass (IOmlTextLiteral, lit)) →
            return getValueText (lit),
       (isofclass (IOmlQuoteLiteral, lit)) →
            return getValueQuote (lit),
       (isofclass (IOmlNilLiteral, lit)) →
            return getValueNil (lit)
    end;
public
  getValueNumeric : IOmlNumericLiteral →o
                    SEM`NUM
  getValueNumeric (lit) △
    return mk-SEM`NUM (lit.getVal ());
public
  getValueReal : IOmlRealLiteral →o
                 SEM`NUM
  getValueReal (lit) △
    return mk-SEM`NUM (lit.getVal ());
public
```

```
        getValueBoolean : IOmlBooleanLiteral ^o→
                          SEM`BOOL
        getValueBoolean (lit) △
          return mk-SEM`BOOL (lit.getVal ());
public
        getValueChar : IOmlCharacterLiteral ^o→
                       SEM`CHAR
        getValueChar (lit) △
          return mk-SEM`CHAR (lit.getVal ());
public
        getValueText : IOmlTextLiteral ^o→
                       SEM`SEQ
        getValueText (lit) △
          let str = lit.getVal () in
          return mk-SEM`SEQ ([mk-SEM`CHAR (str (i)) | i ∈ inds str]);
public
        getValueQuote : IOmlQuoteLiteral ^o→
                        SEM`QUOTE
        getValueQuote (lit) △
          return mk-SEM`QUOTE (lit.getVal ());
public
        getValueNil : IOmlNilLiteral ^o→
                      SEM`NIL
        getValueNil (-) △
          return mk-SEM`NIL ()
```

4.8 Evaluation of Patterns and Bindings

```
functions
        evalBindList : IOmlBind* × Context → Context-set
        evalBindList (b-l, cxt) △
          if b-l = []
          then {{↦}}
          else let c-s1 = evalBind (hd b-l, cxt),
                   c-s2 = evalBindList (tl b-l, cxt) in
               CombineContexts (c-s1, c-s2)
operations
        evalBind : IOmlBind × Context ^o→ Context-set
        evalBind (bind, cxt) △
          if isofclass (IOmlSetBind, bind)
          then evalSetBind(bind, cxt)
          else (    RTERR`ReportError(bind, RTERR`TYPE-BIND-EVAL) ;
                    return {{↦}}
               );
        evalSetBind : IOmlSetBind × Context ^o→ Context-set
        evalSetBind (bind, cxt) △
          let p-l = bind.getPattern (),
              s = bind.getExpression (),
              s-v = evaluateExpression (s, cxt),
```

```
        c-l-s = if is-SEM`SET (s-v)
                then {[PatternMatch (p-l (i), v) | i ∈ inds p-l] |
                      v ∈ s-v.v}
                else {} in
    (   if ¬ is-SEM`SET (s-v)
        then (  RTERR`ReportError(bind, RTERR`TYPE-BIND-EVAL) ;
                return {{↦}}
              )
        else return {MergeContextList (c-l) | c-l ∈ c-l-s}
    )
functions
public
    CombineContexts : Context-set × Context-set →
                      Context-set
    CombineContexts (c-s1, c-s2) △
      {c1 † c2 |
          c1 ∈ c-s1, c2 ∈ c-s2};
public
    MergeContextList : Context+ → Context
    MergeContextList (cxt-l) △
      if len cxt-l = 1
      then hd cxt-l
      else hd cxt-l † MergeContextList (tl cxt-l);
    LenCList : Context+ → ℕ
    LenCList (c-l) △
      len c-l
operations
public
    PatternMatch : IOmlPattern × SEM`VAL →o Context
    PatternMatch (pat, val) △
      if isofclass (IOmlPatternIdentifier, pat)
      then return MatchPatternId (pat, val)
      else error
functions
    MatchPatternId : IOmlPatternIdentifier × SEM`VAL → Context
    MatchPatternId (patid, val) △
      let id = patid.getIdentifier () in
      {id ↦ val}
end Eval
```

Test Suite : vdm.tc
Class : Eval

Name	#Calls	Coverage
Eval`Eval	74	√
Eval`EvalEq	0	0%
Eval`EvalGE	10	65%
Eval`EvalGt	0	0%
Eval`EvalHd	0	0%
Eval`EvalLE	0	0%
Eval`EvalLT	12	65%
Eval`EvalNE	0	0%
Eval`EvalOr	0	0%
Eval`EvalTl	0	0%

Name	#Calls	Coverage
Eval‘EvalAbs	0	0%
Eval‘EvalAnd	1	65%
Eval‘EvalDiv	2	65%
Eval‘EvalDom	0	0%
Eval‘EvalLen	0	0%
Eval‘EvalMod	0	0%
Eval‘EvalNot	0	0%
Eval‘EvalRem	0	0%
Eval‘EvalRng	0	0%
Eval‘EvalCard	0	0%
Eval‘EvalComp	0	0%
Eval‘EvalConc	0	0%
Eval‘EvalInds	0	0%
Eval‘EvalMult	1	65%
Eval‘EvalPlus	0	0%
Eval‘LenCList	0	0%
Eval‘evalBind	4	50%
Eval‘getValue	185	√
Eval‘EvalDConc	0	0%
Eval‘EvalElems	0	0%
Eval‘EvalEquiv	0	0%
Eval‘EvalFloor	0	0%
Eval‘EvalImply	0	0%
Eval‘EvalInSet	1	56%
Eval‘EvalInter	1	65%
Eval‘EvalMerge	0	0%
Eval‘EvalMinus	2	65%
Eval‘EvalPower	0	0%
Eval‘EvalUPlus	0	0%
Eval‘EvalUnion	0	0%
Eval‘UnOpApply	0	0%
Eval‘BinOpApply	31	37%
Eval‘EvalDInter	0	0%
Eval‘EvalDUnion	0	0%
Eval‘EvalDivide	0	0%
Eval‘EvalMUnion	0	0%
Eval‘EvalModify	0	0%
Eval‘EvalSubset	0	0%
Eval‘EvalTupSel	0	0%
Eval‘EvalUMinus	0	0%
Eval‘EvalDefList	1	√
Eval‘EvalInverse	0	0%
Eval‘EvalIterate	0	0%
Eval‘EvalPSubset	0	0%
Eval‘evalSetBind	4	82%
Eval‘getValueNil	2	√
Eval‘EvalDomResBy	0	0%
Eval‘EvalNotInSet	0	0%
Eval‘PatternMatch	95	90%
Eval‘evalBindList	8	√

Name	#Calls	Coverage
Eval‘evaluateName	37	√
Eval‘getValueChar	1	√
Eval‘getValueReal	1	√
Eval‘getValueText	1	√
Eval‘evaluateUnary	0	0%
Eval‘getValueQuote	1	√
Eval‘EvalDifference	1	65%
Eval‘MatchPatternId	95	√
Eval‘evaluateBinary	31	78%
Eval‘CombineContexts	53	√
Eval‘EvalMapDomResTo	0	0%
Eval‘EvalMapRngResBy	0	0%
Eval‘EvalMapRngResTo	0	0%
Eval‘getValueBoolean	58	√
Eval‘getValueNumeric	121	√
Eval‘MergeContextList	209	√
Eval‘evaluateSetRange	3	84%
Eval‘getValueOfSymLit	185	√
Eval‘evaluateExpression	311	87%
Eval‘evaluateFieldSelect	3	83%
Eval‘evaluateIfExpression	3	71%
Eval‘evaluateLetExpression	1	√
Eval‘evaluateNewExpression	7	72%
Eval‘evaluateMapEnumeration	0	0%
Eval‘evaluateSeqEnumeration	0	0%
Eval‘evaluateSetEnumeration	30	√
Eval‘evaluateTokenExpression	1	√
Eval‘evaluateElseIfExpression	0	0%
Eval‘evaluateSetComprehension	4	√
Eval‘evaluateRecordConstructor	3	96%
Eval‘evaluateBracketedExpression	3	√
Total Coverage		**39%**

5 Semantic values during evaluation

This class provides a representation for semantic values resulting from evaluation of expressions. In addition it provides functionality to transform such semantic values to semantically equivalent syntactic expression in AST format. These can then subsequently be translated into VDM++ concrete syntax in the form of a sequence of chars.

class SEM

types

public $VAL = BasicVal \mid SEQ \mid SET \mid MAP \mid TUPLE \mid REC \mid TOKEN \mid OBJ \mid OBJ\text{-}Ref$;

public $BasicVal = BOOL \mid NUM \mid CHAR \mid QUOTE \mid NIL$;

public

$BOOL :: v : \mathbb{B}$;

public

$NUM :: v : \mathbb{R}$;

public

$CHAR :: v$: char;

public
$QUOTE :: v : \text{char}^*;$
public
$NIL :: \ ;$
public
$SEQ :: v : VAL^*;$
public
$SET :: v : VAL\text{-set};$
public
$MAP :: v : VAL \xrightarrow{m} VAL;$
public
$TUPLE :: v : VAL^*;$
public
$REC :: tag : DEF`Name$
$v : VAL^*;$
public
$TOKEN :: v : VAL;$
public
$OBJ :: nm : \text{char}^*$
$e\text{-}l : VAL^*;$
public
$OBJ\text{-}Ref :: \ \mathbb{N}$
instance variables
static $cacheval : VAL \xrightarrow{m} IOmlExpression := \{\mapsto\};$

operations
public static
$VAL2IOmlExpr : VAL \xrightarrow{o} IOmlExpression$
$VAL2IOmlExpr\,(val) \triangleq$
if $val \in \text{dom}\ cacheval$
then return $cacheval\,(val)$

```
        else let e = cases val :
                        mk-BOOL (v) → let l = new OmlBooleanLiteral (v) in
                                      new OmlSymbolicLiteralExpression (l),
                        mk-NUM (v) → let l = new OmlRealLiteral (v) in
                                     new OmlSymbolicLiteralExpression (l),
                        mk-CHAR (v) → let l = new OmlCharacterLiteral (v) in
                                      new OmlSymbolicLiteralExpression (l),
                        mk-QUOTE (v) → let l = new OmlQuoteLiteral (v) in
                                       new OmlSymbolicLiteralExpression (l),
                        mk-NIL () → let l = new OmlNilLiteral () in
                                    new OmlSymbolicLiteralExpression (l),
                        mk-SET (v) → let e-l = VALSet2IOmlExpr (v) in
                                     new OmlSetEnumeration (e-l),
                        mk-SEQ (v) → let e-l = VALSeq2IOmlExpr (v) in
                                     new OmlSequenceEnumeration (e-l),
                        mk-MAP (v) → let e-l = VALMap2IOmlExpr (v) in
                                     new OmlMapEnumeration (e-l),
                        mk-TUPLE (v) → let e-l = VALSeq2IOmlExpr (v) in
                                       new OmlTupleConstructor (e-l),
                        mk-REC (t, v) → let e-l = VALSeq2IOmlExpr (v),
                                            nm = new OmlName (t.clnm, t.tnm) in
                                        new OmlRecordConstructor (nm, e-l),
                        mk-TOKEN (v) → let v1 = VAL2IOmlExpr (v) in
                                       new OmlTokenExpression (v1),
                        mk-OBJ (id, vl) → let v1 = VALSeq2IOmlExpr (vl),
                                              nm = new OmlName (nil , id) in
                                          new OmlNewExpression (nm, [], v1),
                        others → undefined
                    end in
                (   cacheval(val) := e;
                    return e
                );
public static
        VALSet2IOmlExpr : VAL-set -o-> IOmlExpression*
        VALSet2IOmlExpr (val-s) ≜
          (   dcl e-l : IOmlExpression* := [];
              for all val ∈ val-s
              do e-l := [VAL2IOmlExpr (val)] ⌢ e-l;
              return e-l
          );
public static
        VALMap2IOmlExpr : VAL -m-> VAL -o-> IOmlMaplet*
        VALMap2IOmlExpr (val-m) ≜
          (   dcl e-l : IOmlMaplet* := [];
              for all val ∈ dom val-m
              do let d = VAL2IOmlExpr (val),
                     r = VAL2IOmlExpr (val-m (val)) in
                 e-l := [new OmlMaplet (d, r)] ⌢ e-l;
              return e-l
          )
functions
public static
```

$VALSeq2IOmlExpr : VAL^* \to IOmlExpression^*$
$VALSeq2IOmlExpr\,(val\text{-}l) \triangleq$
$[VAL2IOmlExpr\,(val\text{-}l\,(i)) \mid i \in \mathsf{inds}\ val\text{-}l]$

end *SEM*

Test Suite : vdm.tc
Class : SEM

Name	#Calls	Coverage
SEM‘VAL2IOmlExpr	96	43%
SEM‘VALMap2IOmlExpr	0	0%
SEM‘VALSeq2IOmlExpr	5	$\surd$
SEM‘VALSet2IOmlExpr	0	0%
Total Coverage		**35%**

6 Filtering dublicate errors away

class *Filtering*
values
public
$tcPass : \mathbb{N} = 1;$
public
$tcFail : \mathbb{N} = 2;$
public
$tcInconc : \mathbb{N} = 3$
instance variables
$allTestCases : (DEF‘Name \times \mathbb{N}_1) \xrightarrow{m} IOmlExpression^*;$
$allTestCasesAsSeq : (DEF‘Name \times \mathbb{N}_1)^*;$
$currentTest : \mathbb{N} := 0;$
$succTestCaes : (DEF‘Name \times \mathbb{N}_1) \xrightarrow{m} \mathbb{N} := \{\mapsto\};$
$failedTestCases : (DEF‘Name \times \mathbb{N}_1) \xrightarrow{m} \mathbb{N};$
$argfiles : (DEF‘Name \times \mathbb{N}_1) \xrightarrow{m} \mathsf{char}^* := \{\mapsto\};$
$resfiles : (DEF‘Name \times \mathbb{N}_1) \xrightarrow{m} \mathsf{char}^* := \{\mapsto\};$
$passTestCases : \mathbb{N};$
$setOfSelected : DEF‘Identifier\text{-}\mathsf{set};$
$setOfDeleted : DEF‘Identifier\text{-}\mathsf{set};$
$tb : ToolBox;$
$ppvisitor : Oml2VppVisitor := \mathsf{new}\ Oml2VppVisitor\,();$
$\mathsf{static}\ errmsgs : DEF‘Identifier \xrightarrow{m} ErrMsg\text{-}\mathsf{set} := \{\mapsto\};$

types
public

$Statistics :: initText : \text{char}^*$
$selected : (\text{char}^*)^*$
$deletedText : \text{char}^*$
$deleted : (\text{char}^*)^*$
$totSelectedText : \text{char}^*$
$totSelected : \mathbb{N}$
$totFailedText : \text{char}^*$
$totFailed : \mathbb{N}$
$totDeletedText : \text{char}^*$
$totDeleted : \mathbb{N}$
$percentText : \text{char}^*$
$percentFailed : \mathbb{R}$
$percentDeletedText : \text{char}^*$
$percentDeleted : \mathbb{R};$

public

$ErrMsg :: line : \mathbb{N}$
$col : \mathbb{N}$
$mes : \text{char}^*$

operations

public

$Filtering : (DEF`Name \xrightarrow{m} IOmlExpression^*\text{-set}) \times ToolBox \xrightarrow{o} Filtering$
$Filtering\,(t, tool) \triangleq$
$(\ allTestCases := fillMap\,(t);$
$allTestCasesAsSeq := StdLib`SetToSeq[DEF`Name \times \mathbb{N}_1]\,(\text{dom } allTestCases);$
$failedTestCases := \{\mapsto\};$
$passTestCases := 0;$
$setOfSelected := \{\};$
$setOfDeleted := \{\};$
$tb := tool$
$);$

public static

$AddErrMsg : DEF`Identifier \times \mathbb{N} \times \mathbb{N} \times RTERR`ERR \xrightarrow{o} ()$
$AddErrMsg\,(clnm, line, col, errmsg) \triangleq$
is not yet specified;

public static

$GetErrMsg : DEF`Identifier \xrightarrow{o} ErrMsg\text{-set}$
$GetErrMsg\,(clnm) \triangleq$
if $clnm \in \text{dom } errmsgs$
then return $errmsgs\,(clnm)$
else return $\{\}$

functions

public

$fillMap : DEF`Name \xrightarrow{m} IOmlExpression^*\text{-set} \rightarrow (DEF`Name \times \mathbb{N}_1) \xrightarrow{m} IOmlExpression^*$
$fillMap\,(tc\text{-}m\text{-}s) \triangleq$
if $tc\text{-}m\text{-}s = \{\mapsto\}$
then $\{\mapsto\}$
else let $nm \in \text{dom } tc\text{-}m\text{-}s$ in
$SpreadTestCase\,(nm, tc\text{-}m\text{-}s\,(nm), 1) \cup^{m}$
$fillMap\,(\{nm\} \lhd tc\text{-}m\text{-}s);$

```
MapSize : DEF`Name ->m IOmlExpression*-set -> N
MapSize (m) ≜
  card dom m;
SpreadTestCase : DEF`Name × (IOmlExpression*-set) × N →
                 (DEF`Name × N1) ->m IOmlExpression*
SpreadTestCase (nm, tc-s, n) ≜
  if tc-s = {}
  then {↦}
  else let tc ∈ tc-s in
       {mk-(nm, n) ↦ tc} ⊌
       SpreadTestCase (nm, tc-s \ {tc}, n + 1);
SetSize : DEF`Name × (IOmlExpression*-set) × N → N
SetSize (-, s, -) ≜
  card s
```

Running the actual test cases with filtering switched on is done by the `filter` operation which in turns makes use of the `RunTestCase` operation in case no earlier test cases have failed on the same prefix.

