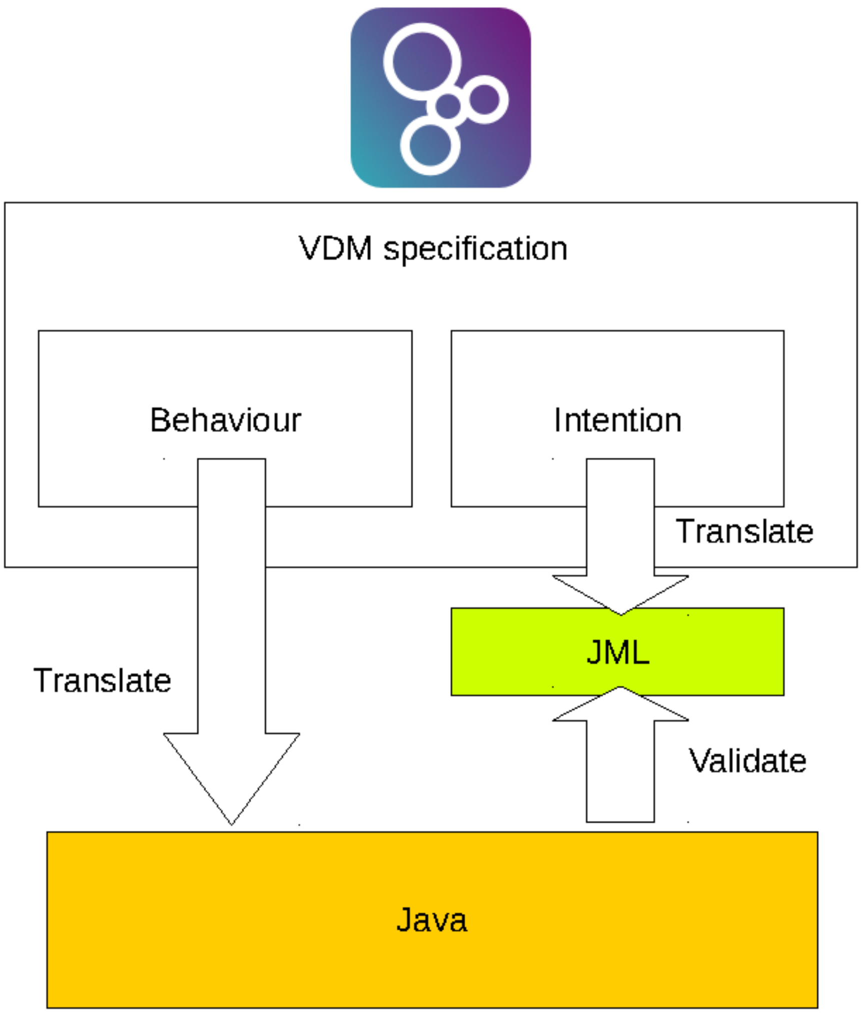
AUTOMATED TRANSLATION OF VDM-SL TO JML-ANNOTATED JAVA

Electrical and Computer Engineering
Technical Report ECE-TR-29



AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

DATA SHEET

Title: Automated translation of VDM-SL to JML-annotated Java
Subtitle: Electrical and Computer Engineering
Series title and no.: Technical report ECE-TR-29

Author: Peter W. V. Tran-Jørgensen
Department of Engineering – Electrical and Computer Engineering,
Aarhus University

Internet version: The report is available in electronic format (pdf) at the Department of Engineering website http://www.eng.au.dk.

Publisher: Aarhus University©
URL: http://www.eng.au.dk

Year of publication: 2017 Pages: 384
Editing completed: February 2017

Abstract: When a system specified using the Vienna Development Method (VDM) is realised using code-generation, no guarantees are currently made about the correctness of the generated code. In this technical report, we improve code-generation of VDM models by taking contract-based elements such as invariants and pre- and postconditions into account during the code-generation process. The contract-based elements of the Vienna Development Method Specification Language (VDM-SL) are translated into corresponding constructs in the Java Modelling Language (JML) and used to validate the generated code against the properties of the VDM model. VDM-SL and JML are both Design-by-Contract (DbC) languages, with the difference that VDM-SL supports abstract modelling and system specification, while JML is used for detailed specification of Java classes and interfaces. We describe the semantic differences between the contract-based elements of VDM-SL and JML and formulate the translation as a set of rules. We further demonstrate how dynamic JML assertion checks can be used to ensure the consistency of VDM's subtypes when a model is code-generated. The translator is fully automated and produces JML-annotated Java programs that can be checked for correctness using JML tools. Specifically, it is shown how such analysis can be performed using the OpenJML runtime assertion checker. The translation is demonstrated using a case study example of an Automated Teller Machine and several other VDM-SL models, which have been used to validate and asses the translation.

Keywords: Design-by-Contract, Formal methods, VDM, Java, JML, Code-generation

Please cite as: Peter W. V. Tran-Jørgensen, 2017. Automated translation of VDM-SL to JML-annotated Java. Department of Engineering, Aarhus University. Denmark. 384 pp. - Technical report ECE-TR-29

Cover image: Peter W. V. Tran-Jørgensen.

ISSN: 2245-2087

Reproduction permitted provided the source is explicitly acknowledged.

AUTOMATED TRANSLATION OF VDM-SL TO JML-ANNOTATED JAVA

Peter W. V. Tran-Jørgensen
Aarhus University, Department of Engineering

Abstract

When a system specified using the Vienna Development Method (VDM) is realised using code-generation, no guarantees are currently made about the correctness of the generated code. In this technical report, we improve code-generation of VDM models by taking contract-based elements such as invariants and pre- and postconditions into account during the code-generation process. The contract-based elements of the Vienna Development Method Specification Language (VDM-SL) are translated into corresponding constructs in the Java Modelling Language (JML) and used to validate the generated code against the properties of the VDM model. VDM-SL and JML are both Design-by-Contract (DbC) languages, with the difference that VDM-SL supports abstract modelling and system specification, while JML is used for detailed specification of Java classes and interfaces. We describe the semantic differences between the contract-based elements of VDM-SL and JML and formulate the translation as a set of rules. We further demonstrate how dynamic JML assertion checks can be used to ensure the consistency of VDM's subtypes when a model is code-generated. The translator is fully automated and produces JML-annotated Java programs that can be checked for correctness using JML tools. Specifically, it is shown how such analysis can be performed using the OpenJML runtime assertion checker. The translation is demonstrated using a case study example of an Automated Teller Machine and several other VDM-SL models, which have been used to validate and asses the translation.

Table of Contents

1

Introduction

A model specified using the VDM can be validated against its contract-based elements (e.g. pre- and postconditions and invariants) in order to ensure that the system behaves as intended. This can, for example, be done through model animation using Overture's VDM interpreter.