```
operations
public
        filterAll : () ->o B
        filterAll () ≜
          (   dcl found-errs : B := false;
              for all nm ∈ dom allTestCases
              do let tc-ast = allTestCases (nm) in
                   if ExecuteTraceTestCase (nm, tc-ast)
                   then found-errs := true;
              return found-errs
          );
public
        filterNext : () ->o B
        filterNext () ≜
          (   dcl found-errs : B := false;
              currentTest := currentTest + 1;
              let nm = allTestCasesAsSeq (currentTest),
                  tc-ast = allTestCases (nm) in
              if ExecuteTraceTestCase (nm, tc-ast)
              then found-errs := true;
              return found-errs
          )
        pre currentTest + 1 ≤ len allTestCasesAsSeq ;
public
        isAllSingleStepTestsCompleted : () ->o B
        isAllSingleStepTestsCompleted () ≜
          return currentTest = len allTestCasesAsSeq;
public
        ExecuteTraceTestCase : (DEF`Name × N1) × IOmlExpression* ->o B
        ExecuteTraceTestCase (nm, tc-ast) ≜
          (   dcl found-errs : B := false,
                  index : N;
              if NoFailedPrefix (tc-ast)
              then (    dcl argexpr-l : char** := [],
                            res-l : char** := [];
```

```
                    tb.InitToolbox() ;
                    tb.Create((nm.#1).clnm, "1internal") ;
                    index := 1;
                    while index ≤ len tc-ast
                    do (    ppvisitor.visitExpression(tc-ast (index)) ;
                            let expr-str = ppvisitor.result,
                                res = tb.vdmToolsCall (expr-str) in
                            (    argexpr-l := argexpr-l ↷ [expr-str];
                                 res-l := res-l ↷ [res.output];
                                 if res.verdict = tcFail
                                 then (    failedTestCases(nm) := index;
                                           index := len tc-ast;
                                           found-errs := true
                                      ) ;
                                 index := index + 1
                            ) ;
                            passTestCases := passTestCases + 1
                       )
                )
           else (    setOfDeleted := setOfDeleted ∪ {Name2String (nm)};
                     found-errs := true
                ) ;
           return found-errs
      );
    NoFailedPrefix : IOmlExpression* →o B
    NoFailedPrefix (e-l) ≜
      return ∃ tcnm ∈ dom failedTestCases ·
                 let failindex = failedTestCases (tcnm) in
                 e-l (1, ..., failindex) = allTestCases (tcnm) (1, ..., failindex)
functions
    Name2String : (DEF`Name × N1) → DEF`Identifier
    Name2String (mk-(mk-DEF`Name (clnm, defnm), n)) ≜
      (if clnm = nil
       then ""
       else clnm) ↷
      defnm ↷
      StdLib`ToStringInt (n)
operations
public
    ppTestCases : () →o Statistics
    ppTestCases () ≜
      let f = filterAll (),
```

```
        firstFailed = card (setOfSelected) − passTestCases in
    if f
    then return mk-Statistics
                    (
                     "Executed test cases: \n",
                     StdLib`SetToSeq[DEF`Identifier]
                         (
                          {tcnm ↷ "\n" | tcnm ∈ setOfSelected}),
                     "\n Failed test cases:\n",
                     StdLib`SetToSeq[DEF`Identifier]
                         (
                          {tcnm ↷ "\n" | tcnm ∈ setOfDeleted}),
                     "\n\n Statistics:\n Number of selected test cases: ",
                     card (setOfSelected),
                     "\n Number of failed test cases: ",
                     card (setOfDeleted),
                     "\n Number of deleted test cases: ",
                     card (setOfDeleted) − firstFailed,
                     "\n Percentage of failed test cases: ",
                     ((card (setOfDeleted))/card dom allTestCases) × 100,
                     "Percentage of deleted test cases: ",
                     ((card (setOfDeleted) − firstFailed)/card dom allTestCases) × 100)
    else return mk-Statistics ([], [], [], [], [], 0, [], 0, [], 0, [], 0, [], 0)
end Filtering
```

Test Suite : vdm.tc
Class : Expanded

Name	#Calls	Coverage
Total Coverage		**1%**

7 Interface to VDMTools' Interpreter

The `ToolBox` class represent an interface to VDMTools enabling interpretation of test cases automatically generated using the combinational test generation approach. Probably it will subsequently be necessary to change the `OmlSpecifications` in its AST format to simply be a set of names of the files in which these are stored such that these can be loaded into VDMTools.

```
class ToolBox
types
public
        interpreterResult :: verdict : ℕ
                             output : char*
        inv x ≜ x.verdict ∈ {0, 1, 2, 3}
instance variables
        public specsFiles : DEF`Identifier-set;

operations
public
```

```
        ToolBox : DEF`Identifier-set ^o→
                  ToolBox
        ToolBox (sp) △
          (   specsFiles := sp;
              InitToolbox()
          );
public
        InitToolbox : () ^o→ ()
        InitToolbox () △
          skip;
public
        Create : DEF`Identifier × DEF`Identifier ^o→ ()
        Create (clnm, instnm) △
          skip;
public
        vdmToolsCall : char* ^o→
                       interpreterResult
        vdmToolsCall (-) △
          return mk-interpreterResult (2, "Failed")
end ToolBox
```

8 Visitor for pretty-pinting ASTs

```
class Oml2VppVisitor is subclass of OmlVisitor
types
         String = char*
values
private
        nl : char* = "\n"
instance variables
        public result : char* := [];
        private lvl : ℕ := 0;

operations
private
        printNodeField : IOmlNode ^o→ ()
        printNodeField (pNode) △ pNode.
          accept(self) ;
private
        printBoolField : 𝔹 ^o→ ()
        printBoolField (pval) △
          result := if pval
                    then "true"
                    else "false";
private
        printNatField : ℕ ^o→ ()
        printNatField (pval) △
          result := StdLib`ToStringInt (pval);
private
        printRealField : ℝ ^o→ ()
        printRealField (pval) △
          result := VDMUtil`val2seq-of-char[ℝ] (pval);
private
```

```
        printCharField : char ^o→ ()
        printCharField (pval) ≜
          result := [pval];
private
        printField : IOmlNode`FieldValue ^o→ ()
        printField (fld) ≜
          if is-ℬ (fld)
          then printBoolField(fld)
          elseif is-char (fld)
          then printCharField(fld)
          elseif is-ℕ (fld)
          then printNatField(fld)
          elseif is-ℝ (fld)
          then printRealField(fld)
          elseif isofclass (IOmlNode, fld)
          then printNodeField(fld)
          else printStringField(fld) ;
private
        printStringField : char* ^o→ ()
        printStringField (str) ≜
          result := "\"" ↷ str ↷ "\"";
private
        printSeqofField : IOmlNode`FieldValue* ^o→ ()
        printSeqofField (pval) ≜
          (   dcl str : char* := "",
                  cnt : ℕ := len pval;
              while cnt > 0
              do (   printField(pval (len pval − cnt + 1)) ;
                     str := str ↷ result;
                     if cnt > 1
                     then str := str ↷ ",  ";
                     cnt := cnt − 1
                 ) ;
              result := str
          );
public
        visitNode : IOmlNode ^o→ ()
        visitNode (pNode) ≜ pNode.
          accept(self) ;
public
        visitDocument : IOmlDocument ^o→ ()
        visitDocument (pcmp) ≜
          (   dcl str : char* := "−BEGIN  FileName :  " ↷ pcmp.getFilename () ↷ nl;
              if pcmp.hasSpecifications ()
              then visitSpecifications(pcmp.getSpecifications ()) ;
              result := str ↷ result ↷ "−END  FileName :  " ↷ pcmp.getFilename ()
          );
public
        visitSpecifications : IOmlSpecifications ^o→ ()
        visitSpecifications (pcmp) ≜
          (   dcl str : char* := nl;
              for node in pcmp.getClassList ()
```

```
            do (    printNodeField(node) ;
                    str := str ↷ nl ↷ result ↷ nl
               ) ;
            result := str
      );
public
      visitClass : IOmlClass ⟶ ()
      visitClass (pcmp) ≜
        (   dcl str : char* := "class " ↷ pcmp.getIdentifier ();
            if pcmp.hasInheritanceClause ()
            then printNodeField(pcmp.getInheritanceClause ())
            else result := "";
            str := str ↷ result ↷ nl;
            for db in pcmp.getClassBody ()
            do (    printNodeField(db) ;
                    str := str ↷ nl ↷ result
               ) ;
            result := str ↷ nl ↷ "end " ↷ pcmp.getIdentifier ()
        );
public
      visitInheritanceClause : IOmlInheritanceClause ⟶ ()
      visitInheritanceClause (pcmp) ≜
        (   dcl str : char* := " is subclass of ",
                list : String* := pcmp.getIdentifierList (),
                length : ℕ := len list,
                i : ℕ := 1;
            while i ≤ length
            do (    str := str ↷ list (i);
                    i := i + 1;
                    if i ≤ length
                    then str := str ↷ " , "
               ) ;
            result := str ↷ nl
        );
public
      visitValueDefinitions : IOmlValueDefinitions ⟶ ()
      visitValueDefinitions (pcmp) ≜
        (   dcl str : char* := nl ↷ "values" ↷ nl;
            for db in pcmp.getValueList ()
            do (    printNodeField(db) ;
                    str := str ↷ result ↷ nl
               ) ;
            if len pcmp.getValueList () = 0
            then result := ""
            else result := str
        );
public
      visitValueDefinition : IOmlValueDefinition ⟶ ()
      visitValueDefinition (pcmp) ≜
        (   dcl str : char*;
```

```
            printNodeField(pcmp.getAccess()) ;
            str := result;
            printNodeField(pcmp.getShape()) ;
            str := str ↷ result ↷ "; " ↷ nl;
            result := str
        );
public
    visitAccessDefinition : IOmlAccessDefinition ⟶ᵒ ()
    visitAccessDefinition (pcmp) ≜
        (   dcl str : char* := "";
            if pcmp.getStaticAccess()
            then str := "  static  ";
            printNodeField(pcmp.getScope()) ;
            str := str ↷ result ↷ "  ";
            result := str
        );
public
    visitScope : IOmlScope ⟶ᵒ ()
    visitScope (pNode) ≜
        (   cases pNode.getValue():
              (OmlScopeQuotes`IQPUBLIC) → result := "public",
              (OmlScopeQuotes`IQPRIVATE),
              (OmlScopeQuotes`IQDEFAULT) → result := "private",
              (OmlScopeQuotes`IQPROTECTED) → result := "protected",
              others → error
            end
        );
public
    visitValueShape : IOmlValueShape ⟶ᵒ ()
    visitValueShape (pcmp) ≜
        (   dcl str : char*;
            printNodeField(pcmp.getPattern()) ;
            str := result ↷ "  ";
            if pcmp.hasType()
            then (   printNodeField(pcmp.getType()) ;
                     str := str ↷ " : " ↷ result ↷ "  "
                 )
            else result := "";
            printNodeField(pcmp.getExpression()) ;
            str := str ↷ " = " ↷ result ↷ "  ";
            result := str
        );
public
    visitPattern : IOmlPattern ⟶ᵒ ()
    visitPattern (pNode) ≜ pNode.
        accept(self) ;
public
    visitExpression : IOmlExpression ⟶ᵒ ()
    visitExpression (pNode) ≜ pNode.
        accept(self) ;
public
    visitBinaryExpression : IOmlBinaryExpression ⟶ᵒ ()
    visitBinaryExpression (pcmp) ≜
        (   dcl str : char* := "";
```

```
        printNodeField(pcmp.getLhsExpression());
        str := str ↷ result;
        printNodeField(pcmp.getOperator());
        str := str ↷ result;
        printNodeField(pcmp.getRhsExpression());
        str := str ↷ result ↷ nl;
        result := str
    );
public
    visitUnaryExpression : IOmlUnaryExpression -o-> ()
    visitUnaryExpression(pcmp) ≜
    (   dcl str : char* := "";
        printNodeField(pcmp.getOperator());
        str := str ↷ result;
        printNodeField(pcmp.getExpression());
        str := str ↷ result ↷ nl;
        result := str
    );
public
```

```
visitBinaryOperator : IOmlBinaryOperator ==o> ()
visitBinaryOperator (pNode) ==
  result := cases pNode.getValue () :
              (OmlBinaryOperatorQuotes`IQMODIFY) -> " ++",
              (OmlBinaryOperatorQuotes`IQGE) -> ">= ",
              (OmlBinaryOperatorQuotes`IQLT) -> "<",
              (OmlBinaryOperatorQuotes`IQPSUBSET) -> " psubset ",
              (OmlBinaryOperatorQuotes`IQMOD) -> " mod ",
              (OmlBinaryOperatorQuotes`IQMAPDOMRESBY) -> "<-: ",
              (OmlBinaryOperatorQuotes`IQINTER) -> " inter ",
              (OmlBinaryOperatorQuotes`IQCOMP) -> " comp ",
              (OmlBinaryOperatorQuotes`IQMINUS) -> "-",
              (OmlBinaryOperatorQuotes`IQREM) -> " rem ",
              (OmlBinaryOperatorQuotes`IQAND) -> " and ",
              (OmlBinaryOperatorQuotes`IQUNION) -> " union ",
              (OmlBinaryOperatorQuotes`IQINSET) -> " in set ",
              (OmlBinaryOperatorQuotes`IQEQUIV) -> "<=>",
              (OmlBinaryOperatorQuotes`IQMAPRNGRESTO) -> " :>",
              (OmlBinaryOperatorQuotes`IQITERATE) -> " **",
              (OmlBinaryOperatorQuotes`IQSUBSET) -> " subset ",
              (OmlBinaryOperatorQuotes`IQMAPRNGRESBY) -> " :->",
              (OmlBinaryOperatorQuotes`IQTUPSEL) -> "#.",
              (OmlBinaryOperatorQuotes`IQNOTINSET) -> " not in set ",
              (OmlBinaryOperatorQuotes`IQMULTIPLY) -> " * ",
              (OmlBinaryOperatorQuotes`IQIMPLY) -> " =>",
              (OmlBinaryOperatorQuotes`IQOR) -> " or ",
              (OmlBinaryOperatorQuotes`IQGT) -> ">",
              (OmlBinaryOperatorQuotes`IQPLUS) -> " + ",
              (OmlBinaryOperatorQuotes`IQMUNION) -> " munion ",
              (OmlBinaryOperatorQuotes`IQMAPDOMRESTO) -> "<: ",
              (OmlBinaryOperatorQuotes`IQEQ) -> " = ",
              (OmlBinaryOperatorQuotes`IQDIV) -> " div ",
              (OmlBinaryOperatorQuotes`IQDIFFERENCE) -> "-",
              (OmlBinaryOperatorQuotes`IQCONC) -> "^",
              (OmlBinaryOperatorQuotes`IQLE) -> "<= ",
              (OmlBinaryOperatorQuotes`IQDIVIDE) -> "\\",
              (OmlBinaryOperatorQuotes`IQNE) -> "<>"
            end;
public
```