When sufficient insight into the system under development has been obtained during the formal analysis, development proceeds to the implementation phase, where the system is realised. One way to realise a VDM model (which forms the focus of this technical report) is by implementing it in a programming language using code generation. However, since no guarantees are currently made about the correctness of the generated code, other measures must be taken to increase the confidence in the correctness of the derived model implementation.

To support this approach, Overture enables fully automated translation of VDM-SL's contract-based elements (pre- and postconditions, and invariants) and type constraints into JML annotations. This translation is achieved using Overture's *JML translator*, which translates VDM-SL models into JML-annotated Java programs. In this way JML tools can be used to validate the generated Java code against the *intended* system behaviour, described using JML. This work-flow is illustrated in fig. 1.1.

1.1 The tool implementation

The translation is defined as a set of rules that are implemented as an extension of Overture's VDM-to-Java code generator [16] to make the approach fully automated. The JML translator is available in Overture 2.3.8 onwards and can be downloaded via the Overture tool's website [30]. For instructions on how to use the JML translator we refer the reader to Overture's user guide [22], specifically the chapter on the VDM-to-Java code generator.

The generated Java programs can be checked for correctness using JML tools that support Java 7 or later. In particular, the generated Java programs, including this translation, have been tested using the OpenJML [8] runtime assertion checker, which at the current time of writing, supports Java 8. In particular, OpenJML, is the only JML tool that we are aware of that currently supports all the JML constructs generated by the JML translator. The most recent version of OpenJML is available via the OpenJML website [29].

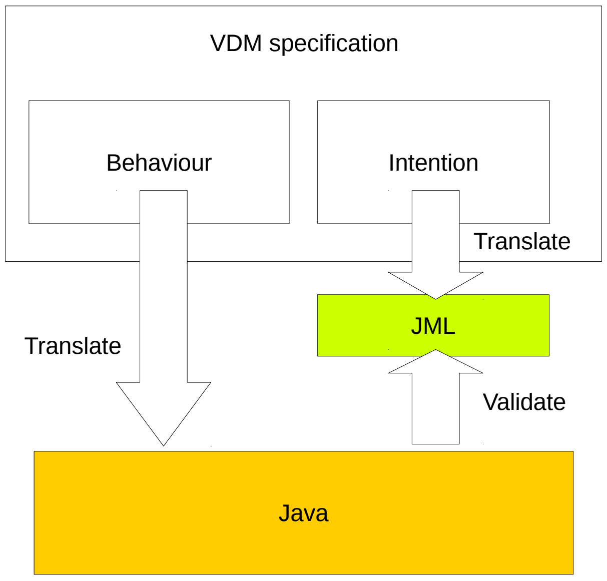


Figure 1.1: Overview of the VDM-to-JML translation.

1.2 About this technical report

The purpose of this report is to assist VDM users with getting started using the JML translator. Moreover, this report is complementary to [36] – a journal paper that describes the JML translation. In that paper, as well as in this report, the translation is exemplified using a VDM-SL model of an ATM. Due to space limitations only a limited number of examples are provided in [36]. To improve this, this report presents a more complete definition of the translation, including more examples that demonstrate how the translation works.

This report is structured as follows: Chapter 2 presents the translation and demonstrates how the JML translator supports the implementation of VDM-SL models in Java. Appendix A contains the complete version of the ATM model, and the corresponding Java/JML implementation is available in appendix B. Finally, all the regression tests that are used to validate the translation rules are presented in appendix C.

2

The Translation

2.1 Introduction

Design-by-Contract (DbC) is an approach for designing software based on concepts such as pre-conditions, postconditions and invariants [27]. These concepts are referred to as "*contracts*", according to a conceptual metaphor for the conditions and obligations of a business contract. An example of a formal method that uses DbC elements is the Vienna Development Method (VDM), which was originally developed at IBM in Vienna for the development of a compiler for PL/1 [5, 10, 11]. One way to realise a VDM specification in a programming language is through refinement [39]. This is a stepwise process by which one can transform a formal model into a program that can be verified to semantically satisfy its contracts [15].

Another way to realise a VDM specification is using code-generation. The idea is for the generated code to be a refinement of the specification, but which is not achieved through step-wise refinement, but rather in one step through code-generation translation rules. Code-generation aims to reduce the resources needed to realise the model as well as to avoid introducing problems in the implementation due to manual translation of model into code. However, current VDM code-generators do not make any guarantees about the correctness of the generated code, nor do they provide the necessary means to help check that the code meets the specification. Naturally, this casts doubt on the value of code-generation as a way to realise a VDM model, since the goal is to develop software that meets the specification.

In this report we improve code-generation of VDM models by allowing the generated code to be checked against the system properties described by the VDM contracts. This helps ensure that the generated code meets the VDM specification, and is achieved as described in this report.

Some DbC technologies are tailored to specify detailed designs of programming interfaces for a particular programming language [38]. An example of one such technology is the Java Modeling Language (JML) [6] – a formal specification language that uses DbC elements, written as specialized comments, to specify the behaviour of Java classes and interfaces. JML annotations can be analysed statically or checked dynamically using JML tools. Therefore, JML can be seen as a technology that serves to bridge the gap between an abstract system specification and its Java implementation.

In this report we attempt to bridge this gap even further by proposing a way to automatically translate a specification written in the Vienna Development Method Specification Language (VDM-SL) to a JML-annotated Java implementation. Current VDM code-generators either ignore or provide limited code-generation support for the contract-based elements and type constraints of VDM. Ideally we should be able to preserve the contracts and type constraints when the system specification is implemented, since (1) they serve to document the intention and properties of the system and (2) they can be used to check the system realisation for correctness. Ensuring that the contracts and type constraints, as originally specified in VDM, hold for the system implementation potentially requires many extra checks to be added to the code. Adding these checks to the code manually is tedious and prone to errors. Instead, these checks could be generated automatically. Representing contracts and type constraints in JML also has the advantage that these checks may be ignored by the Java compiler. This allows the system realisation to be executed without the overhead of checking the contracts and type constraints, if desired.

The two main contributions of our work are (1) a collection of semantics-preserving rules for translating a VDM-SL specification to a JML-annotated Java program and (2) an implementation of these rules as an extension to Overture's [19, 30] VDM-to-Java code-generator [16].

The rules propose ways to translate the DbC elements of VDM-SL to JML annotations; these annotations are added to the Java code produced by Overture's Java code-generator. The rules cover checking of preconditions, postconditions and invariants, but the translator also produces JML checks to ensure that no type constraints are violated across the translation. We present the rules one by one and demonstrate, using a case study model of an Automated Teller Machine (ATM), how the code-generator extension translates a VDM-SL specification to JML-annotated Java code.