```
        visitUnaryOperator : IOmlUnaryOperator ==> ()
        visitUnaryOperator (pNode) ==
          result := cases pNode.getValue () :
                      (OmlUnaryOperatorQuotes`IQABS) -> "abs ",
                      (OmlUnaryOperatorQuotes`IQINVERSE) -> "inverse ",
                      (OmlUnaryOperatorQuotes`IQHD) -> "hd ",
                      (OmlUnaryOperatorQuotes`IQELEMS) -> "elems ",
                      (OmlUnaryOperatorQuotes`IQINDS) -> "inds ",
                      (OmlUnaryOperatorQuotes`IQTL) -> "tl ",
                      (OmlUnaryOperatorQuotes`IQCARD) -> "card ",
                      (OmlUnaryOperatorQuotes`IQDUNION) -> "dunion ",
                      (OmlUnaryOperatorQuotes`IQPOWER) -> "power ",
                      (OmlUnaryOperatorQuotes`IQLEN) -> "len ",
                      (OmlUnaryOperatorQuotes`IQPLUS) -> " + ",
                      (OmlUnaryOperatorQuotes`IQDOM) -> "dom ",
                      (OmlUnaryOperatorQuotes`IQDMERGE) -> "merge ",
                      (OmlUnaryOperatorQuotes`IQDINTER) -> "dinter ",
                      (OmlUnaryOperatorQuotes`IQNOT) -> "not ",
                      (OmlUnaryOperatorQuotes`IQMINUS) -> "-",
                      (OmlUnaryOperatorQuotes`IQRNG) -> "rng ",
                      (OmlUnaryOperatorQuotes`IQFLOOR) -> "floor ",
                      (OmlUnaryOperatorQuotes`IQDCONC) -> "conc "
                    end;
public
        visitSetEnumeration : IOmlSetEnumeration ==> ()
        visitSetEnumeration (pcmp) ==
          (   dcl str : char* := "{";
              printSeqofField(pcmp.getExpressionList ()) ;
              str := str ^ result ^ "}";
              result := str
          );
public
        visitLetExpression : IOmlLetExpression ==> ()
        visitLetExpression (pcmp) ==
          (   dcl str : char* := "let ";
              printSeqofField(pcmp.getDefinitionList ()) ;
              str := str ^ result ^ nl ^ "  in ";
              printNodeField(pcmp.getExpression ()) ;
              str := str ^ result ^ nl;
              result := str
          );
public
        visitFieldSelect : IOmlFieldSelect ==> ()
        visitFieldSelect (pcmp) ==
          (   dcl str : char* := "";
              printNodeField(pcmp.getExpression ()) ;
              str := str ^ result ^ ".";
              printNodeField(pcmp.getName ()) ;
              str := str ^ result;
              result := str
          );
public
```

```
        visitApplyExpression : IOmlApplyExpression ==> ()
        visitApplyExpression (pcmp) ==
          (   dcl str : char* := "";
              printNodeField(pcmp.getExpression ());
              str := str ^ result ^ "(";
              printSeqofField(pcmp.getExpressionList ());
              str := str ^ result ^ ")";
              result := str
          );
public
        visitTupleConstructor : IOmlTupleConstructor ==> ()
        visitTupleConstructor (pcmp) ==
          (   dcl str : char* := "mk_(";
              printSeqofField(pcmp.getExpressionList ());
              str := str ^ result ^ ")";
              result := str
          );
public
        visitRecordConstructor : IOmlRecordConstructor ==> ()
        visitRecordConstructor (pcmp) ==
          (   dcl str : char* := "mk_";
              printNodeField(pcmp.getName ());
              str := str ^ result ^ "(";
              printSeqofField(pcmp.getExpressionList ());
              str := str ^ result ^ ")";
              result := str
          );
public
        visitTokenExpression : IOmlTokenExpression ==> ()
        visitTokenExpression (pcmp) ==
          (   dcl str : char* := "mk_token(";
              printNodeField(pcmp.getExpression ());
              str := str ^ result ^ ")";
              result := str
          );
public
        visitLiteral : IOmlLiteral ==> ()
        visitLiteral (pNode) == pNode.
          accept(self);
public
        visitType : IOmlType ==> ()
        visitType (pNode) == pNode.
          accept(self);
public
        visitPatternIdentifier : IOmlPatternIdentifier ==> ()
        visitPatternIdentifier (pcmp) ==
          (   dcl str : char* := pcmp.getIdentifier () ^ "  ";
              result := str
          );
public
        visitSymbolicLiteralExpression : IOmlSymbolicLiteralExpression ==> ()
        visitSymbolicLiteralExpression (pcmp) ==
          (   dcl str : char* := "";
```

```
            printNodeField(pcmp.getLiteral())
        );
public
    visitTextLiteral : IOmlTextLiteral ->o ()
    visitTextLiteral (pcmp) △
        (   dcl str : char* := pcmp.getVal();
            result := "\"" ↷ str ↷ "\""
        );
public
    visitCharacterLiteral : IOmlCharacterLiteral ->o ()
    visitCharacterLiteral (pcmp) △
        (   dcl str : char* := "'";
            printCharField(pcmp.getVal()) ;
            str := str ↷ result ↷ "'";
            result := str
        );
public
    visitSeq0Type : IOmlSeq0Type ->o ()
    visitSeq0Type (pcmp) △
        (   dcl str : char* := "seq of ";
            printNodeField(pcmp.getType()) ;
            str := str ↷ result;
            result := str
        );
public
    visitCharType : IOmlCharType ->o ()
    visitCharType (-) △
        (   dcl str : char* := "char";
            result := str
        );
public
    visitInstanceVariableDefinitions : IOmlInstanceVariableDefinitions ->o ()
    visitInstanceVariableDefinitions (pcmp) △
        (   dcl str : char* := nl ↷ "instance variables" ↷ nl ↷ nl;
            for db in pcmp.getVariablesList()
            do (    printNodeField(db) ;
                    str := str ↷ result ↷ nl
               ) ;
            if len pcmp.getVariablesList() = 0
            then result := ""
            else result := str
        );
public
    visitInstanceVariable : IOmlInstanceVariable ->o ()
    visitInstanceVariable (pcmp) △
        (   dcl str : char* := "";
            printNodeField(pcmp.getAccess()) ;
            str := str ↷ result;
            printNodeField(pcmp.getAssignmentDefinition()) ;
            str := str ↷ result;
            result := str
        );
public
```

```
        visitAssignmentDefinition : IOmlAssignmentDefinition ^o→ ()
        visitAssignmentDefinition (pcmp) △
          (   dcl str : char* := "";
              str := str ↷ pcmp.getIdentifier ();
              printNodeField(pcmp.getType ()) ;
              str := str ↷ "  :  " ↷ result;
              if pcmp.hasExpression ()
              then (   printNodeField(pcmp.getExpression ()) ;
                       str := str ↷ "  :=  "
                   )
              else result := "";
              str := str ↷ result ↷ "; ";
              result := str
          );
public
        visitTypeName : IOmlTypeName ^o→ ()
        visitTypeName (pcmp) △
          (   printNodeField(pcmp.getName ())
          );
public
        visitName : IOmlName ^o→ ()
        visitName (pcmp) △
          (   dcl str : char* := "";
              if pcmp.hasClassIdentifier ()
              then str := str ↷ pcmp.getClassIdentifier () ↷ "`";
              str := str ↷ pcmp.getIdentifier ();
              result := str
          );
public
        visitIntType : IOmlIntType ^o→ ()
        visitIntType (-) △
          (   dcl str : char* := "int";
              result := str
          );
public
        visitNatType : IOmlNatType ^o→ ()
        visitNatType (-) △
          (   dcl str : char* := "nat";
              result := str
          );
public
        visitNat1Type : IOmlNat1Type ^o→ ()
        visitNat1Type (-) △
          (   dcl str : char* := "nat1";
              result := str
          );
public
        visitBoolType : IOmlBoolType ^o→ ()
        visitBoolType (-) △
          (   dcl str : char* := "bool";
              result := str
          );
public
```

```
visitSeq1Type : IOmlSeq1Type ==> ()
visitSeq1Type (pcmp) ==
  (   dcl str : char* := "seq1 of ";
      printNodeField(pcmp.getType ()) ;
      str := str ^ result;
      result := str
  );
public
visitRealType : IOmlRealType ==> ()
visitRealType (-) ==
  (   dcl str : char* := "real";
      result := str
  );
public
visitSetType : IOmlSetType ==> ()
visitSetType (pcmp) ==
  (   dcl str : char* := "set of ";
      printNodeField(pcmp.getType ()) ;
      str := str ^ result;
      result := str
  );
public
visitTypeDefinitions : IOmlTypeDefinitions ==> ()
visitTypeDefinitions (pcmp) ==
  (   dcl str : char* := nl ^ "types" ^ nl ^ nl;
      for db in pcmp.getTypeList ()
      do (   printNodeField(db) ;
             str := str ^ result ^ nl
         ) ;
      result := str
  );
public
visitTypeDefinition : IOmlTypeDefinition ==> ()
visitTypeDefinition (pcmp) ==
  (   dcl str : char* := "";
      printNodeField(pcmp.getAccess ()) ;
      str := str ^ result;
      printNodeField(pcmp.getShape ()) ;
      str := str ^ result ^ ";";
      result := str
  );
public
visitSimpleType : IOmlSimpleType ==> ()
visitSimpleType (pcmp) ==
  (   dcl str : char* := pcmp.getIdentifier ();
      printNodeField(pcmp.getType ()) ;
      result := str ^ "  =  " ^ result
  );
public
visitEmptyType : IOmlEmptyType ==> ()
visitEmptyType (-) ==
  (   dcl str : char* := "()";
      result := str
  );
```

```
public
        visitNewExpression : IOmlNewExpression ^o→ ()
        visitNewExpression (pcmp) ≜
          (   dcl str : char* := "new ";
              printNodeField(pcmp.getName ()) ;
              str := str ↷ result ↷ "(";
              printSeqofField(pcmp.getExpressionList ()) ;
              str := str ↷ result ↷ ")" ↷ nl;
              result := str
          );
public
        visitIfExpression : IOmlIfExpression ^o→ ()
        visitIfExpression (pcmp) ≜
          (   dcl str : char* := "if ";
              printNodeField(pcmp.getIfExpression ()) ;
              str := str ↷ result ↷ " then " ↷ nl;
              printNodeField(pcmp.getThenExpression ()) ;
              str := str ↷ result ↷ nl;
              printSeqofField(pcmp.getElseifExpressionList ()) ;
              str := str ↷ result ↷ " else ";
              printNodeField(pcmp.getElseExpression ()) ;
              str := str ↷ result;
              result := str
          );
public
        visitElseIfExpression : IOmlElseIfExpression ^o→ ()
        visitElseIfExpression (pcmp) ≜
          (   dcl str : char* := " elseif ";
              printNodeField(pcmp.getElseifExpression ()) ;
              str := str ↷ result ↷ " then ";
              printNodeField(pcmp.getThenExpression ()) ;
              str := str ↷ result ↷ nl;
              result := str
          );
public
        visitBracketedExpression : IOmlBracketedExpression ^o→ ()
        visitBracketedExpression (pcmp) ≜
          (   dcl str : char* := "(";
              printNodeField(pcmp.getExpression ()) ;
              str := str ↷ result;
              str := str ↷ ")";
              result := str
          );
public
        visitNumericLiteral : IOmlNumericLiteral ^o→ ()
        visitNumericLiteral (pcmp) ≜
          (   dcl str : char* := "";
              printNatField(pcmp.getVal ()) ;
              str := result;
              result := str
          );
public
```

```
        visitRealLiteral : IOmlRealLiteral ==o==> ()
        visitRealLiteral (pcmp) ≜
          (   dcl str : char* := "";
              printRealField(pcmp.getVal ()) ;
              str := result;
              result := str
          );
public
        visitQuoteLiteral : IOmlQuoteLiteral ==o==> ()
        visitQuoteLiteral (pcmp) ≜
          (   dcl str : char* := "<";
              printStringField(pcmp.getVal ()) ;
              str := str ↷ result (2, ..., len result − 1) ↷ ">";
              result := str
          );
public
        visitBooleanLiteral : IOmlBooleanLiteral ==o==> ()
        visitBooleanLiteral (pcmp) ≜
          (   dcl str : char* := "";
              printBoolField(pcmp.getVal ()) ;
              result := "  " ↷ result ↷ "  "
          );
public
        visitNilLiteral : IOmlNilLiteral ==o==> ()
        visitNilLiteral (pcmp) ≜
          result := "  nil  ";
public
        visitOperationDefinitions : IOmlOperationDefinitions ==o==> ()
        visitOperationDefinitions (pcmp) ≜
          (   dcl str : char* := nl ↷ "operations" ↷ nl ↷ nl;
              for db in pcmp.getOperationList ()
              do (   printNodeField(db) ;
                     str := str ↷ result ↷ nl
                 ) ;
              if len pcmp.getOperationList () > 0
              then result := str
              else result := ""
          );
public
        visitOperationDefinition : IOmlOperationDefinition ==o==> ()
        visitOperationDefinition (pcmp) ≜
          (   dcl str : char* := "";
              printNodeField(pcmp.getAccess ()) ;
              str := str ↷ result;
              printNodeField(pcmp.getShape ()) ;
              str := str ↷ result;
              result := str
          );
public
        visitExplicitOperation : IOmlExplicitOperation ==o==> ()
        visitExplicitOperation (pcmp) ≜
          (   dcl str : char* := pcmp.getIdentifier () ↷ "  :  ";
```

```
            printNodeField(pcmp.getType());
            str := str ↷ result;
            str := str ↷ nl ↷ pcmp.getIdentifier() ↷ "(";
            if len pcmp.getParameterList() > 0
            then (  for db in pcmp.getParameterList()
                    do (  printNodeField(db);
                          str := str ↷ result ↷ ",  "
                       );
                    str := str(1, ..., len str − 2)
                 );
            str := str ↷ ")  ==  ";
            printNodeField(pcmp.getBody());
            str := str ↷ result ↷ "; " ↷ nl;
            result := str
        );
public
    visitOperationType : IOmlOperationType →o ()
    visitOperationType(pcmp) ≜
        (  dcl str : char* := "";
            printNodeField(pcmp.getDomType());
            str := str ↷ result ↷ "  ==>  ";
            printNodeField(pcmp.getRngType());
            str := str ↷ result;
            result := str
        );
public
    visitOperationBody : IOmlOperationBody →o ()
    visitOperationBody(pcmp) ≜
        (  dcl str : char* := "(";
            if pcmp.getNotYetSpecified()
            then (  result := "is not yet specified";
                    return
                 )
            else (  if pcmp.hasStatement()
                    then printNodeField(pcmp.getStatement())
                    else result := "";
                    str := str ↷ result
                 );
            if pcmp.getSubclassResponsibility()
            then (  result := "sub class responsibility";
                    return
                 );
            str := str ↷ ")";
            result := str
        );
public
    visitSkipStatement : IOmlSkipStatement →o ()
    visitSkipStatement(-) ≜
        (  dcl str : char* := "skip";
            result := str
        );
public
```

```
        visitParameter : IOmlParameter ==o==> ()
        visitParameter (pcmp) ≜
          (   dcl str : char* := "";
              for db in pcmp.getPatternList ()
              do (   printNodeField(db) ;
                     str := str ⌢ result ⌢ ",  "
                 ) ;
              str := str (1, ..., len str − 2);
              result := str
          );
public
        visitFunctionDefinitions : IOmlFunctionDefinitions ==o==> ()
        visitFunctionDefinitions (pcmp) ≜
          (   dcl str : char* := nl ⌢ "functions" ⌢ nl ⌢ nl;
              for db in pcmp.getFunctionList ()
              do (   printNodeField(db) ;
                     str := str ⌢ result ⌢ nl
                 ) ;
              result := str
          );
public
        visitFunctionDefinition : IOmlFunctionDefinition ==o==> ()
        visitFunctionDefinition (pcmp) ≜
          (   dcl str : char* := "";
              printNodeField(pcmp.getAccess ()) ;
              str := str ⌢ result;
              printNodeField(pcmp.getShape ()) ;
              str := str ⌢ result;
              result := str
          );
public
        visitExplicitFunction : IOmlExplicitFunction ==o==> ()
        visitExplicitFunction (pcmp) ≜
          (   dcl str : char* := pcmp.getIdentifier () ⌢ "  :  ";
              if len pcmp.getTypeVariableList () > 0
              then (   for db in pcmp.getTypeVariableList ()
                       do (   printNodeField(db) ;
                              str := str ⌢ result ⌢ " * "
                          ) ;
                       str := str (1, ..., len str − 2)
                   ) ;
              printNodeField(pcmp.getType ()) ;
              str := str ⌢ result;
              str := str ⌢ nl ⌢ pcmp.getIdentifier () ⌢ "(";
              for db in pcmp.getParameterList ()
              do (   printNodeField(db) ;
                     str := str ⌢ result ⌢ ",  "
                 ) ;
              str := str (1, ..., len str − 2);
              str := str ⌢ ")  ==  is not yet specified;";
              result := str
          );
public
```

```
        visitPartialFunctionType : IOmlPartialFunctionType ^o^→ ()
        visitPartialFunctionType (pcmp) ≜
          (    dcl str : char* := " ";
               printNodeField(pcmp.getDomType ()) ;
               str := str ↷ result ↷ "  ->  ";
               printNodeField(pcmp.getRngType ()) ;
               str := str ↷ result;
               result := str
          );
public
        visitUnionType : IOmlUnionType ^o^→ ()
        visitUnionType (pcmp) ≜
          (    dcl str : char* := " ";
               pcmp.getLhsType    ( ) .accept(self) ;
               str := str ↷ result;
               pcmp.getRhsType    ( ) .accept(self) ;
               str := str ↷ "   |   " ↷ result;
               result := str
          );
public
        visitProductType : IOmlProductType ^o^→ ()
        visitProductType (pcmp) ≜
          (    dcl str : char* := " ";
               pcmp.getLhsType    ( ) .accept(self) ;
               str := str ↷ result;
               pcmp.getRhsType    ( ) .accept(self) ;
               str := str ↷ "   *   " ↷ result;
               result := str
          );
public
        visitTraceDefinitions : IOmlTraceDefinitions ^o^→ ()
        visitTraceDefinitions (pcmp) ≜
          (    dcl str : char* := nl ↷ "traces" ↷ nl ↷ nl;
               for db in pcmp.getTraces ()
               do (    printNodeField(db) ;
                       str := str ↷ result ↷ nl
                  ) ;
               result := str
          );
public
        visitNamedTrace : IOmlNamedTrace ^o^→ ()
        visitNamedTrace (pcmp) ≜
          (    dcl str : char* := "    ";
               str := str ↷ pcmp.getName () ↷ "   :   ";
               printNodeField(pcmp.getDefs ()) ;
               str := str ↷ result;
               result := str
          );
public
        visitTraceDefinition : IOmlTraceDefinition ^o^→ ()
        visitTraceDefinition (pNode) ≜ pNode.
          accept(self) ;
public
```

```
        visitTraceDefinitionItem : IOmlTraceDefinitionItem ==o=> ()
        visitTraceDefinitionItem (pcmp) ==
          (   dcl str : char* := "";
              printSeqofField(pcmp.getBind ()) ;
              str := str ^~ result;
              printNodeField(pcmp.getTest ()) ;
              str := str ^~ result;
              if pcmp.hasRegexpr ()
              then printNodeField(pcmp.getRegexpr ())
              else result := "";
              str := str ^~ result;
              result := str
          );
public
        visitTraceBinding : IOmlTraceBinding ==o=> ()
        visitTraceBinding (pNode) == pNode.
          accept(self) ;
public
        visitTraceLetBinding : IOmlTraceLetBinding ==o=> ()
        visitTraceLetBinding (pcmp) ==
          (   dcl str : char* := "";
              printSeqofField(pcmp.getDefinitionList ()) ;
              str := str ^~ result;
              result := str
          );
public
        visitTraceBracketedDefinition : IOmlTraceBracketedDefinition ==o=> ()
        visitTraceBracketedDefinition (pcmp) ==
          (   dcl str : char* := "(";
              printNodeField(pcmp.getDefinition ()) ;
              str := str ^~ result ^~ ")";
              result := str
          );
public
        visitTraceMethodApply : IOmlTraceMethodApply ==o=> ()
        visitTraceMethodApply (pcmp) ==
          (   dcl str : char* := "";
              str := str ^~ pcmp.getVariableName () ^~ ".";
              str := str ^~ pcmp.getMethodName () ^~ "(";
              printSeqofField(pcmp.getArgs ()) ;
              str := str ^~ result ^~ ")";
              result := str
          );
public
        visitTraceCoreDefinition : IOmlTraceCoreDefinition ==o=> ()
        visitTraceCoreDefinition (pNode) == pNode.
          accept(self) ;
public
        visitTraceRepeatPattern : IOmlTraceRepeatPattern ==o=> ()
        visitTraceRepeatPattern (pNode) == pNode.
          accept(self) ;
public
```

```
        visitTraceZeroOrMore : IOmlTraceZeroOrMore ->o ()
        visitTraceZeroOrMore (-) ≜
          (   dcl str : char* := " * ";
              result := str
          );
public
        visitTraceOneOrMore : IOmlTraceOneOrMore ->o ()
        visitTraceOneOrMore (-) ≜
          (   dcl str : char* := " + ";
              result := str
          );
public
        visitTraceZeroOrOne : IOmlTraceZeroOrOne ->o ()
        visitTraceZeroOrOne (-) ≜
          (   dcl str : char* := "?";
              result := str
          );
public
        visitTraceRange : IOmlTraceRange ->o ()
        visitTraceRange (pcmp) ≜
          (   dcl str : char* := "{";
              printNodeField(pcmp.getLower ()) ;
              str := str ^ result;
              if pcmp.hasUpper ()
              then (   printNodeField(pcmp.getUpper ()) ;
                       str := str ^ ", " ^ result
                   ) ;
              str := str ^ "}";
              result := str
          );
public
        visitTraceChoiceDefinition : IOmlTraceChoiceDefinition ->o ()
        visitTraceChoiceDefinition (pcmp) ≜
          (   dcl str : char* := "",
                  count : ℕ := 1;
              for db in pcmp.getDefs ()
              do (   printNodeField(db) ;
                     if len pcmp.getDefs () = count
                     then str := str ^ result
                     else str := str ^ result ^ "   |   ";
                     count := count + 1
                 );
              result := str
          );
public
        visitTraceSequenceDefinition : IOmlTraceSequenceDefinition ->o ()
        visitTraceSequenceDefinition (pcmp) ≜
          (   dcl str : char* := "",
                  count : ℕ := 1;
              for db in pcmp.getDefs ()
              do (   printNodeField(db) ;
                     if len pcmp.getDefs () = count
                     then str := str ^ result
```

```
                    else str := str ⌢ result ⌢ "  ;  ";
                    count := count + 1
                );
            result := str
        );
public
    visitLexem : IOmlLexem ==> ()
    visitLexem (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitOldName : IOmlOldName ==> ()
    visitOldName (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitSeqConcPattern : IOmlSeqConcPattern ==> ()
    visitSeqConcPattern (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitPeriodicThread : IOmlPeriodicThread ==> ()
    visitPeriodicThread (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitCallStatement : IOmlCallStatement ==> ()
    visitCallStatement (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitIsofclassExpression : IOmlIsofclassExpression ==> ()
    visitIsofclassExpression (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitIndexForLoop : IOmlIndexForLoop ==> ()
    visitIndexForLoop (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitReqExpression : IOmlReqExpression ==> ()
    visitReqExpression (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitPermissionPredicate : IOmlPermissionPredicate ==> ()
    visitPermissionPredicate (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitProcedureThread : IOmlProcedureThread ==> ()
    visitProcedureThread (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitMapEnumeration : IOmlMapEnumeration ==> ()
    visitMapEnumeration (-) ≜
        result := "NOT  YET  SUPPORTED";
public
    visitAtomicStatement : IOmlAtomicStatement ==> ()
    visitAtomicStatement (-) ≜
        result := "NOT  YET  SUPPORTED";
public
```

```
        visitFieldReference : IOmlFieldReference ==> ()
        visitFieldReference (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitBlockStatement : IOmlBlockStatement ==> ()
        visitBlockStatement (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitInjectiveMapType : IOmlInjectiveMapType ==> ()
        visitInjectiveMapType (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitMatchValue : IOmlMatchValue ==> ()
        visitMatchValue (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitOperationTrailer : IOmlOperationTrailer ==> ()
        visitOperationTrailer (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitTypeBind : IOmlTypeBind ==> ()
        visitTypeBind (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitRecordModifier : IOmlRecordModifier ==> ()
        visitRecordModifier (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitObjectDesignatorExpression : IOmlObjectDesignatorExpression ==> ()
        visitObjectDesignatorExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitIdentifierTypePair : IOmlIdentifierTypePair ==> ()
        visitIdentifierTypePair (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitPatternBindExpression : IOmlPatternBindExpression ==> ()
        visitPatternBindExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitTrapDefinition : IOmlTrapDefinition ==> ()
        visitTrapDefinition (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSelfExpression : IOmlSelfExpression ==> ()
        visitSelfExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitRecursiveTrapStatement : IOmlRecursiveTrapStatement ==> ()
        visitRecursiveTrapStatement (-) ≜
          result := "NOT YET SUPPORTED";
public
```

```
        visitWhileLoop : IOmlWhileLoop ==> ()
        visitWhileLoop (-) ==
          result := "NOT YET SUPPORTED";
public
        visitDefStatement : IOmlDefStatement ==> ()
        visitDefStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitSetForLoop : IOmlSetForLoop ==> ()
        visitSetForLoop (-) ==
          result := "NOT YET SUPPORTED";
public
        visitDefExpression : IOmlDefExpression ==> ()
        visitDefExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitDurationStatement : IOmlDurationStatement ==> ()
        visitDurationStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitImplicitOperation : IOmlImplicitOperation ==> ()
        visitImplicitOperation (-) ==
          result := "NOT YET SUPPORTED";
public
        visitTypeVariable : IOmlTypeVariable ==> ()
        visitTypeVariable (-) ==
          result := "NOT YET SUPPORTED";
public
        visitCompositeType : IOmlCompositeType ==> ()
        visitCompositeType (-) ==
          result := "NOT YET SUPPORTED";
public
        visitCasesStatementAlternative : IOmlCasesStatementAlternative ==> ()
        visitCasesStatementAlternative (-) ==
          result := "NOT YET SUPPORTED";
public
        visitActiveExpression : IOmlActiveExpression ==> ()
        visitActiveExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitForAllExpression : IOmlForAllExpression ==> ()
        visitForAllExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitCasesExpression : IOmlCasesExpression ==> ()
        visitCasesExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitCasesStatement : IOmlCasesStatement ==> ()
        visitCasesStatement (-) ==
          result := "NOT YET SUPPORTED";
public
```

```
        visitErrorStatement : IOmlErrorStatement ==> ()
        visitErrorStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitImplicitFunction : IOmlImplicitFunction ==> ()
        visitImplicitFunction (-) ==
          result := "NOT YET SUPPORTED";
public
        visitSamebaseclassExpression : IOmlSamebaseclassExpression ==> ()
        visitSamebaseclassExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitComplexType : IOmlComplexType ==> ()
        visitComplexType (-) ==
          result := "NOT YET SUPPORTED";
public
        visitExternals : IOmlExternals ==> ()
        visitExternals (-) ==
          result := "NOT YET SUPPORTED";
public
        visitSubsequenceExpression : IOmlSubsequenceExpression ==> ()
        visitSubsequenceExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitElseIfStatement : IOmlElseIfStatement ==> ()
        visitElseIfStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitGeneralMapType : IOmlGeneralMapType ==> ()
        visitGeneralMapType (-) ==
          result := "NOT YET SUPPORTED";
public
        visitSpecificationStatement : IOmlSpecificationStatement ==> ()
        visitSpecificationStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitTuplePattern : IOmlTuplePattern ==> ()
        visitTuplePattern (-) ==
          result := "NOT YET SUPPORTED";
public
        visitField : IOmlField ==> ()
        visitField (-) ==
          result := "NOT YET SUPPORTED";
public
        visitTokenType : IOmlTokenType ==> ()
        visitTokenType (-) ==
          result := "NOT YET SUPPORTED";
public
        visitSameclassExpression : IOmlSameclassExpression ==> ()
        visitSameclassExpression (-) ==
          result := "NOT YET SUPPORTED";
public
```

```
        visitExitStatement : IOmlExitStatement ^o^→ ()
        visitExitStatement (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitExistsExpression : IOmlExistsExpression ^o^→ ()
        visitExistsExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitFunctionTypeInstantiation : IOmlFunctionTypeInstantiation ^o^→ ()
        visitFunctionTypeInstantiation (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSequenceEnumeration : IOmlSequenceEnumeration ^o^→ ()
        visitSequenceEnumeration (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitObjectApply : IOmlObjectApply ^o^→ ()
        visitObjectApply (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSetUnionPattern : IOmlSetUnionPattern ^o^→ ()
        visitSetUnionPattern (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitStartStatement : IOmlStartStatement ^o^→ ()
        visitStartStatement (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitError : IOmlError ^o^→ ()
        visitError (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitIfStatement : IOmlIfStatement ^o^→ ()
        visitIfStatement (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitLetBeExpression : IOmlLetBeExpression ^o^→ ()
        visitLetBeExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitTotalFunctionType : IOmlTotalFunctionType ^o^→ ()
        visitTotalFunctionType (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSporadicThread : IOmlSporadicThread ^o^→ ()
        visitSporadicThread (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitMapComprehension : IOmlMapComprehension ^o^→ ()
        visitMapComprehension (-) ≜
          result := "NOT YET SUPPORTED";
public
```

```
        visitSetBind : IOmlSetBind ^o→ ()
        visitSetBind (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitNondeterministicStatement : IOmlNondeterministicStatement ^o→ ()
        visitNondeterministicStatement (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSymbolicLiteralPattern : IOmlSymbolicLiteralPattern ^o→ ()
        visitSymbolicLiteralPattern (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitOptionalType : IOmlOptionalType ^o→ ()
        visitOptionalType (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitMutexAllPredicate : IOmlMutexAllPredicate ^o→ ()
        visitMutexAllPredicate (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSequenceComprehension : IOmlSequenceComprehension ^o→ ()
        visitSequenceComprehension (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitEqualsDefinition : IOmlEqualsDefinition ^o→ ()
        visitEqualsDefinition (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitMaplet : IOmlMaplet ^o→ ()
        visitMaplet (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitExistsUniqueExpression : IOmlExistsUniqueExpression ^o→ ()
        visitExistsUniqueExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitThreadIdExpression : IOmlThreadIdExpression ^o→ ()
        visitThreadIdExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitExtendedExplicitOperation : IOmlExtendedExplicitOperation ^o→ ()
        visitExtendedExplicitOperation (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSetComprehension : IOmlSetComprehension ^o→ ()
        visitSetComprehension (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitIotaExpression : IOmlIotaExpression ^o→ ()
        visitIotaExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
```