Since the translation is not formally defined we have used the OpenJML [8] runtime assertion checker to validate our work — in particular by generating JML constructs supported by this tool. More specifically, the JML translator has been tested by running examples through the tool in order to validate each of the translation rules (see section 2.8 for more details).

Following this section, we describe DbC with VDM-SL and JML in section 2.2. We continue by presenting the implementation of the JML translator in section 2.3. Then we describe the rules used to translate a VDM-SL specification to a JML-annotated Java program in sections 2.5 to 2.7. Next, we assess the correctness of the translation in section 2.8. Finally we describe related work in section 2.9 and present future plans and conclude in section 2.10.

2.2 DbC with VDM-SL and JML

In this section we describe VDM-SL and JML. We cover different types and all the contract-based elements of VDM-SL, focusing specifically on the VDM-10 release, which we are targeting in our work. The JML constructs described in this section cover those that are used to implement the translation rules.

2.2.1 VDM-SL

VDM-SL is an ISO standardised sequential modelling language that supports description of data and functionality. The ISO standard has later been informally extended with modules to allow type definitions, values (constants) and functionality to be imported and exported between modules. A module may define a single state component, which can be constrained by a state invariant. State is modified by assigning a new value to a state designator, which can be either a name, a field reference or a map or sequence reference, as described in the VDM language reference manual [21].

Module state, if specified, implicitly defines a record type, which is tagged with the state name and also defines the type of the state component. The state type can be used like any other record type explicitly defined by the modeller – the difference being that the state invariant [3] constrains the state type and thus every instance of this record type.

Data are defined by means of built-in basic types covering, for instance, numbers, booleans, quote types and characters. A quote type corresponds to an enumerated type in a language such as Pascal. The basic types can be used to form new structured data types using built-in type constructors that support creation of union types, tuple types and record types. A type may also be declared optional, which allows **`nil`** to be used to represent the absence of a value. For collections of values, VDM-SL supports sets, sequences and maps. The built-in data types, type constructors and collections can be used to form named user defined types, which can be constrained by invariants. We refer to these types as *named invariant types*. As an example, Listing 2.1 shows the definition of the named invariant type `Amount`, which is used to represent an amount of money deposited or withdrawn by an account holder. This type is defined based on natural numbers (excluding zero), i.e. the built-in basic type **`nat1`** in VDM-SL. For this particular example, we say that **`nat1`** is the *domain type* of `Amount`. We further constrain `Amount` using an invariant, by requiring a value of this type to be less than 2000. Specifically, for the invariant shown in Listing 2.1, the `a` on the left-hand side of the equality is a pattern that matches values of type `Amount`. This pattern is used to express the invariant predicate for this named invariant type.

```
types
Amount = nat1
inv a == a < 2000;
```

Listing 2.1: Example of a VDM-SL named invariant type.

2.2.1.1 Functional descriptions

In VDM, functionality can be defined in terms of functions and operations over data types with a traditional call-by-value semantics. Functions are referentially transparent and therefore they are not allowed to access or manipulate state directly, whereas operations are. Therefore, a function cannot call an operation.[1] In addition to accessing module state, operations may also use the **`dcl`** statement to declare local state designators which can be assigned to. Subsequently the term *functional description* will be used to refer to both functions and operations. As an example, a function that uses the `MATH`sqrt` library function to calculate the square root of a real number is shown in Listing 2.2.

```
sqrt :  real -> real
sqrt (x) == MATH`sqrt(x)
pre x >= 0
post RESULT * RESULT = x;
```

Listing 2.2: VDM-SL function for calculating the square root of a number.

Functional descriptions can be implicitly defined in terms of pre- and postconditions, which specify conditions that must hold before and after invoking the functional description. Alternatively, a functional description can be *explicitly* defined by means of an algorithm, as shown in

[1]With the recent introduction of **`pure`** operations into VDM-10 (not to be confused with **`pure`** methods in JML) it has become possible to invoke operations, albeit **`pure`** ones, from a function. This feature was introduced to address issues with the object-oriented dialect of VDM, called VDM++, but was made available in every VDM-10 dialect (including VDM-SL).

Listing 2.2. The JML translator supports both implicitly and explicitly defined functional descriptions. However, only methods that originate from explicitly defined functional descriptions can be executed.

The precondition of a function can refer to all the arguments of the function it guards. The same applies to the postcondition of a function, which can also refer to the result of the execution using the reserved word **RESULT**. For the square root function in Listing 2.2 we require that the input is a positive number (the precondition), and that the square of the function result equals the input value (the postcondition).

Function definitions are derived for the pre- and postconditions of `sqrt` from `sqrt`'s **pre** and **post** clauses. These function definitions do not appear in the model, but they are used internally by the Overture interpreter to check for contract violations. However, to clarify, the pre- and postcondition functions of `sqrt` are shown in Listing 2.3. In this listing, `+>` specifies that `pre_sqrt` and `post_sqrt` are total functions, and not partial functions, which use the `->` type constructor.

```
pre_sqrt:real +> bool
pre_sqrt(x) == x >= 0;

post_sqrt:real*real +> bool
post_sqrt(x,RESULT) == RESULT * RESULT = x;
```

Listing 2.3: Pre- and postcondition functions for the `sqrt` function shown in Listing 2.2.

Similarly, the pre- and postcondition functions of an operation are also derived. To demonstrate this, consider the `inc` operation in Listing 2.4. This operation takes a real number as input, adds it to a counter (defined using a state designator), and returns the new counter value. In this listing `counter~` and `counter` refer to the counter values before and after the operation has been invoked, respectively.

```
inc : real ==> real
inc (i) == (
 counter := counter + i;
 return counter;
)
pre i > 0
post counter = counter~ + i and
     RESULT = counter;
```

Listing 2.4: VDM-SL operation for incrementing a counter.

A precondition of an operation can refer to the state, `s`, before executing the operation, whereas the postcondition of an operation can read both the before and after states. State access is achieved by passing copies of the state to the pre- and postcondition functions. The corresponding pre- and postcondition functions for `inc` are shown in Listing 2.5 where the parameters `s~` and `s` of `post_inc` refer to the state (that contains the counter value) before and after execution of `inc`. We further use `S` to denote the record type that represents the module's state.

```
pre_inc:real*S +> bool
pre_inc(i,s) == i > 0;

post_inc:real*real*S*S +> bool
post_inc(i,RESULT,s~,s) ==
s.counter = s~.counter + i and
RESULT = s.counter;
```

Listing 2.5: Pre- and postcondition functions for the `inc` operation shown in Listing 2.4.

The function descriptions in Listing 2.5 assume that the pre- and postconditions are defined (using the **`pre`** and **`post`** clauses) and that the state of the module enclosing the functional description exists. For the cases where pre- and postconditions are not defined they can be thought of as functions that yield **`true`** for every input. Furthermore, when no state component is defined, the pre- and postcondition functions simply omit the state parameters. Similarly, when an operation does not return a result (it specifies void as the return type) the postcondition function omits the **`RESULT`** parameter.