```
        visitReturnStatement : IOmlReturnStatement ==> ()
        visitReturnStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitSetEnumPattern : IOmlSetEnumPattern ==> ()
        visitSetEnumPattern (-) ==
          result := "NOT YET SUPPORTED";
public
        visitTrapStatement : IOmlTrapStatement ==> ()
        visitTrapStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitFunctionTypeSelect : IOmlFunctionTypeSelect ==> ()
        visitFunctionTypeSelect (-) ==
          result := "NOT YET SUPPORTED";
public
        visitClassTypeInstantiation : IOmlClassTypeInstantiation ==> ()
        visitClassTypeInstantiation (-) ==
          result := "NOT YET SUPPORTED";
public
        visitCyclesStatement : IOmlCyclesStatement ==> ()
        visitCyclesStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitPreconditionExpression : IOmlPreconditionExpression ==> ()
        visitPreconditionExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitTraceLetBeBinding : IOmlTraceLetBeBinding ==> ()
        visitTraceLetBeBinding (-) ==
          result := "NOT YET SUPPORTED";
public
        visitVarInformation : IOmlVarInformation ==> ()
        visitVarInformation (-) ==
          result := "NOT YET SUPPORTED";
public
        visitIsofbaseclassExpression : IOmlIsofbaseclassExpression ==> ()
        visitIsofbaseclassExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitLetStatement : IOmlLetStatement ==> ()
        visitLetStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitActExpression : IOmlActExpression ==> ()
        visitActExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitExceptions : IOmlExceptions ==> ()
        visitExceptions (-) ==
          result := "NOT YET SUPPORTED";
public
```

```
        visitIsExpression : IOmlIsExpression ->o ()
        visitIsExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitCasesExpressionAlternative : IOmlCasesExpressionAlternative ->o ()
        visitCasesExpressionAlternative (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitLetBeStatement : IOmlLetBeStatement ->o ()
        visitLetBeStatement (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitMutexPredicate : IOmlMutexPredicate ->o ()
        visitMutexPredicate (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitUndefinedExpression : IOmlUndefinedExpression ->o ()
        visitUndefinedExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitAssignStatement : IOmlAssignStatement ->o ()
        visitAssignStatement (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSequenceForLoop : IOmlSequenceForLoop ->o ()
        visitSequenceForLoop (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitWaitingExpression : IOmlWaitingExpression ->o ()
        visitWaitingExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitInvariant : IOmlInvariant ->o ()
        visitInvariant (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitObjectFieldReference : IOmlObjectFieldReference ->o ()
        visitObjectFieldReference (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitBracketedType : IOmlBracketedType ->o ()
        visitBracketedType (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitFinExpression : IOmlFinExpression ->o ()
        visitFinExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitSetRangeExpression : IOmlSetRangeExpression ->o ()
        visitSetRangeExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
```

```
        visitFunctionBody : IOmlFunctionBody ==> ()
        visitFunctionBody (-) ==
          result := "NOT YET SUPPORTED";
public
        visitLambdaExpression : IOmlLambdaExpression ==> ()
        visitLambdaExpression (-) ==
          result := "NOT YET SUPPORTED";
public
        visitInstanceVariableInvariant : IOmlInstanceVariableInvariant ==> ()
        visitInstanceVariableInvariant (-) ==
          result := "NOT YET SUPPORTED";
public
        visitDontCarePattern : IOmlDontCarePattern ==> ()
        visitDontCarePattern (-) ==
          result := "NOT YET SUPPORTED";
public
        visitSeqEnumPattern : IOmlSeqEnumPattern ==> ()
        visitSeqEnumPattern (-) ==
          result := "NOT YET SUPPORTED";
public
        visitTypelessExplicitFunction : IOmlTypelessExplicitFunction ==> ()
        visitTypelessExplicitFunction (-) ==
          result := "NOT YET SUPPORTED";
public
        visitThreadDefinition : IOmlThreadDefinition ==> ()
        visitThreadDefinition (-) ==
          result := "NOT YET SUPPORTED";
public
        visitAlwaysStatement : IOmlAlwaysStatement ==> ()
        visitAlwaysStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitRecordPattern : IOmlRecordPattern ==> ()
        visitRecordPattern (-) ==
          result := "NOT YET SUPPORTED";
public
        visitQuoteType : IOmlQuoteType ==> ()
        visitQuoteType (-) ==
          result := "NOT YET SUPPORTED";
public
        visitDclStatement : IOmlDclStatement ==> ()
        visitDclStatement (-) ==
          result := "NOT YET SUPPORTED";
public
        visitPatternTypePair : IOmlPatternTypePair ==> ()
        visitPatternTypePair (-) ==
          result := "NOT YET SUPPORTED";
public
        visitMapOrSequenceReference : IOmlMapOrSequenceReference ==> ()
        visitMapOrSequenceReference (-) ==
          result := "NOT YET SUPPORTED";
public
```

```
        visitSynchronizationDefinitions : IOmlSynchronizationDefinitions ==> ()
        visitSynchronizationDefinitions (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitMuExpression : IOmlMuExpression ==> ()
        visitMuExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitFunctionTrailer : IOmlFunctionTrailer ==> ()
        visitFunctionTrailer (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitTimeExpression : IOmlTimeExpression ==> ()
        visitTimeExpression (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitRatType : IOmlRatType ==> ()
        visitRatType (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitStateDesignatorName : IOmlStateDesignatorName ==> ()
        visitStateDesignatorName (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitExtendedExplicitFunction : IOmlExtendedExplicitFunction ==> ()
        visitExtendedExplicitFunction (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitContextInfo : IOmlContextInfo ==> ()
        visitContextInfo (-) ≜
          result := "NOT YET SUPPORTED";
public
        visitMode : IOmlMode ==> ()
        visitMode (-) ≜
          result := "NOT YET SUPPORTED"
end Oml2VppVisitor
```

Test Suite : vdm.tc
Class : Oml2VppVisitor

Name	#Calls	Coverage
Oml2VppVisitor`visitMode	0	0%
Oml2VppVisitor`visitName	1858	√
Oml2VppVisitor`visitNode	0	0%
Oml2VppVisitor`visitType	0	0%
Oml2VppVisitor`printField	1167	58%
Oml2VppVisitor`visitClass	0	0%
Oml2VppVisitor`visitError	0	0%
Oml2VppVisitor`visitField	0	0%
Oml2VppVisitor`visitLexem	0	0%
Oml2VppVisitor`visitScope	0	0%
Oml2VppVisitor`visitMaplet	0	0%
Oml2VppVisitor`visitIntType	0	0%
Oml2VppVisitor`visitLiteral	0	0%

Name	#Calls	Coverage
Oml2VppVisitor‘visitNatType	0	0%
Oml2VppVisitor‘visitOldName	0	0%
Oml2VppVisitor‘visitPattern	0	0%
Oml2VppVisitor‘visitRatType	0	0%
Oml2VppVisitor‘visitSetBind	0	0%
Oml2VppVisitor‘visitSetType	0	0%
Oml2VppVisitor‘printNatField	322	√
Oml2VppVisitor‘visitBoolType	0	0%
Oml2VppVisitor‘visitCharType	0	0%
Oml2VppVisitor‘visitDocument	0	0%
Oml2VppVisitor‘visitNat1Type	0	0%
Oml2VppVisitor‘visitRealType	0	0%
Oml2VppVisitor‘visitSeq0Type	0	0%
Oml2VppVisitor‘visitSeq1Type	0	0%
Oml2VppVisitor‘visitTypeBind	0	0%
Oml2VppVisitor‘visitTypeName	0	0%
Oml2VppVisitor‘printBoolField	24	80%
Oml2VppVisitor‘printCharField	24	√
Oml2VppVisitor‘printNodeField	6282	√
Oml2VppVisitor‘printRealField	454	√
Oml2VppVisitor‘visitEmptyType	0	0%
Oml2VppVisitor‘visitExternals	0	0%
Oml2VppVisitor‘visitInvariant	0	0%
Oml2VppVisitor‘visitParameter	0	0%
Oml2VppVisitor‘visitQuoteType	0	0%
Oml2VppVisitor‘visitTokenType	0	0%
Oml2VppVisitor‘visitUnionType	0	0%
Oml2VppVisitor‘visitWhileLoop	0	0%
Oml2VppVisitor‘printSeqofField	1167	√
Oml2VppVisitor‘visitExceptions	0	0%
Oml2VppVisitor‘visitExpression	715	√
Oml2VppVisitor‘visitMatchValue	0	0%
Oml2VppVisitor‘visitNamedTrace	0	0%
Oml2VppVisitor‘visitNilLiteral	48	√
Oml2VppVisitor‘visitSetForLoop	0	0%
Oml2VppVisitor‘visitSimpleType	0	0%
Oml2VppVisitor‘visitTraceRange	0	0%
Oml2VppVisitor‘visitValueShape	446	69%
Oml2VppVisitor‘printStringField	40	√
Oml2VppVisitor‘visitComplexType	0	0%
Oml2VppVisitor‘visitContextInfo	0	0%
Oml2VppVisitor‘visitFieldSelect	717	√
Oml2VppVisitor‘visitIfStatement	0	0%
Oml2VppVisitor‘visitObjectApply	0	0%
Oml2VppVisitor‘visitProductType	0	0%
Oml2VppVisitor‘visitRealLiteral	454	√
Oml2VppVisitor‘visitTextLiteral	48	√
Oml2VppVisitor‘visitDclStatement	0	0%
Oml2VppVisitor‘visitDefStatement	0	0%
Oml2VppVisitor‘visitFunctionBody	0	0%

Name	#Calls	Coverage
Oml2VppVisitor‘visitIfExpression	24	√
Oml2VppVisitor‘visitIndexForLoop	0	0%
Oml2VppVisitor‘visitIsExpression	0	0%
Oml2VppVisitor‘visitLetStatement	0	0%
Oml2VppVisitor‘visitMuExpression	0	0%
Oml2VppVisitor‘visitOptionalType	0	0%
Oml2VppVisitor‘visitQuoteLiteral	40	√
Oml2VppVisitor‘visitTraceBinding	0	0%
Oml2VppVisitor‘visitTuplePattern	0	0%
Oml2VppVisitor‘visitTypeVariable	0	0%
Oml2VppVisitor‘visitActExpression	0	0%
Oml2VppVisitor‘visitBracketedType	0	0%
Oml2VppVisitor‘visitCallStatement	0	0%
Oml2VppVisitor‘visitCompositeType	0	0%
Oml2VppVisitor‘visitDefExpression	0	0%
Oml2VppVisitor‘visitExitStatement	0	0%
Oml2VppVisitor‘visitFinExpression	0	0%
Oml2VppVisitor‘visitLetExpression	374	√
Oml2VppVisitor‘visitNewExpression	16	√
Oml2VppVisitor‘visitOperationBody	0	0%
Oml2VppVisitor‘visitOperationType	0	0%
Oml2VppVisitor‘visitRecordPattern	0	0%
Oml2VppVisitor‘visitReqExpression	0	0%
Oml2VppVisitor‘visitSkipStatement	0	0%
Oml2VppVisitor‘visitTrapStatement	0	0%
Oml2VppVisitor‘visitUnaryOperator	0	0%
Oml2VppVisitor‘visitBinaryOperator	195	56%
Oml2VppVisitor‘visitBlockStatement	0	0%
Oml2VppVisitor‘visitBooleanLiteral	24	√
Oml2VppVisitor‘visitCasesStatement	0	0%
Oml2VppVisitor‘visitErrorStatement	0	0%
Oml2VppVisitor‘visitFieldReference	0	0%
Oml2VppVisitor‘visitGeneralMapType	0	0%
Oml2VppVisitor‘visitIotaExpression	0	0%
Oml2VppVisitor‘visitLetBeStatement	0	0%
Oml2VppVisitor‘visitMapEnumeration	0	0%
Oml2VppVisitor‘visitMutexPredicate	0	0%
Oml2VppVisitor‘visitNumericLiteral	322	√
Oml2VppVisitor‘visitPeriodicThread	0	0%
Oml2VppVisitor‘visitRecordModifier	0	0%
Oml2VppVisitor‘visitSelfExpression	0	0%
Oml2VppVisitor‘visitSeqConcPattern	0	0%
Oml2VppVisitor‘visitSeqEnumPattern	0	0%
Oml2VppVisitor‘visitSetEnumPattern	0	0%
Oml2VppVisitor‘visitSetEnumeration	36	√
Oml2VppVisitor‘visitSpecifications	0	0%
Oml2VppVisitor‘visitSporadicThread	0	0%
Oml2VppVisitor‘visitStartStatement	0	0%
Oml2VppVisitor‘visitTimeExpression	0	0%
Oml2VppVisitor‘visitTraceOneOrMore	0	0%

Name	#Calls	Coverage
Oml2VppVisitor‘visitTraceZeroOrOne	0	0%
Oml2VppVisitor‘visitTrapDefinition	0	0%
Oml2VppVisitor‘visitTypeDefinition	0	0%
Oml2VppVisitor‘visitVarInformation	0	0%
Oml2VppVisitor‘visitAlwaysStatement	0	0%
Oml2VppVisitor‘visitApplyExpression	715	√
Oml2VppVisitor‘visitAssignStatement	0	0%
Oml2VppVisitor‘visitAtomicStatement	0	0%
Oml2VppVisitor‘visitCasesExpression	0	0%
Oml2VppVisitor‘visitCyclesStatement	0	0%
Oml2VppVisitor‘visitDontCarePattern	0	0%
Oml2VppVisitor‘visitElseIfStatement	0	0%
Oml2VppVisitor‘visitFunctionTrailer	0	0%
Oml2VppVisitor‘visitLetBeExpression	0	0%
Oml2VppVisitor‘visitPatternTypePair	0	0%
Oml2VppVisitor‘visitProcedureThread	0	0%
Oml2VppVisitor‘visitReturnStatement	0	0%
Oml2VppVisitor‘visitSequenceForLoop	0	0%
Oml2VppVisitor‘visitSetUnionPattern	0	0%
Oml2VppVisitor‘visitTokenExpression	24	√
Oml2VppVisitor‘visitTraceDefinition	0	0%
Oml2VppVisitor‘visitTraceLetBinding	0	0%
Oml2VppVisitor‘visitTraceZeroOrMore	0	0%
Oml2VppVisitor‘visitTypeDefinitions	0	0%
Oml2VppVisitor‘visitUnaryExpression	0	0%
Oml2VppVisitor‘visitValueDefinition	0	0%
Oml2VppVisitor‘visitAccessDefinition	0	0%
Oml2VppVisitor‘visitActiveExpression	0	0%
Oml2VppVisitor‘visitBinaryExpression	195	√
Oml2VppVisitor‘visitCharacterLiteral	24	√
Oml2VppVisitor‘visitElseIfExpression	12	√
Oml2VppVisitor‘visitEqualsDefinition	0	0%
Oml2VppVisitor‘visitExistsExpression	0	0%
Oml2VppVisitor‘visitExplicitFunction	0	0%
Oml2VppVisitor‘visitForAllExpression	0	0%
Oml2VppVisitor‘visitImplicitFunction	0	0%
Oml2VppVisitor‘visitInjectiveMapType	0	0%
Oml2VppVisitor‘visitInstanceVariable	0	0%
Oml2VppVisitor‘visitLambdaExpression	0	0%
Oml2VppVisitor‘visitMapComprehension	0	0%
Oml2VppVisitor‘visitOperationTrailer	0	0%
Oml2VppVisitor‘visitSetComprehension	0	0%
Oml2VppVisitor‘visitThreadDefinition	0	0%
Oml2VppVisitor‘visitTraceDefinitions	0	0%
Oml2VppVisitor‘visitTraceMethodApply	0	0%
Oml2VppVisitor‘visitTupleConstructor	0	0%
Oml2VppVisitor‘visitValueDefinitions	0	0%
Oml2VppVisitor‘visitDurationStatement	0	0%
Oml2VppVisitor‘visitExplicitOperation	0	0%
Oml2VppVisitor‘visitImplicitOperation	0	0%

Name	#Calls	Coverage
Oml2VppVisitor‘visitInheritanceClause	0	0%
Oml2VppVisitor‘visitMutexAllPredicate	0	0%
Oml2VppVisitor‘visitPatternIdentifier	446	√
Oml2VppVisitor‘visitRecordConstructor	2	√
Oml2VppVisitor‘visitTotalFunctionType	0	0%
Oml2VppVisitor‘visitTraceLetBeBinding	0	0%
Oml2VppVisitor‘visitWaitingExpression	0	0%
Oml2VppVisitor‘visitFunctionDefinition	0	0%
Oml2VppVisitor‘visitFunctionTypeSelect	0	0%
Oml2VppVisitor‘visitIdentifierTypePair	0	0%
Oml2VppVisitor‘visitSetRangeExpression	0	0%
Oml2VppVisitor‘visitThreadIdExpression	0	0%
Oml2VppVisitor‘visitTraceRepeatPattern	0	0%
Oml2VppVisitor‘visitBracketedExpression	17	√
Oml2VppVisitor‘visitFunctionDefinitions	0	0%
Oml2VppVisitor‘visitIsofclassExpression	0	0%
Oml2VppVisitor‘visitOperationDefinition	0	0%
Oml2VppVisitor‘visitPartialFunctionType	0	0%
Oml2VppVisitor‘visitPermissionPredicate	0	0%
Oml2VppVisitor‘visitSameclassExpression	0	0%
Oml2VppVisitor‘visitSequenceEnumeration	0	0%
Oml2VppVisitor‘visitStateDesignatorName	0	0%
Oml2VppVisitor‘visitTraceCoreDefinition	0	0%
Oml2VppVisitor‘visitTraceDefinitionItem	0	0%
Oml2VppVisitor‘visitUndefinedExpression	0	0%
Oml2VppVisitor‘visitAssignmentDefinition	0	0%
Oml2VppVisitor‘visitObjectFieldReference	0	0%
Oml2VppVisitor‘visitOperationDefinitions	0	0%
Oml2VppVisitor‘visitPatternBindExpression	0	0%
Oml2VppVisitor‘visitSequenceComprehension	0	0%
Oml2VppVisitor‘visitSubsequenceExpression	0	0%
Oml2VppVisitor‘visitTraceChoiceDefinition	0	0%
Oml2VppVisitor‘visitClassTypeInstantiation	0	0%
Oml2VppVisitor‘visitExistsUniqueExpression	0	0%
Oml2VppVisitor‘visitMapOrSequenceReference	0	0%
Oml2VppVisitor‘visitPreconditionExpression	0	0%
Oml2VppVisitor‘visitRecursiveTrapStatement	0	0%
Oml2VppVisitor‘visitSpecificationStatement	0	0%
Oml2VppVisitor‘visitSymbolicLiteralPattern	0	0%
Oml2VppVisitor‘visitIsofbaseclassExpression	0	0%
Oml2VppVisitor‘visitSamebaseclassExpression	0	0%
Oml2VppVisitor‘visitTraceSequenceDefinition	0	0%
Oml2VppVisitor‘visitExtendedExplicitFunction	0	0%
Oml2VppVisitor‘visitTraceBracketedDefinition	0	0%
Oml2VppVisitor‘visitTypelessExplicitFunction	0	0%
Oml2VppVisitor‘visitCasesStatementAlternative	0	0%
Oml2VppVisitor‘visitExtendedExplicitOperation	0	0%
Oml2VppVisitor‘visitFunctionTypeInstantiation	0	0%
Oml2VppVisitor‘visitInstanceVariableInvariant	0	0%
Oml2VppVisitor‘visitNondeterministicStatement	0	0%

Name	#Calls	Coverage
Oml2VppVisitor‘visitSymbolicLiteralExpression	960	$\surd$
Oml2VppVisitor‘visitCasesExpressionAlternative	0	0%
Oml2VppVisitor‘visitObjectDesignatorExpression	0	0%
Oml2VppVisitor‘visitSynchronizationDefinitions	0	0%
Oml2VppVisitor‘visitInstanceVariableDefinitions	0	0%
Total Coverage		**26%**

9 Standard Utilities

class $StdLib$
types
public $String = \text{char}^*$
functions
public static
$ToString[@Elem] : @Elem \to String$
$ToString\,(s) \triangleq$
cases true :
$(\text{is-}\mathbb{Z}\,(s)) \to ToStringInt\,(s)$,
$(\text{is-}\mathbb{N}\,(s)) \to ToStringInt\,(s)$,
$(\text{is-}\mathbb{N}_1\,(s)) \to ToStringInt\,(s)$,
$(\text{is-}\mathbb{B}\,(s)) \to ToStringBool\,(s)$,
others $\to$ undefined
end;
public static
$ToStringBool : \mathbb{B} \to String$
$ToStringBool\,(pval) \triangleq$
if $pval$
then "$true$"
else "$false$";
public static
$ToStringInt : \mathbb{Z} \to String$
$ToStringInt\,(val) \triangleq$
let $result : \mathbb{Z} = val$ mod 10,
$rest : \mathbb{Z} = val$ div 10 in
if $rest > 0$
then $ToStringInt\,(rest) \curvearrowright GetStringFromNum\,(result)$
else $GetStringFromNum\,(result)$
pre $val \geq 0$;
public static
$GetStringFromNum : \mathbb{Z} \to String$
$GetStringFromNum\,(val) \triangleq$
["0123456789" $(val + 1)$]
pre $val < 10$;
public static
$StringToInt : String \to \mathbb{Z}$
$StringToInt\,(text) \triangleq$
if len $text = 1$
then $CharToInt$ (hd $text$, len $text$)
else $CharToInt$ (hd $text$, len $text$) $+$ $StringToInt$ (tl $text$);
private static