For each type definition constrained by an invariant, such as `Amount` shown in Listing 2.1, a function is implicitly created to represent the invariant – see Listing 2.6. The Overture tool uses this function internally to check whether a value is consistent with respect to a given type (e.g. `Amount`) [23]. Note that since all invariants are functions they are not allowed to depend on state of other modules. Specifically, invariants can only invoke functions and access global constants (possibly defined in other modules).

```
inv_Amount : Amount +> bool
inv_Amount (a) == a < 2000;
```

Listing 2.6: Invariant function for type definition `Amount`.

2.2.1.2 Atomic execution

Multiple consecutive statements are sometimes needed to update the state designators to make them consistent with the system's invariants. For example, assume that we have a system that uses two state designators called $\texttt{evenID}_1$ and $\texttt{evenID}_2$ to store even and different numbers. For this example, we will assume that these state designators are of type `Even` – a type that constrains these state designators to store even numbers. To help ensure that the uniqueness constraint (a state invariant) is not violated during an update, multiple assignments can be grouped in an **`atomic`** statement block as shown in Listing 2.7. Given the type `Even` of the state designators $\texttt{evenID}_1$ and $\texttt{evenID}_2$ it is as if the atomic statement is evaluated as shown in Listing 2.8.

```
atomic (
 evenID1 := exp1;
 evenID2 := exp2;
)
```

Listing 2.7: Atomic update in VDM.

```
let t1 : Even = exp1,
    t2 : Even = exp2
in (
 -- Turn off invariants
 evenID1 := t1;
 evenID2 := t2;
 -- Turn on invariants
 -- Check invariants hold
);
```

Listing 2.8: The execution semantics of the **atomic** statement.

Executing the **atomic** statement block is semantically equivalent to first evaluating the right-hand sides of all the assignments before turning off invariant checks, and then binding the results to the corresponding state designators. After all the assignments have been executed, it must be ensured that all invariants hold.

There are three properties that follow from the evaluation semantics of the **atomic** statement block that are worth mentioning:

1. When evaluating the right-hand sides of the assignment statements, potential contract violations will be reported.
2. Temporary identifiers, used to store the right-hand side results, are explicitly typed and therefore violations of named invariant types for these variables will be reported. The explicit type annotations thus ensure that the right-hand side of a state designator assignment is checked to be consistent with the type of said state designator.
3. Assignment statements cannot see intermediate values of state designators.

2.2.2 JML

Although JML [24] is designed to specify arbitary sequential Java programs, in this subsection we only describe the features needed for the translation from VDM-SL.

A method specified with the **pure** modifier in JML is not permitted to have write effects; such methods are allowed to be used in specifications. Pure methods are used to translate VDM-SL functions.

A class invariant in JML should hold whenever the non-helper methods of that class are not being executed; thus invariants must hold in each method's before and after states. However, a method declared with the **helper** annotation in a type `T` does not have its pre- and postconditions augmented with `T`'s invariants. Helper methods (and constructors) must either be pure or private [24], so that the invariant will hold at the beginning and end of all client-visible methods [28]. The before and after states of non-helper methods and constructors are said to be *visible states*; thus invariants must hold in all visible states. JML distinguishes between instance and static invariants. An *instance* invariant can refer to the non-static (i.e. instance) fields of an object. A *static* invariant cannot refer to an object's non-static fields; thus static invariants are used to specify properties of static fields.

An assertion can reference the invariant for an object explicity using a predicate of the form **\invariant_for**`(e)`, which is equivalent to the invariant for `e`'s static type [24, section 12.4.22].

In JML pre- and postconditions are written using the keywords **requires** and **ensures**, respectively. In the specification of a postcondition, one writes `\`**`old`**`(e)` to refer to the before state value of an expression `e`. For example, an `increment` method that writes a field `count` could be specified as shown in Listing 2.9.

```
//@ requires count < Integer.MAX_VALUE;
//@ modifies count;
//@ ensures count == \old(count)+1;
void increment() {
  count++;
}
```

Listing 2.9: Example of a JML specification for a Java method.

Method postconditions may also use the keyword **`\result`** to refer to the value returned by the method.

Specification expressions in JML can use Java expressions that are pure (have no write effects), and also some logical operators, such as implication ==>, and quantifiers such as **`\forall`** and **`\exists`**.

In addition to method pre- and postconditions, one can also write assertions anywhere a Java statement can appear, using JML's **`assert`** keyword. Such assertions must hold whenever they are executed.

One way to specify the abstract state of a class is to use JML's **`ghost`** variables. Ghost variables are specification-only variables and fields of objects that can only be used in JML specifications and in JML **`set`** statements. A set statement is an assignment statement whose target is a ghost variable.

By default, JML variables and fields may not hold the **`null`** value. However, should one wish to specify that all fields of a class may hold **`null`**, then one can annotate the class's declaration with **`nullable_by_default`**.

2.3 The implementation of the JML translator

The JML translator is implemented as an extension to Overture's VDM-SL-to-Java code-generator, which provides code-generation support for a large executable subset of VDM. This section describes how the JML translator has been implemented, and explains the details of the Java code-generator that are needed in order to understand how the JML translator works.

2.3.1 The implementation

The Java code-generator is developed using Overture's code-generation platform – a framework for constructing code-generators for VDM [16]. This platform is used by the Java code-generator to parse the VDM-SL model sources and to construct an Intermediate Representation (IR) of the model – an Abstract Syntax Tree (AST) that constitutes an internal representation of the generated code. The Java code-generator uses the code-generation platform to *transform* the IR into a tree structure that eventually is translated directly into Java code. The translation of the IR into Java is handled by the code-generation platform's code emission framework, which uses the Apache Velocity template engine [4].

The Java code-generator exposes the IR during the code-generation process, which allows the JML translator to intercept the code-generation process and further transform the IR. These additional transformations are used to decorate the IR with nodes that contain the JML annotations. Using the code emission framework, the final version of the IR is translated into a JML-annotated Java program.

The JML translator is publicly available in Overture version 2.3.8 (as of July 2016) onwards [30]. Furthermore, the JML translator's source code is available via the Overture tool's open-source code repository [31].

2.3.2 Overview of the translation

In the generated code, a module is represented using a **`final`** Java class with a **`private`** constructor, since VDM-SL does not support inheritance and a module cannot be instantiated. Due to the latter, both operations and functions are code-generated as **`static`** Java methods.

Module state is represented using a **`static`** class field in the module class to ensure that only a single state component exists at any given time. The state component is represented using a record value, and as a consequence, an additional record type is generated to represent it.

Each variable in VDM-SL is passed by value, i.e. as a *deep copy*, when it is passed as an argument, appears on the right-hand side of an assignment or is returned as a result. As a consequence, aliasing can never occur in a VDM-SL model. Types are different in Java, where objects are modified via object references or pointers. Therefore different object references can be used to modify the same object. To avoid such aliasing in the generated code, data types are code-generated with functionality to support value type behaviour.

Every record definition code-generates to a class definition with accessor methods for reading and manipulating the fields. This class implements `equals` and `copy` methods to support comparison based on structural equivalence and deep copying, respectively. In this way the call-by-value semantics of VDM-SL can be preserved in the generated code by invoking the `copy` method, which helps to prevent aliasing. Similarly the `equals` method can be invoked to compare code-generated records based on structural equivalence rather than comparing addresses of object references. A record object can then be obtained by invoking the constructor of the record class or by invoking the `copy` method of an existing record object.

Java does not support the definition of aliases of existing types, such as the `Amount` named invariant type in Listing 2.1. Therefore, the Java code-generator chooses not to code-generate class definitions for these types. Instead, a use of a named invariant type is replaced with its domain type (described in subsection 2.2.1). Since the named invariant type is an alias of an existing type this is fine, as long as we make sure to check that the type invariant holds.

To assist the translation of VDM to Java, the existing Java code-generator uses a runtime library, which among other things, includes Java implementations for some of the different VDM types and operators. The `Tuple` class, for example, is used to represent tuple types and enables construction of tuple values. Sets, sequences and maps are represented using the `VDMSet`, `VDMSeq` and `VDMMap` classes, which themselves are based on Java collections, and so on. The runtime library's collection classes are used as raw types (e.g. `VDMSet`) in the generated code, and therefore they are never passed a generic type argument. Raw types provide a convenient way to represent VDM collections that store elements of some union type – a kind of type that Java does not support.

In addition to using the existing runtime library, the JML translator also contributes a small runtime library to aid the generation of JML checks. This runtime library, which we subsequently refer to as `V2J`, is an extension of the existing Java code-generator runtime library. As we shall see in subsection 2.6.6, the `V2J` runtime is mostly used in the generated JML checks to ensure that instances of collections respect the VDM types that produce them.

2.4 Case study example

Throughout the report we will demonstrate the translation rules using a case study model of an ATM. The model consists of a single module, `ATM` (shown in Listing 2.10), which uses a state definition to record information about

- The debit cards considered valid by the system (named `validCards`).
- The debit card currently inserted into the ATM, if any (`currentCard`).
- If a valid PIN code has been entered (`pinOk`) for the debit card currently inserted into the ATM and,
- all the bank accounts known to the system (`accounts`).

```
module ATM
definitions
state St of
 validCards : set of Card
 currentCard : [Card]
 pinOk : bool
 accounts : map AccountId to Account
 init St == St = mk_St({},nil,false,{|->})
 inv mk_St(v,c,p,a) ==
  (p or c <> nil => c in set v)
  and
  forall id1, id2 in set dom a &
   id1 <> id2 =>
   a(id1).cards inter a(id2).cards = {}
end
 ...
operations
GetStatus : () ==> bool * seq of char
GetStatus () == ...

OpenAccount : set of Card * AccountId ==> ()
OpenAccount (cards,id) == ...

AddCard : Card ==> ()
AddCard (c) == ...

RemoveCard : Card ==> ()
RemoveCard (c) == ...

InsertCard : Card ==>
  <Accept>|<Busy>|<Reject>
InsertCard (c) == ...

EnterPin : Pin ==> ()
EnterPin (pin) == ...

ReturnCard : () ==> ()
ReturnCard () == ...

Withdraw : AccountId * Amount ==> real
Withdraw (id, amount) == ...

Deposit : AccountId * Amount ==> real
Deposit (id, amount) == ...
end
```

Listing 2.10: VDM-SL module representing an ATM.

For simplicity, Listing 2.10 omits type definitions and only shows the state definition (including the state invariant) and the signatures for some of the operations. The state invariant, shown in Listing 2.10, requires that at all times the following two conditions must be met: a debit card must at most be associated with a single account and secondly, for a PIN code to be considered valid, the debit card currently inserted into the ATM must itself be a valid debit card.

When the ATM model is translated to a JML-annotated Java program it can be checked for correctness using JML tools. To demonstrate this, consider the example in Listing 2.11, which creates a debit card, inserts it into the ATM, and performs a transaction scenario.

```
Card c = new Card(5,1234);
// atm.ATM.AddCard(c); (missing statement)
atm.ATM.InsertCard(c);
atm.ATM.EnterPin(1234);
System.out.println(atm.ATM.GetStatus());
/* Transaction related code omitted */
atm.ATM.ReturnCard();
```

Listing 2.11: Java code demonstrating use of the implementation of the ATM model.

If this program is executed using the OpenJML runtime assertion checker the output in Listing 2.12 is reported.

```
Exception in thread "main" java.lang.AssertionError: Main.java:12: JML
    precondition is false
         atm.ATM.EnterPin(1234);
                         ^
atm/ATM.java:276: Associated declaration: Main.java:12:
  //@ requires pre_EnterPin(pin,St);
      ^
         at Main.main(Main.java:17)
```

Listing 2.12: Inconsistent use of the system detected using the OpenJML runtime assertion checker.

For this particular example, this error is reported because the debit card `c` is not recognised as a valid debit card by the system. Specifically, the scenario did not invoke `atm.ATM.AddCard(c)` immediately after creating the debit card. The return value of the `Insert` method did indicate that the debit card was rejected, but this value was mistakenly discarded in Listing 2.11. The error is reported by the runtime assertion checker because entering a PIN code when no debit card is inserted into the ATM is considered an error. After changing the example in Listing 2.11 to add `c` as a valid debit card, no problems are detected by the runtime assertion checker, as expected. Therefore, the code executes as if it was compiled using a standard Java compiler and executed on a regular Java virtual machine. More, specifically, the system will report the status as shown in Listing 2.13 to indicate that the ATM is not awaiting a debit card, and that a transaction is in progress.

```
mk_(false, "transaction in progress.")
```

Listing 2.13: System output after fixing the problem in Listing 2.11.

As we proceed, in section 2.5 and section 2.6 we elaborate on the specifics of each VDM definition in the case study model and demonstrate the translation to JML-annotated Java.

2.5 Translating VDM-SL contracts to JML

In this section we present the rules used to translate the DbC elements of VDM-SL to JML annotations that are added to the generated Java code. For each of the elements, we describe the approach used to translate the element to JML. This is afterwards generalised as a rule, which appears in a grey box.

2.5.1 Allowing null values by default

Overture's Java code-generator may sometimes introduce auxiliary variables that are initialised to **`null`** when it code-generates some of the constructs of VDM. To avoid having errors reported when checking the generated code with a JML tool, we allow **`null`** as a legal value by default for all references in the generated code.