```
        CharToInt : char × ℤ → ℤ
        CharToInt (c, pos) ≜
          let valueMap = {'0' ↦ 0,
                          '1' ↦ 1,
                          '2' ↦ 2,
                          '3' ↦ 3,
                          '4' ↦ 4,
                          '5' ↦ 5,
                          '6' ↦ 6,
                          '7' ↦ 7,
                          '8' ↦ 8,
                          '9' ↦ 9} in
          valueMap (c) × (10 ↑ (pos − 1))
        pre ∃ tmp ∈ elems "1234567890" · tmp = c ;
public static
        SetToSeq[@Elem] : @Elem-set → @Elem*
        SetToSeq (s) ≜
          if s = {}
          then []
          else let x ∈ s in
               SetToSeq[@Elem] (s \ {x}) ↷ [x];
public static
        StringToBool : String → 𝔹
        StringToBool (val) ≜
          val = "true";
public static
        Split : char* × char → char**
        Split (text, delimiter) ≜
          let del-l = [i | i ∈ inds text · text (i) = delimiter] in
          if del-l = []
          then [text]
          else [text (1, ..., del-l (1) − 1)] ↷
               [text (del-l (i) + 1, ..., del-l (i + 1) − 1) | i ∈ inds del-l \ {len del-l}] ↷
               [text (del-l (len del-l) + 1, ..., len text)]
        pre len text > 1
end StdLib
```

Test Suite : vdm.tc
Class : StdLib

Name	#Calls	Coverage
StdLib'Split	0	0%
StdLib'SetToSeq	0	0%
StdLib'ToString	15	31%
StdLib'CharToInt	0	0%
StdLib'StringToInt	0	0%
StdLib'ToStringInt	1506	87%
StdLib'StringToBool	0	0%
StdLib'ToStringBool	0	0%
StdLib'GetStringFromNum	1506	66%
Total Coverage		**16%**

```
    class TC is subclass of TestCase
operations
```

public

$TC : () \xrightarrow{o} TC$

$TC\,() \triangleq$

$TestCase(\text{"}TC\text{"})$;

protected

$SetUp : () \xrightarrow{o} ()$

$SetUp\,() \triangleq$

skip;

public

$test\text{-}numminus\text{-}numminus\text{-}01 : () \xrightarrow{o} ()$

$test\text{-}numminus\text{-}numminus\text{-}01\,() \triangleq$

(let $specTest : SpecTest =$ new $SpecTest\,()$ in

$assertTrue$

$(specTest.Run\,(TCTestData`test\text{-}numminus\text{-}numminus\text{-}01\,().getSpecifications\,(),$

$\text{"}numminus\text{-}01.vdm\text{"}) =$

$\{\text{"}UseStackPushBeforePop1\text{"} \mapsto [\text{"}s.Reset()\text{"}, \text{"}let\ x = 46\ in\ s.Push(x)\text{"}, \text{"}let\ x =$
$46\ in\ s.Push(x)\text{"}, \text{"}s.Pop()\text{"}, \text{"}s.Pop()\text{"}, \text{"}s.Pop()\text{"}], \text{"}UseStackPushBeforePop2\text{"} \mapsto [\text{"}s.Reset()\text{"}, \text{"}let\ x =$
$46\ in\ s.Push(x)\text{"}, \text{"}s.Pop()\text{"}, \text{"}s.Pop()\text{"}, \text{"}s.Pop()\text{"}], \text{"}UseStackPushBeforePop3\text{"} \mapsto [\text{"}s.Reset()\text{"}, \text{"}let\ x =$
$46\ in\ s.Push(x)\text{"}, \text{"}let\ x = 46\ in\ s.Push(x)\text{"}, \text{"}s.Pop()\text{"}, \text{"}s.Pop()\text{"}], \text{"}UseStackPushBeforePop4\text{"} \mapsto$
$[\text{"}s.Reset()\text{"}, \text{"}let\ x = 46\ in\ s.Push(x)\text{"}, \text{"}s.Pop()\text{"}, \text{"}s.Pop()\text{"}], \text{"}UseStackPushBeforePop5\text{"} \mapsto$
$[\text{"}s.Reset()\text{"}, \text{"}let\ x = 46\ in\ s.Push(x)\text{"}, \text{"}let\ x = 46\ in\ s.Push(x)\text{"}, \text{"}s.Pop()\text{"}], \text{"}UseStackPushBeforePop6\text{"} \mapsto$
$[\text{"}s.Reset()\text{"}, \text{"}let\ x = 46\ in\ s.Push(x)\text{"}, \text{"}s.Pop()\text{"}]\})$

);

public

$test\text{-}setinter\text{-}setinter\text{-}01 : () \xrightarrow{o} ()$

$test\text{-}setinter\text{-}setinter\text{-}01\,() \triangleq$

(let $specTest : SpecTest =$ new $SpecTest\,()$ in

$assertTrue$

$(specTest.Run\,(TCTestData`test\text{-}setinter\text{-}setinter\text{-}01\,().getSpecifications\,(),$

$\text{"}setinter\text{-}01.vdm\text{"}) =$

$\{\text{"}UseTreeT27\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}], \text{"}UseTreeT26$
$[\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*$
$x)\text{"}], \text{"}UseTreeT29\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}, \text{"}let\ x =$
$5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}], \text{"}UseTreeT28\text{"} \mapsto$
$[\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}, \text{"}let\ x =$
$5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}], \text{"}UseTreeT23\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))$
$5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}], \text{"}UseTreeT22\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))$
$5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}], \text{"}UseTreeT25\text{"} \mapsto$
$[\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*$
$x)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}], \text{"}UseTreeT24\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x =$
$5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2$
$[\text{"}t1.addRoot(8)\text{"}, \text{"}let\ x = 6\ in\ t1.insertNode(x)\text{"}], \text{"}UseTreeT14\text{"} \mapsto [\text{"}t1.addRoot(8)\text{"}, \text{"}let\ x =$
$9\ in\ t1.insertNode(x)\text{"}], \text{"}UseTreeT11\text{"} \mapsto [\text{"}t1.addRoot(8)\text{"}, \text{"}let\ x = 1\ in\ t1.insertNode(x)\text{"}], \text{"}UseTreeT12\text{"} \mapsto$
$[\text{"}t1.addRoot(8)\text{"}, \text{"}let\ x = 3\ in\ t1.insertNode(x)\text{"}], \text{"}UseTreeT211\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x =$
$5\ in\ t2.insertNode(x*x)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}], \text{"}U$
$[\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*$
$x)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}], \text{"}UseTreeT210\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x =$
$5\ in\ t2.insertNode(x*x)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}], \text{"}UseTreeT21\text{"} \mapsto$
$[\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}, \text{"}let\ x = 5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}, \text{"}let$
$5\ in\ t1.insertNode(x\text{-}(x\ div\ 2))\text{"}], \text{"}UseTreeT213\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*$
$x)\text{"}, \text{"}let\ x = 5\ in\ t2.insertNode(x*x)\text{"}], \text{"}UseTreeT214\text{"} \mapsto [\text{"}t2.addRoot(8)\text{"}, \text{"}let\ x =$
$5\ in\ t2.insertNode(x * x)\text{"}]\})$

);

public

test-ifthen-ifthen-01 : () $\xrightarrow{o}$ ()
test-ifthen-ifthen-01 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*‘*test-ifthen-ifthen*-01 ().*getSpecifications* (),
"*ifthen*-01.*vdm*") =
{"*UseStackPushBeforePop*6" ↦ ["*s.Reset*()", "*let x = 46 in s.Push*(*x*)", "*let x =*
46 *in s.Push*(*x*)", "*s.Pop*()"], "*UseStackPushBeforePop*2" ↦ ["*s.Reset*()", "*let x = 46 in s.Push*(*x*)", "*s.Pop*()"
["*s.Reset*()", "*let x = 46 in s.Push*(*x*)", "*s.Pop*()"], "*UseStackPushBeforePop*4" ↦ ["*s.Reset*()", "*let x =*
46 *in s.Push*(*x*)", "*let x = 46 in s.Push*(*x*)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*5" ↦
["*s.Reset*()", "*let x = 46 in s.Push*(*x*)", "*let x = 46 in s.Push*(*x*)", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBef
["*s.Reset*()", "*let x = 46 in s.Push*(*x*)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"]})
);
public
test-ifthen-ifthen-02 : () $\xrightarrow{o}$ ()
test-ifthen-ifthen-02 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*‘*test-ifthen-ifthen*-02 ().*getSpecifications* (),
"*ifthen*-02.*vdm*") =
{"*UseStackPushBeforePop*1" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>,
11)", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*2" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in*
11)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*3" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in*
11)", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*4" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’,
11)", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*5" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’,
11)", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)", "*s.Pop*()"], "*UseStackPushBeforePop*6" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *n*
11)", "*s.Pop*()"], "*UseStackPushBeforePop*7" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *n*
11)", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)"], "*UseStackPushBeforePop*8" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_to*
11)"]})
);
public

test-ifthen-ifthen-03 : () $\xrightarrow{o}$ ()
test-ifthen-ifthen-03 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*‘*test-ifthen-ifthen*-03 ().*getSpecifications* (),
"*ifthen*-03.*vdm*") =
{"*UseStackPushBeforePop*1" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*2" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in*
11)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*3" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in*
11)", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*4" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’,
11)", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*5" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’,
11)", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)", "*s.Pop*()"], "*UseStackPushBeforePop*6" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *n*
11)", "*s.Pop*()"], "*UseStackPushBeforePop*7" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *n*
11)", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_token*(8), \"*hi*\", 8.699999999999999} *then* 7
11)"], "*UseStackPushBeforePop*8" ↦ ["*s.Reset*()", "*s.Push*(*if true and nil in set* {7, ’*c*’, <*Q*>, *nil* , *mk_to*
11)"]})
);
public
test-letexpr-letexpr-01 : () $\xrightarrow{o}$ ()
test-letexpr-letexpr-01 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*‘*test-letexpr-letexpr*-01 ().*getSpecifications* (),
"*letexpr*-01.*vdm*") =
{"*UseStackPushBeforePop*9" ↦ ["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*let x* =
1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()"], "*UseStackPushBeforePop*8" ↦
["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()", "*s
["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()", "*s
["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()"], "*UseStackPushBeforePop*2" ↦
["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*3" ↦
["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()"], "*UseStackPushBeforePop*4" ↦ ["*s.Reset*()", "*let x* =
1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*5" ↦
["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBefore
["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* =
1 *in s.Push*(*x*)", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*12" ↦ ["*s.Reset*()", "*let x* =
1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* =
1 *in s.Push*(*x*)", "*s.Pop*()"], "*UseStackPushBeforePop*1" ↦ ["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*s.Pop*()", "
["*s.Reset*()", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* = 1 *in s.Push*(*x*)", "*let x* =
1 *in s.Push*(*x*)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"]})
);
public

test-lookup-lookup-01 : () $\xrightarrow{o}$ ()
test-lookup-lookup-01 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*`*test-lookup-lookup*-01 ().*getSpecifications* (),
"*lookup*-01.*vdm*") =
{"*UseStackTest2Dir*22" $\mapsto$ ["*let z = 4 in s.Push(z)*", "*let z =*
4 *in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*21" $\mapsto$ ["*let z = 4 in s.Push(z)*", "*let z =*
4 *in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*2" $\mapsto$ ["*s2.Reset*()"], "*UseStackTest2Dir*1" $\mapsto$
["*s2.Reset*()", "*let a = 1 , b = 2 in s.Push(a)*", "*let a = 1 , b = 2 in s2.Push(b)*"], "*UseStackPushBefo*
["*s.Reset*()", "*let x = 1 in s.Push(x)*", "*s.Pop*()"], "*UseStackPushBeforePop*3" $\mapsto$ ["*s.Reset*()", "*let x =*
2 *in s.Push(x)*", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*4" $\mapsto$ ["*s.Reset*()", "*let x =*
2 *in s.Push(x)*", "*s.Pop*()"], "*UseStackPushBeforePop*1" $\mapsto$ ["*s.Reset*()", "*let x = 1 in s.Push(x)*", "*s.Pop*()", "
["*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*212" $\mapsto$
["*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*215" $\mapsto$
[], "*UseStackTest2Dir*214" $\mapsto$ ["*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*211" $\mapsto$ ["*let z =*
5 *in s.Push(z)*", "*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*210" $\mapsto$
["*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*27" $\mapsto$ ["*let z =*
4 *in s.Push(z)*"], "*UseStackTest2Dir*28" $\mapsto$ ["*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*", "*let z =*
4 *in s.Push(z)*"], "*UseStackTest2Dir*29" $\mapsto$ ["*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*", "*let z =*
5 *in s.Push(z)*"], "*UseStackTest2Dir*23" $\mapsto$ ["*let z = 4 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStack*
["*let z = 4 in s.Push(z)*", "*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*25" $\mapsto$
["*let z = 4 in s.Push(z)*", "*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*26" $\mapsto$
["*let z = 4 in s.Push(z)*", "*let z = 5 in s.Push(z)*"]})
);
public
test-lookup-lookup-02 : () $\xrightarrow{o}$ ()
test-lookup-lookup-02 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*`*test-lookup-lookup*-02 ().*getSpecifications* (),
"*lookup*-02.*vdm*") =
{"*UseStackTest2Dir*1" $\mapsto$ ["*s2.Reset*()", "*let b = 2 , a = 1 in s.Push(a)*", "*let b =*
2 , *a = 1 in s2.Push(b)*"], "*UseStackTest2Dir*2" $\mapsto$ ["*s2.Reset*()"], "*UseStackTest2Dir*21" $\mapsto$
["*let z = 4 in s.Push(z)*", "*let z = 4 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*22" $\mapsto$
["*let z = 4 in s.Push(z)*", "*let z = 4 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*23" $\mapsto$
["*let z = 4 in s.Push(z)*", "*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*24" $\mapsto$
["*let z = 4 in s.Push(z)*", "*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*25" $\mapsto$
["*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*26" $\mapsto$
["*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*27" $\mapsto$
["*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*28" $\mapsto$
["*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*29" $\mapsto$
["*let z = 4 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*210" $\mapsto$
["*let z = 4 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*211" $\mapsto$
["*let z = 5 in s.Push(z)*", "*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*212" $\mapsto$
["*let z = 5 in s.Push(z)*", "*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*213" $\mapsto$
["*let z = 4 in s.Push(z)*"], "*UseStackTest2Dir*214" $\mapsto$ ["*let z = 5 in s.Push(z)*"], "*UseStackTest2Dir*215" $\mapsto$
[], "*UseStackPushBeforePop*1" $\mapsto$ ["*s.Reset*()", "*let x = 1 in s.Push(x)*", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBe*
["*s.Reset*()", "*let x = 2 in s.Push(x)*", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop*3" $\mapsto$
["*s.Reset*()", "*let x = 1 in s.Push(x)*", "*s.Pop*()"], "*UseStackPushBeforePop*4" $\mapsto$ ["*s.Reset*()", "*let x =*
2 *in s.Push(x)*", "*s.Pop*()"]})
);
public