1. Allowing null values by default
Annotate every class output by the Java code-generator with the **`nullable_by_default`** modifier to allow all references to use **`null`** as a legal value.

As a consequence we also have to guard against **`null`** values for variables that originate from VDM variables or patterns[2] that do not allow **`nil`**.

2.5.2 Translating functional descriptions to JML

Recall that a VDM-SL function code-generates to a **`static`** Java method. In addition, a VDM-SL function does not have side-effects and therefore the code-generated version of the method can be annotated as JML **`pure`**.

2. Translation of functions
Any function – whether it is defined by the user or derived, e.g. from a **`pre`** or **`post`** condition clause – code-generates to a **`static`** Java method that is annotated with the **`pure`** modifier.

Operations, on the other hand, can read and manipulate the state of the enclosing module, or invoke other operations that may have side-effects. Therefore, the method that the operation code-generates to cannot be annotated as JML **`pure`**.

When a VDM-SL definition (e.g. a functional description) is code-generated to Java, the visibility of the corresponding Java definition can, in principle, be set according to whether the VDM-SL definition is exported (**`public`**) or not (**`private`**). In the presentation of the translation rules following this section, we omit explicit use of access specifiers in the rule formulation as we do not consider it crucial to our work.

2.5.3 Translating preconditions to JML

In terms of semantics there is no difference between a precondition in VDM-SL and JML. There are, however, interesting issues worth mentioning regarding how the JML translator implements the translation. We start by covering preconditions of operations, and we end this subsection by describing how they differ from those of functions. As an example of how a VDM-SL precondition is translated, consider the operation in Listing 2.14. This operation models withdrawal from a bank account identified by the parameter `id`.

[2]The generated code uses variables to represent the patterns (record pattern, tuple pattern, identifier pattern etc.) introduced by use of pattern matching in VDM.

```
Withdraw : AccountId * Amount ==> real
Withdraw (id, amount) ==
let newBalance =
    accounts(id).balance - amount
in (
 accounts(id).balance := newBalance;
 return newBalance;
)
pre
currentCard in set validCards and pinOk and
currentCard in set accounts(id).cards and
id in set dom accounts
```

Listing 2.14: VDM-SL operation for bank account withdrawal guarded by a precondition.

In order to withdraw money from the account, we require that a valid card has been inserted, the PIN code is accepted, and that the bank account exists. Note that since `currentCard` is of the optional type `[Card]` it can be **`nil`**, which is not a valid member of `validCards`. Therefore, the precondition is **`false`** when no debit card has been inserted into the ATM. The `pre_Withdraw` function, which is not a visible part of the model, is derived from the **`pre`** clause of the `Withdraw` operation. In the generated code this function is represented using a **`pure`** method according to rule 2 – see Listing 2.15. Note that for the method in Listing 2.15, the Java code-generator uses extra variables to perform the equivalent VDM computation. These extra variables are also type checked using JML (although they are only used to store intermediate results).

The `Withdraw` operation is translated to the method shown in Listing 2.16. This method introduces several JML assertions that will be described in section 2.6. Note that the method `pre_Withdraw` is invoked from the **`requires`** clause of the `Withdraw` method to check whether the precondition is met. In addition to the input parameters of the `Withdraw` method, the `pre_Withdraw` method is also passed the state `St`.

3. Translating the precondition of an operation

Let `op` be a method code-generated from a VDM-SL user-defined operation and let the signature of `op` be:
 `static` `R op(`I_1 i_1`,...,`I_n i_n`)`
 Then `op` has a code-generated precondition method `pre_op` that is **`pure`** and which in addition to the parameters of `op` also takes the state component `s` as an argument, i.e.
 `/*@` **`pure`** `@*/` **`static boolean`**
 `pre_op(`I_1 i_1`,...,`I_n i_n`,S s)`
 To ensure that the precondition is evaluated, we annotate `op` with the following **`requires`** annotation:
 `//@` **`requires`** `pre_op(`i_1`,...,`i_n`,s);`

Rule 3 assumes the existence of a state component `s`. However, when the state of the module enclosing `op` is not defined, rule 3 changes to not include the state parameter in the definition of `pre_op`.

The example above considers the case where the precondition is guarding an operation (i.e. `Withdraw`). As described in section 2.2, a precondition is defined differently for a function than it is for an operation. In particular, the precondition of a function is not passed the state, so neither is the code-generated version of it. We also note that the visibility of the precondition function

```
/*@ pure @*/
public static Boolean pre_Withdraw(
    final Number id, final Number amount, final atm.ATMtypes.St St) {
  //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));
  //@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));
  //@ assert Utils.is_(St,atm.ATMtypes.St.class);
  Boolean andResult_3 = false;
  //@ assert Utils.is_bool(andResult_3);
  if (SetUtil.inSet(St.get_currentCard(), St.get_validCards())) {
    Boolean andResult_4 = false;
    //@ assert Utils.is_bool(andResult_4);
    if (St.get_pinOk()) {
      Boolean andResult_5 = false;
      //@ assert Utils.is_bool(andResult_5);
      if (SetUtil.inSet(St.get_currentCard(),
          ((atm.ATMtypes.Account) Utils.get(St.accounts, id)).get_cards()))
              {
        if (SetUtil.inSet(id, MapUtil.dom(St.get_accounts()))) {
          andResult_5 = true;
          //@ assert Utils.is_bool(andResult_5);
        }
      }
      if (andResult_5) {
        andResult_4 = true;
        //@ assert Utils.is_bool(andResult_4);
      }
    }
    if (andResult_4) {
      andResult_3 = true;
      //@ assert Utils.is_bool(andResult_3);
    }
  }
  Boolean ret_29 = andResult_3;
  //@ assert Utils.is_bool(ret_29);
  return ret_29;
}
```

Listing 2.15: Code-generated version of the `pre_Withdraw` operation.

must be the same as that of the functional description it guards. Otherwise it cannot be invoked from the corresponding **requires** clause.

4. Translating the precondition of a function

Let `f` be a method code-generated from a VDM-SL user-defined function and let the signature of `f` be:

static R f(I_1 i_1,...,I_n i_n)

Then `f` has a code-generated precondition method `pre_f` that is **pure** and which accepts the same parameters as `f`, i.e.

/*@ **pure** @*/ **static boolean**
pre_f(I_1 i_1,...,I_n i_n)

To ensure that the precondition is evaluated, we annotate `f` with the following **requires** annotation:

//@ **requires** pre_f(i_1,...,i_n);

```
//@ requires pre_Withdraw(id,amount,St);
public static Number Withdraw(final Number id, final Number amount) {
 //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));
 //@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));
 final Number newBalance =
     ((atm.ATMtypes.Account) Utils.get(St.accounts, id)).get_balance().
         doubleValue()
         - amount.longValue();
 //@ assert Utils.is_real(newBalance);
 {
   VDMMap stateDes_1 = St.get_accounts();
   atm.ATMtypes.Account stateDes_2 = ((atm.ATMtypes.Account) Utils.get(
       stateDes_1, id));
   //@ assert stateDes_2 != null;
   stateDes_2.set_balance(newBalance);
   //@ assert (V2J.isMap(stateDes_1) && (\forall int i; 0 <= i && i < V2J.
       size(stateDes_1); (Utils.is_nat(V2J.getDom(stateDes_1,i)) &&
       inv_ATM_AccountId(V2J.getDom(stateDes_1,i))) && Utils.is_(V2J.getRng(
       stateDes_1,i),atm.ATMtypes.Account.class)));
   //@ assert Utils.is_(St,atm.ATMtypes.St.class);
   //@ assert \invariant_for(St);
   Number ret_7 = newBalance;
   //@ assert Utils.is_real(ret_7);
   return ret_7;
 }
}
```

Listing 2.16: Code-generated version of the `Withdraw` operation.

2.5.4 Translating postconditions to JML

Postconditions in VDM-SL and JML are semantically similar, although VDM-SL represents the postcondition function as a derived function definition (as was done for preconditions). Furthermore, in VDM and JML postconditions of operations and methods, respectively, can access both the before and after states. Returning to the `Withdraw` operation, one could specify a postcondition requiring that exactly the value specified by the `amount` parameter is withdrawn from the account – see Listing 2.17.

```
Withdraw : AccountId * Amount ==> real
Withdraw (id, amount) == ...
post
let accountPre = accounts~(id),
    accountPost = accounts(id)
in
 accountPre.balance =
 accountPost.balance + amount and
 accountPost.balance = RESULT;
```

Listing 2.17: The `Withdraw` operation guarded by a postcondition.

The JML translator produces a **`pure`** Java method to represent the postcondition function. This Java method is invoked from the **`ensures`** clause to check that the postcondition holds. The invocation of the postcondition method of the `Withdraw` operation is shown in Listing 2.18.

```
//@ requires pre_Withdraw(id,amount,St);
//@ ensures post_Withdraw(id,amount,\result,\old(St.copy()),St);
```

```
public static Number Withdraw(final Number id, final Number amount) {...}
```

Listing 2.18: Code-generated version of the `Withdraw` operation.

Note in particular how the before and after states are passed to the `post_Withdraw` method. Reasoning about before state is achieved using JML's **`\old`** expression. For the `Withdraw` operation the before state is constructed as **`\old`**`(St.copy())`. Since `St.copy()` is a deep copy of the state (as explained in section 2.3) the evaluation inside the **`\old`** expression ensures that the result indeed is a representation of the before state.

The JML translator deep copies the state because Java represents every composite data type using a class. So without deep copying the state, only the address of the before state object reference is copied. In effect, only a single object would exist to represent the pre- and post states. This would never work, since state changes made by the operation would affect what was intended to be a representation of the before state. Therefore, the state is deep copied to get a separate object to represent the before state.

5. Translating the postcondition of an operation

Let `op` be a method code-generated from a VDM-SL user-defined operation and let the signature of `op` be:
`static` `R op(`I_1 i_1`,...,`I_n i_n`)`
Then `op` has a code-generated postcondition method `post_op` that is **`pure`** and which in addition to the parameters of `op`, also takes the result and the before and after states of `op` as arguments, i.e.
`/*@` **`pure`** `@*/` **`static boolean`**
`post_op(`I_1 i_1`,...,`I_n i_n`,`
   `R RESULT,S _s,S s)`
To ensure that the postcondition is evaluated we annotate `op` with the following **`ensures`** annotation:
`//@` **`ensures`** `post_op(`i_1`,...,`i_n`,`**`\result`**`,`
   `\old``(s.copy()),s);`

While the primary concern is to preserve the behaviour of the specification across the translation, deep copying values may significantly affect system performance in a negative way. In particular, because it is difficult (in general) to avoid deep copying values unnecessarily when they are passed around in the generated code. To address this issue, the Java code-generator (and hence the JML translator) offers an option that, when selected by the user, omits deep copying of values (other than the old state). While the purpose of this is to generate performance-efficient code, this option is, however, only safe to use if Java objects that originate from VDM-SL values are not modified via aliases.

Yi et al. identify and address a number of problems with the **`\old`** expression [40]. In particular, the authors of that work conduct experiments showing that passing deep copies of the old state may drastically increase a system's memory usage. To address this, Yi et al. propose the **`\past`** expression as a more memory-efficient alternative to the **`\old`** expression. Yi et al. further show that the **`\past`** expression can be implemented as an extension of the OpenJML runtime assertion checker by means of aspect-oriented programming principles. However, OpenJML does not officially support the **`\past`** expression yet, which is why the translation rules do not currently rely on this expression. As we see it, the ideas proposed by Yi et al. could potentially support the development of a more performance-efficient way to handle old states in the JML translator.

Similar to rule 3, rule 5 also assumes that the state of the module enclosing `op` exists. If the state component does not exist, rule 5 changes to not include the state parameters in the definition of `post_op`. Furthermore, if `op` does not return a result (the return type is void), then the definition of `post_op` does not include the `RESULT` parameter.

The example above considers the postcondition of an operation (i.e. `Withdraw`). As described in section 2.2, the postcondition of a function is not allowed to access state. Therefore, the code-generated version of the postcondition function is not passed the state.

6. Translating the postcondition of a function

Let `f` be a method code-generated from a VDM-SL user-defined function and let the signature of `f` be:
`static` `R f(`I_1 i_1`,...,`I_n i_n`)`
Then `f` has a code-generated postcondition method `post_f` that is **`pure`** and which in addition to the parameters of `f` also takes the result of `f` as an argument, i.e.
`/*@` **`pure`** `@*/` **`static boolean`**
`post_f(`I_1 i_1`,...,`I_n i_n`,R RESULT)`
To ensure that the postcondition is evaluated we annotate `f` with the following **`ensures`** annotation:
`//@` **`ensures`** `post_f(`i_1`,...,`i_n`,\`**`result`**`);`

2.5.5 Translating record invariants to JML

A record can, like any other type definition in VDM-SL, be constrained by an invariant. As an example, Listing 2.19 shows a record definition modelling a bank account.

```
Account ::
 cards : set of Card
 balance : real
 inv a == a.balance >= -1000;
```

Listing 2.19: A VDM-SL record definition modelling a bank account.

An `Account` comprises the available balance as well as the debit cards associated with the account. We further constrain an `Account` to not have a balance of less than -1000, which is expressed using an invariant.

As described in section 2.3, a record definition is translated to a class that emulates the behaviour of a value type using `copy` and `equals` methods.