```
        test-newexpr-newexpr-01 : () →o ()
        test-newexpr-newexpr-01 () ≜
          (   let specTest : SpecTest = new SpecTest () in
              assertTrue
                  (specTest.Run (TCTestData`test-newexpr-newexpr-01 ().getSpecifications (),
                                 "newexpr-01.vdm") =
                   {"UseOSstartok1" ↦ ["let rl = 1 , p = new Httpd() in os.addProcess(rl, p)", "let rl =
1 , p = new Httpd() in os.bootSequenceList(rl)"], "UseOSstartok2" ↦ ["let rl = 2 , p =
new Httpd() in os.addProcess(rl, p)", "let rl = 2 , p = new Httpd() in os.bootSequenceList(rl)"], "UseOSsta
["let rl = 1 , p = new Kerneld() in os.addProcess(rl, p)", "let rl = 1 , p =
new Kerneld() in os.bootSequenceList(rl)"], "UseOSstartok4" ↦ ["let rl = 2 , p =
new Kerneld() in os.addProcess(rl, p)", "let rl = 2 , p = new Kerneld() in os.bootSequenceList(rl)"], "Use
["let rl = 1 , p = new Httpd() in os.addProcess(rl, p)", "let rl = 1 , p =
new Httpd() in os.bootSequence(rl)"], "UseOSstartok6" ↦ ["let rl = 2 , p = new Httpd() in os.addProcess(r
2 , p = new Httpd() in os.bootSequence(rl)"], "UseOSstartok7" ↦ ["let rl = 1 , p =
new Kerneld() in os.addProcess(rl, p)", "let rl = 1 , p = new Kerneld() in os.bootSequence(rl)"], "UseOSs
["let rl = 2 , p = new Kerneld() in os.addProcess(rl, p)", "let rl = 2 , p =
new Kerneld() in os.bootSequence(rl)"]})
          );
public
        test-recordconstructor-recordconstructor-01 : () →o ()
        test-recordconstructor-recordconstructor-01 () ≜
          (   let specTest : SpecTest = new SpecTest () in
              assertTrue
                  (specTest.Run (TCTestData`test-recordconstructor-recordconstructor-01 ().getSpecifications (),
                                 "recordconstructor-01.vdm") =
                   {"UseStackPushBeforePop1" ↦ ["s.Reset()", "let x = mk_Stack`R(2) in s.Push(x.a)", "s.Pop
          );
public
        test-simpletraces-simpletraces-01 : () →o ()
        test-simpletraces-simpletraces-01 () ≜
          (   let specTest : SpecTest = new SpecTest () in
              assertTrue
                  (specTest.Run (TCTestData`test-simpletraces-simpletraces-01 ().getSpecifications (),
                                 "simpletraces-01.vdm") =
                   {"UseStackPushBeforePop1" ↦ ["s.Reset()"]})
          );
public
        test-tracebind-tracebind-01 : () →o ()
        test-tracebind-tracebind-01 () ≜
          (   let specTest : SpecTest = new SpecTest () in
              assertTrue
                  (specTest.Run (TCTestData`test-tracebind-tracebind-01 ().getSpecifications (),
                                 "tracebind-01.vdm") =
                   {"UseStackPushBeforePop1" ↦ ["let y = 3 , x = 1 in s.Push(x)", "let y =
3 , x = 1 in s.Push(y)"], "UseStackPushBeforePop2" ↦ ["let y = 8 , x = 1 in s.Push(x)", "let y =
8 , x = 1 in s.Push(y)"], "UseStackPushBeforePop3" ↦ ["let y = 3 , x = 3 in s.Push(x)", "let y =
3 , x = 3 in s.Push(y)"], "UseStackPushBeforePop4" ↦ ["let y = 8 , x = 3 in s.Push(x)", "let y =
8 , x = 3 in s.Push(y)"]})
          );
public
```

test-tracebind-tracebind-02 : () $\xrightarrow{o}$ ()
test-tracebind-tracebind-02 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*`*test-tracebind-tracebind-02* ().*getSpecifications* (),
"*tracebind-02.vdm*") =
{"*UseStackPushBeforePop1*" ↦ ["*let y = 3 , x = 1 in s.Push(x)*", "*let y =*
3 , x = 1 in s.Push(y)"], "*UseStackPushBeforePop2*" ↦ ["*let y = 3 , x = 1 in s.Push(x)*", "*let y =*
8 , x = 1 in s.Push(y)"], "*UseStackPushBeforePop3*" ↦ ["*let y = 3 , x = 1 in s.Push(x)*", "*let y =*
3 , x = 3 in s.Push(y)"], "*UseStackPushBeforePop4*" ↦ ["*let y = 3 , x = 1 in s.Push(x)*", "*let y =*
8 , x = 3 in s.Push(y)"], "*UseStackPushBeforePop5*" ↦ ["*let y = 8 , x = 1 in s.Push(x)*", "*let y =*
3 , x = 1 in s.Push(y)"], "*UseStackPushBeforePop6*" ↦ ["*let y = 8 , x = 1 in s.Push(x)*", "*let y =*
8 , x = 1 in s.Push(y)"], "*UseStackPushBeforePop7*" ↦ ["*let y = 8 , x = 1 in s.Push(x)*", "*let y =*
3 , x = 3 in s.Push(y)"], "*UseStackPushBeforePop8*" ↦ ["*let y = 8 , x = 1 in s.Push(x)*", "*let y =*
8 , x = 3 in s.Push(y)"], "*UseStackPushBeforePop9*" ↦ ["*let y = 3 , x = 3 in s.Push(x)*", "*let y =*
3 , x = 1 in s.Push(y)"], "*UseStackPushBeforePop10*" ↦ ["*let y = 3 , x = 3 in s.Push(x)*", "*let y =*
8 , x = 1 in s.Push(y)"], "*UseStackPushBeforePop11*" ↦ ["*let y = 3 , x = 3 in s.Push(x)*", "*let y =*
3 , x = 3 in s.Push(y)"], "*UseStackPushBeforePop12*" ↦ ["*let y = 3 , x = 3 in s.Push(x)*", "*let y =*
8 , x = 3 in s.Push(y)"], "*UseStackPushBeforePop13*" ↦ ["*let y = 8 , x = 3 in s.Push(x)*", "*let y =*
3 , x = 1 in s.Push(y)"], "*UseStackPushBeforePop14*" ↦ ["*let y = 8 , x = 3 in s.Push(x)*", "*let y =*
8 , x = 1 in s.Push(y)"], "*UseStackPushBeforePop15*" ↦ ["*let y = 8 , x = 3 in s.Push(x)*", "*let y =*
3 , x = 3 in s.Push(y)"], "*UseStackPushBeforePop16*" ↦ ["*let y = 8 , x = 3 in s.Push(x)*", "*let y =*
8 , x = 3 in s.Push(y)"]})
);
public
test-tracebind-tracebind-03 : () $\xrightarrow{o}$ ()
test-tracebind-tracebind-03 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*`*test-tracebind-tracebind-03* ().*getSpecifications* (),
"*tracebind-03.vdm*") =
{"*UseStackPushBeforePop6*" ↦ ["*s.Reset*()", "*let x = 28 in s.Push(x)*"], "*UseStackPushBefore*
["*s.Reset*()", "*let x = 28 in s.Push(x)*", "*s.Pop*()"], "*UseStackPushBeforePop3*" ↦ ["*s.Reset*()", "*let x =*
28 in s.Push(x)", "*let x = 28 in s.Push(x)*", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop4*" ↦
["*s.Reset*()", "*let x = 28 in s.Push(x)*", "*let x = 28 in s.Push(x)*", "*s.Pop*()"], "*UseStackPushBeforePop5*" ↦
["*s.Reset*()", "*let x = 28 in s.Push(x)*", "*let x = 28 in s.Push(x)*"], "*UseStackPushBeforePop1*" ↦
["*s.Reset*()", "*let x = 28 in s.Push(x)*", "*s.Pop*()", "*s.Pop*()"]})
);
public
test-tracebind-tracebind-04 : () $\xrightarrow{o}$ ()
test-tracebind-tracebind-04 () $\triangleq$
(let *specTest* : *SpecTest* = new *SpecTest* () in
assertTrue
(*specTest.Run* (*TCTestData*`*test-tracebind-tracebind-04* ().*getSpecifications* (),
"*tracebind-04.vdm*") =
{"*UseStackPushBeforePop1*" ↦ ["*s.Reset*()", "*let x = 1 in s.Push(x)*", "*let x =*
1 in s.Push(x)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop2*" ↦ ["*s.Reset*()", "*let x =*
1 in s.Push(x)", "*s.Pop*()", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop3*" ↦ ["*s.Reset*()", "*let x =*
1 in s.Push(x)", "*let x = 1 in s.Push(x)*", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop4*" ↦
["*s.Reset*()", "*let x = 1 in s.Push(x)*", "*s.Pop*()", "*s.Pop*()"], "*UseStackPushBeforePop5*" ↦
["*s.Reset*()", "*let x = 1 in s.Push(x)*", "*let x = 1 in s.Push(x)*", "*s.Pop*()"], "*UseStackPushBeforePop6*" ↦
["*s.Reset*()", "*let x = 1 in s.Push(x)*", "*s.Pop*()"]})
);
public

```
        test-tracebracket-tracebracket-01 : () →o ()
        test-tracebracket-tracebracket-01 () ≜
          (    let specTest : SpecTest = new SpecTest () in
               assertTrue
                    (specTest.Run (TCTestData`test-tracebracket-tracebracket-01 ().getSpecifications (),
                                   "tracebracket-01.vdm") =
                     {"UseStackPushBeforePop1" ↦ ["let y = 3 , x = 1 in s.Push(x)", "let y =
3 , x = 1 in let z = 6 in s.Push(y)", "let y = 3 , x = 1 in let z =
6 in s.Push(z)"], "UseStackPushBeforePop2" ↦ ["let y = 8 , x = 1 in s.Push(x)", "let y =
8 , x = 1 in let z = 6 in s.Push(y)", "let y = 8 , x = 1 in let z =
6 in s.Push(z)"], "UseStackPushBeforePop3" ↦ ["let y = 3 , x = 3 in s.Push(x)", "let y =
3 , x = 3 in let z = 6 in s.Push(y)", "let y = 3 , x = 3 in let z =
6 in s.Push(z)"], "UseStackPushBeforePop4" ↦ ["let y = 8 , x = 3 in s.Push(x)", "let y =
8 , x = 3 in let z = 6 in s.Push(y)", "let y = 8 , x = 3 in let z =
6 in s.Push(z)"]})
          );
public
        test-tracechoice-tracechoice-01 : () →o ()
        test-tracechoice-tracechoice-01 () ≜
          (    let specTest : SpecTest = new SpecTest () in
               assertTrue
                    (specTest.Run (TCTestData`test-tracechoice-tracechoice-01 ().getSpecifications (),
                                   "tracechoice-01.vdm") =
                     {"UseStackPushBeforePop1" ↦ ["s.Push(6)"], "UseStackPushBeforePop2" ↦
["s.Reset()"]})
          );
public
        test-tracechoice-tracechoice-02 : () →o ()
        test-tracechoice-tracechoice-02 () ≜
          (    let specTest : SpecTest = new SpecTest () in
               assertTrue
                    (specTest.Run (TCTestData`test-tracechoice-tracechoice-02 ().getSpecifications (),
                                   "tracechoice-02.vdm") =
                     {"UseStackPushBeforePop1" ↦ ["s.Push(6)", "s.Push(6)", "s.Push(6)"], "UseStackPushBeforePop
["s.Push(6)", "s.Push(6)"], "UseStackPushBeforePop3" ↦ ["s.Push(6)"], "UseStackPushBeforePop4" ↦
["s.Reset()"], "UseStackPushBeforePop5" ↦ []})
          );
public
        test-tracechoice-tracechoice-03 : () →o ()
        test-tracechoice-tracechoice-03 () ≜
          (    let specTest : SpecTest = new SpecTest () in
               assertTrue
                    (specTest.Run (TCTestData`test-tracechoice-tracechoice-03 ().getSpecifications (),
                                   "tracechoice-03.vdm") =
                     {"CUPInitBeforePlay1" ↦ ["gp.Win(<\"Norway\">, <\"Morocco\">)"], "CUPInitBeforePlay2
["gp.Win(<\"Brazil\">, <\"Denmark\">)"]})
          );
public
```

$test\text{-}tracechoice\text{-}tracechoice\text{-}04 : () \xrightarrow{o} ()$
$test\text{-}tracechoice\text{-}tracechoice\text{-}04\,() \triangleq$
(let $specTest : SpecTest =$ new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}tracechoice\text{-}tracechoice\text{-}04\,().getSpecifications\,(),$
"$tracechoice$-04.vdm") =
{"$UseTreeinsertionBST8$" ↦ ["$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ =$
$2\ \ in\ \ t1.Insert(n)$", "$t1.isEmpty()$"], "$UseTreeinsertionBST9$" ↦ ["$let\ \ n\ \ = 2\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ =$
$1\ \ in\ \ t1.Insert(n)$", "$t1.breadth\_first\_search()$"], "$UseTreeinsertionBST3$" ↦ ["$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$let\ \ n$
$1\ \ in\ \ t1.Insert(n)$", "$t1.inorder()$"], "$UseTreeinsertionBST10$" ↦ ["$let\ \ n\ \ = 2\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ =$
$1\ \ in\ \ t1.Insert(n)$", "$t1.depth\_first\_search()$"], "$UseTreeinsertionBST2$" ↦ ["$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$let\ \ n$
$1\ \ in\ \ t1.Insert(n)$", "$t1.depth\_first\_search()$"], "$UseTreeinsertionBST1$" ↦ ["$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$let\ \ n$
$1\ \ in\ \ t1.Insert(n)$", "$t1.breadth\_first\_search()$"], "$UseTreeinsertionBST7$" ↦ ["$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$let\ \ n$
$2\ \ in\ \ t1.Insert(n)$", "$t1.inorder()$"], "$UseTreeinsertionBST6$" ↦ ["$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ =$
$2\ \ in\ \ t1.Insert(n)$", "$t1.depth\_first\_search()$"], "$UseTreeinsertionBST5$" ↦ ["$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$let\ \ n$
$2\ \ in\ \ t1.Insert(n)$", "$t1.breadth\_first\_search()$"], "$UseTreeinsertionBST4$" ↦ ["$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$let\ \ n$
$1\ \ in\ \ t1.Insert(n)$", "$t1.isEmpty()$"], "$UseTreeinsertionBST15$" ↦ ["$let\ \ n\ \ = 2\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ =$
$2\ \ in\ \ t1.Insert(n)$", "$t1.inorder()$"], "$UseTreeinsertionBST16$" ↦ ["$let\ \ n\ \ = 2\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ =$
$2\ \ in\ \ t1.Insert(n)$", "$t1.isEmpty()$"], "$UseTreeinsertionBST13$" ↦ ["$let\ \ n\ \ = 2\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ =$
$2\ \ in\ \ t1.Insert(n)$", "$t1.breadth\_first\_search()$"], "$UseTreeinsertionBST14$" ↦ ["$let\ \ n\ \ =$
$2\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ = 2\ \ in\ \ t1.Insert(n)$", "$t1.depth\_first\_search()$"], "$UseTreeinsertionBST11$" ↦
["$let\ \ n\ \ = 2\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$t1.inorder()$"], "$UseTreeinsertionBST12$" ↦
["$let\ \ n\ \ = 2\ \ in\ \ t1.Insert(n)$", "$let\ \ n\ \ = 1\ \ in\ \ t1.Insert(n)$", "$t1.isEmpty()$"]})
);
public
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}01 : () \xrightarrow{o} ()$
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}01\,() \triangleq$
(let $specTest : SpecTest =$ new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}tracerepeat\text{-}tracerepeat\text{-}01\,().getSpecifications\,(),$
"$tracerepeat$-01.vdm") =
{"$UseStackPushBeforePop1$" ↦ ["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$"], "$UseStackPu
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$"], "$UseStackPushBeforePop3$" ↦ ["$s.Reset()$", "$s.Push(6)$"]})
);
public
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}02 : () \xrightarrow{o} ()$
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}02\,() \triangleq$
(let $specTest : SpecTest =$ new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}tracerepeat\text{-}tracerepeat\text{-}02\,().getSpecifications\,(),$
"$tracerepeat$-02.vdm") =
{"$UseStackPushBeforePop1$" ↦ ["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$",
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Pop()$"], "$UseStackPushBeforePop3$" ↦
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Pop()$"], "$UseStackPushBeforePop4$" ↦ ["$s.Reset()$", "$s.Push(6)$", "$s.Pop()$
);
public

$test\text{-}tracerepeat\text{-}tracerepeat\text{-}03 : () \xrightarrow{o} ()$
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}03\,() \triangleq$
(let $specTest : SpecTest$ = new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}tracerepeat\text{-}tracerepeat\text{-}03\,().getSpecifications\,(),$
"$tracerepeat$-03.vdm") =
{"$UseStackPushBeforePop1$" ↦ ["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$",
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Pop()$", "$s.Pop()$", "$s.Pop()$"], "$UseStackPushBeforePop3$" ↦
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Pop()$", "$s.Pop()$"], "$UseStackPushBeforePop4$"
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Pop()$", "$s.Pop()$", "$s.Pop()$"], "$UseStackPushBeforePop5$" ↦
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Pop()$", "$s.Pop()$"], "$UseStackPushBeforePop6$" ↦
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Pop()$"], "$UseStackPushBeforePop7$" ↦
["$s.Reset()$", "$s.Push(6)$", "$s.Pop()$", "$s.Pop()$", "$s.Pop()$"], "$UseStackPushBeforePop8$" ↦ ["$s.Reset()$", "$s.Push(6)$",
["$s.Reset()$", "$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$", "$s.Pop()$"], "$UseStackPushBeforePop10$" ↦
["$s.Reset()$", "$s.Push(6)$", "$s.Pop()$", "$s.Pop()$"], "$UseStackPushBeforePop11$" ↦ ["$s.Reset()$", "$s.Push(6)$", "$s.Push(6
["$s.Reset()$", "$s.Push(6)$", "$s.Pop()$"]})
);
public
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}04 : () \xrightarrow{o} ()$
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}04\,() \triangleq$
(let $specTest : SpecTest$ = new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}tracerepeat\text{-}tracerepeat\text{-}04\,().getSpecifications\,(),$
"$tracerepeat$-04.vdm") =
{"$UseStackPushBeforePop1$" ↦ ["$s.Push(6)$", "$s.Push(6)$", "$s.Push(6)$"], "$UseStackPushBeforePop
["$s.Push(6)$", "$s.Push(6)$"], "$UseStackPushBeforePop3$" ↦ ["$s.Push(6)$"], "$UseStackPushBeforePop4$" ↦
[]})
);
public
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}05 : () \xrightarrow{o} ()$
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}05\,() \triangleq$
(let $specTest : SpecTest$ = new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}tracerepeat\text{-}tracerepeat\text{-}05\,().getSpecifications\,(),$
"$tracerepeat$-05.vdm") =
{"$UseStacktrace11$" ↦ ["$s.Push3(1)$", "$s.Push3(1)$", "$s.Push3(1)$"], "$UseStacktrace12$" ↦
["$s.Push3(1)$", "$s.Push3(1)$"], "$UseStacktrace13$" ↦ ["$s.Push3(1)$"], "$UseStacktrace14$" ↦ [], "$UseStacktrace21$" ↦
["$s.Push3(1)$", "$s.Push3(1)$", "$s.Push3(1)$"], "$UseStacktrace22$" ↦ ["$s.Push3(1)$", "$s.Push3(1)$"], "$UseStacktrace23$" ↦
["$s.Push3(1)$"], "$UseStacktrace31$" ↦ ["$s.Push3(1)$"], "$UseStacktrace32$" ↦ [], "$UseStacktrace41$" ↦
["$s.Push3(1)$", "$s.Push3(1)$", "$s.Push3(1)$"], "$UseStacktrace42$" ↦ ["$s.Push3(1)$", "$s.Push3(1)$"], "$UseStacktrace51$" ↦
["$s.Push3(1)$", "$s.Push3(1)$", "$s.Push3(1)$", "$s.Push3(1)$", "$s.Pop()$"], "$UseStacktrace52$" ↦
["$s.Push3(1)$", "$s.Push3(1)$", "$s.Push3(1)$", "$s.Pop()$"], "$UseStacktrace53$" ↦ ["$s.Push3(1)$", "$s.Push3(1)$", "$s.Pop()$"]
["$s.Push3(1)$", "$s.Pop()$"], "$UseStacktrace55$" ↦ ["$s.Pop()$"], "$UseStacktrace61$" ↦ ["$let\ \ x\ \ =$
$1\ \ in\ \ s.Push3(x)$", "$s.Pop()$"], "$UseStacktrace62$" ↦ ["$let\ \ x\ \ = 5\ \ in\ \ s.Push3(x)$", "$s.Pop()$"], "$UseStacktrace63$" ↦
["$let\ \ x\ \ = 10\ \ in\ \ s.Push3(x)$", "$s.Pop()$"], "$UseStacktrace71$" ↦ ["$let\ \ x\ \ = 1\ \ in\ \ s.Push3(x)$", "$s.Pop()$"], "$UseSt
["$let\ \ x\ \ = 2\ \ in\ \ s.Push3(x)$", "$s.Pop()$"], "$UseStacktrace73$" ↦ ["$let\ \ x\ \ = 3\ \ in\ \ s.Push3(x)$", "$s.Pop()$"], "$UseSta
["$let\ \ x\ \ = 1\ \ in\ \ s.Push3(x)$"], "$UseStacktrace91$" ↦ ["$s.Push3(3)$"], "$UseStacktrace92$" ↦
["$s.Push3(2)$"], "$UseStacktrace101$" ↦ ["$s.Push3(1)$", "$s.Push3(1)$", "$s.Push3(1)$"], "$UseStacktrace111$" ↦
["$let\ \ x\ \ = 1\ \ in\ \ var.method(x)$"], "$UseStacktrace121$" ↦ ["$var.method(1)$", "$var.method(1)$"], "$UseStacktrace122$" ↦
["$var.method(1)$"], "$UseStacktrace131$" ↦ ["$let\ \ x\ \ = 1\ \ in\ \ var1.method(x)$", "$var2.meth2(10)$"], "$UseStacktrace132$"
["$let\ \ x\ \ = 2\ \ in\ \ var1.method(x)$", "$var2.meth2(10)$"]})
);
public

$test\text{-}tracerepeat\text{-}tracerepeat\text{-}06 : () \xrightarrow{o} ()$
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}06\,() \triangleq$
(let $specTest : SpecTest =$ new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}tracerepeat\text{-}tracerepeat\text{-}06\,().getSpecifications\,(),$
"$tracerepeat$-06.vdm") =
{"$UseAtrace114$" $\mapsto$ ["$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace113$" $\mapsto$
["$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace112$" $\mapsto$ ["$let\ x\ =$
$2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace13$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace14$" $\mapsto$ ["$let\ x\ =$
$1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace15$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace16$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace111$" $\mapsto$ ["$let\ x\ =$
$2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace110$" $\mapsto$
["$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace11$" $\mapsto$ ["$let\ x\ =$
$1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace12$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace18$" $\mapsto$
["$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace17$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace19$" $\mapsto$ ["$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ =$
$1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"]})
);
public
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}07 : () \xrightarrow{o} ()$
$test\text{-}tracerepeat\text{-}tracerepeat\text{-}07\,() \triangleq$
(let $specTest : SpecTest =$ new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}tracerepeat\text{-}tracerepeat\text{-}07\,().getSpecifications\,(),$
"$tracerepeat$-07.vdm") =
{"$UseAtrace114$" $\mapsto$ ["$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace113$" $\mapsto$
["$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace112$" $\mapsto$ ["$let\ x\ =$
$2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace13$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace14$" $\mapsto$ ["$let\ x\ =$
$1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace15$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace16$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace111$" $\mapsto$ ["$let\ x\ =$
$2\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace110$" $\mapsto$
["$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace11$" $\mapsto$ ["$let\ x\ =$
$1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace12$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"], "$UseAtrace18$" $\mapsto$
["$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$", "$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace17$" $\mapsto$
["$let\ x\ = 1\ in\ obj.op(x)$"], "$UseAtrace19$" $\mapsto$ ["$let\ x\ = 2\ in\ obj.op(x)$", "$let\ x\ =$
$1\ in\ obj.op(x)$", "$let\ x\ = 2\ in\ obj.op(x)$"]})
);
public
$test\text{-}traceseq\text{-}traceseq\text{-}01 : () \xrightarrow{o} ()$
$test\text{-}traceseq\text{-}traceseq\text{-}01\,() \triangleq$
(let $specTest : SpecTest =$ new $SpecTest\,()$ in
$assertTrue$
($specTest.Run\,(TCTestData`test\text{-}traceseq\text{-}traceseq\text{-}01\,().getSpecifications\,(),$
"$traceseq$-01.vdm") =
{"$UseStackPushBeforePop1$" $\mapsto$ ["$s.Reset()$", "$s.Push(6)$"]})
);
public

```
        test-traceseq-traceseq-02 : () →o ()
        test-traceseq-traceseq-02 () △
          (   let specTest : SpecTest = new SpecTest () in
              assertTrue
                  (specTest.Run (TCTestData`test-traceseq-traceseq-02 ().getSpecifications (),
                                 "traceseq-02.vdm") =
                   {"UseStackPushBeforePop2" ↦ ["s.Reset()", "let  x    = 2   in  s.Push(x)", "s.Pop()"], "UseStackI
["s.Reset()", "let  x    = 1   in  s.Push(x)", "s.Pop()"]})
          );
public
        runTest : (TestResult) →o ()
        runTest (ptr) △
          (   trap exc : Throwable
              with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                         then ptr.addError(self, exc)
                         else error in
              (   test-numminus-numminus-01()
              ) ;
              trap exc : Throwable
              with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                         then ptr.addError(self, exc)
                         else error in
              (   test-setinter-setinter-01()
              ) ;
              trap exc : Throwable
              with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                         then ptr.addError(self, exc)
                         else error in
              (   test-ifthen-ifthen-01()
              ) ;
              trap exc : Throwable
              with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                         then ptr.addError(self, exc)
                         else error in
              (   test-ifthen-ifthen-02()
              ) ;
              trap exc : Throwable
              with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                         then ptr.addError(self, exc)
                         else error in
```

```
(    test-ifthen-ifthen-03()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-letexpr-letexpr-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-lookup-lookup-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-lookup-lookup-02()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-newexpr-newexpr-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-recordconstructor-recordconstructor-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-simpletraces-simpletraces-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
```

```
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracebind-tracebind-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracebind-tracebind-02()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracebind-tracebind-03()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracebind-tracebind-04()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracebracket-tracebracket-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracechoice-tracechoice-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
```

```
(    test-tracechoice-tracechoice-02()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracechoice-tracechoice-03()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracechoice-tracechoice-04()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracerepeat-tracerepeat-01()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracerepeat-tracerepeat-02()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracerepeat-tracerepeat-03()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
      else if isofbaseclass (Throwable, exc)
           then ptr.addError(self, exc)
           else error in
(    test-tracerepeat-tracerepeat-04()
) ;
trap exc : Throwable
with  if isofclass (AssertionFailedError, exc)
      then ptr.addFailure(self, exc)
```

```
                    else if isofbaseclass (Throwable, exc)
                        then ptr.addError(self, exc)
                        else error in
                (   test-tracerepeat-tracerepeat-05()
                ) ;
                trap exc : Throwable
                with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                        then ptr.addError(self, exc)
                        else error in
                (   test-tracerepeat-tracerepeat-06()
                ) ;
                trap exc : Throwable
                with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                        then ptr.addError(self, exc)
                        else error in
                (   test-tracerepeat-tracerepeat-07()
                ) ;
                trap exc : Throwable
                with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                        then ptr.addError(self, exc)
                        else error in
                (   test-traceseq-traceseq-01()
                ) ;
                trap exc : Throwable
                with  if isofclass (AssertionFailedError, exc)
                    then ptr.addFailure(self, exc)
                    else if isofbaseclass (Throwable, exc)
                        then ptr.addError(self, exc)
                        else error in
                (   test-traceseq-traceseq-02()
                )
            );
protected
        TearDown : () -o-> ()
        TearDown () ≜
          skip
    end TC
```

Index