Since a record invariant is required to hold for every record value, or object instance in the generated code, we represent it using an **`instance invariant`** in JML as shown in Listing 2.20. Note that the **`instance invariant`** is formulated as an implication such that invariant violations are not reported when invariant checks are disabled. As we shall see in subsection 2.5.6 this has to do with the way VDM-SL handles atomic execution.

The code-generated record `Account` defines an *invariant method* `inv_Account` that takes all the record fields of `Account` as input and evaluates the invariant predicate. This method is invoked directly from the JML invariant, as shown in Listing 2.20. Note that `inv_Account` is a **`static`** method according to rule 2. In addition, this method is annotated as a **`helper`** to avoid the invariant check triggering another invariant check, which eventually would cause a stack-overflow.

```
//@ nullable_by_default
final public class Account implements Record
{
 public VDMSet cards;
 public Number balance;
 //@ public instance invariant atm.ATM.invChecksOn ==> inv_Account(cards,
     balance);
  ...
 /*@ pure @*/
 public boolean equals(final Object obj)
 {...}
 /*@ pure @*/
 public atm.ATMtypes.Account copy(){...}
 /*@ pure @*/
 public VDMSet get_cards() {...}
 public void set_cards(final VDMSet _cards)
 {...}
 /*@ pure @*/
 public Number get_balance() {...}
 public void set_balance(final Number _balance) {...}
 /*@ pure @*/
 /*@ helper @*/
 public static Boolean inv_Account(final VDMSet _cards, final Number
     _balance){
   return _balance.doubleValue() >= -1000L;
 }
}
```

Listing 2.20: Code-generated version of the `Account` record.

7. Translating a record invariant

Let `D` be a code-generated record definition with fields $f_1, \ldots, f_n$ of types $F_1, \ldots, F_n$, respectively, and let `D` be constrained by an invariant. Then `D` has an invariant method `inv_D` that is annotated as a **`helper`** to allow it to be invoked from the invariant clause of `D`. The invariant method can also be annotated as **`pure`** since it originates from a function definition. The annotated signature of `inv_D` thus becomes:

```
/*@ pure @*/
/*@ helper @*/
boolean inv_D(F1 f1,...,Fn fn)
```

Let further `invChecksOn` be a variable that is true if invariant checking is enabled and false otherwise. To represent the record invariant of `D` we annotate `D` with the **`invariant`** annotation:

```
/*@ public instance invariant
invChecksOn ==> inv_D(f1,...,fn); @*/
```

As we shall later see in subsection 2.5.6, atomic execution sometimes requires extra assertions to be inserted into the generated code in order to guarantee that the record invariant semantics of VDM-SL are preserved.

All the methods inside a record class – except for the constructor and the “setter” methods – do not modify the state of the record class and therefore they are marked as **`pure`**. Updates to a record object in the generated code are made using the “setter” methods of the generated record class, or by using the record modification expression [21]. Use of “setter” methods instead of direct field access to manipulate the state of a record (which is how field access is achieved in VDM-SL) forces the record object into a *visible state* (as described in subsection 2.2.2) after it has been updated, thus triggering the invariant check according to the VDM-SL semantics. For example, in VDM-SL we could set the balance of an account as shown in Listing 2.21.

```
acc.balance := newBalance;
```

Listing 2.21: Updating the `Account` balance in VDM-SL.

This assignment produces the Java code shown in Listing 2.22. Note that for this particular case there is no need to generate any additional JML assertions since the state of `acc` becomes visible after the call to `set_balance`. This causes the invariant check of `Account` to trigger.

```
acc.set_balance(newBalance);
```

Listing 2.22: Updating the `Account` balance in the generated code.

2.5.6 Atomic execution

There are situations where multiple assignment statements in VDM-SL need to be evaluated atomically in order to avoid unintentional violation of a state invariant. In our example, this is the case when the ATM returns the card to the owner, which is done as the last step of a transaction. Returning the debit card also requires us to invalidate the PIN code currently entered. These two things have to be done atomically to avoid violating the state invariant of the `ATM` module, which is checked using the `inv_St` function, derived from the state invariant shown in Listing 2.10 in section 2.4. Therefore the body of the `ReturnCard` operation is executed inside an **`atomic`** statement block as shown in Listing 2.23. Note that the invariant is evaluated internally by the interpreter, and therefore the example in Listing 2.23 makes no explicit mention of the invariant.

```
ReturnCard : () ==> ()
ReturnCard () ==
atomic (
 currentCard := nil;
 pinOk := false;
)
pre currentCard <> nil
post currentCard = nil and not pinOk;
```

Listing 2.23: Removal of the debit card from the ATM in VDM-SL.

JML does not include a syntactic construct similar to that of the **`atomic`** statement. Instead atomic execution must be achieved using different means – for example by manipulating state directly using field access or **`helper`** methods.

To be consistent with the way record state is updated, and to reflect the way that VDM-SL handles atomic execution, we believe a better approach is to use a flag that indicates if invariant checks are enabled or not. Since this flag should not affect the generated code, we make it a **`ghost`** field such that it is only visible at the specification level. Since this **`ghost`** field must be accessible everywhere in the translation, we make it a static field of the class, as shown in Listing 2.24. The **`ghost`** field must be added to one of the generated Java classes since Java does not really have global variables. Note that this flag does not affect pre- and postconditions since these checks must always be evaluated.

```
/*@ public ghost static boolean invChecksOn = true; @*/
```

Listing 2.24: Ghost field used to control invariant checking.

The declaration of `invChecksOn` allows us to formulate invariants such that violations are reported only if invariant checking is enabled. An example of this is shown in Listing 2.25 for the record state class of the `ATM` module.

```
//@ public instance invariant atm.ATM.invChecksOn ==> inv_St(validCards,
    currentCard,pinOk,accounts);
```

Listing 2.25: The invariant of the record state class.

The `invChecksOn` flag provides the means to emulate the behaviour of atomic execution in a Java environment as shown in Listing 2.26. Specifically, the JML **`set`** statement is used to disable/enable invariant checking before/after executing the body of the `ReturnCard` method.

```
//@ requires pre_ReturnCard(St);
//@ ensures post_ReturnCard(\old(St.copy()),St);
public static void ReturnCard() {
 atm.ATMtypes.Card atomicTmp_1 = null;
 //@ assert ((atomicTmp_1 == null) || Utils.is_(atomicTmp_1,atm.ATMtypes.
     Card.class));
 Boolean atomicTmp_2 = false;
 //@ assert Utils.is_bool(atomicTmp_2);
 { /* Start of atomic statement */
   //@ set invChecksOn = false;
   //@ assert St != null;
   St.set_currentCard(Utils.copy(atomicTmp_1));
   //@ assert St != null;
   St.set_pinOk(atomicTmp_2);
   //@ set invChecksOn = true;
   //@ assert \invariant_for(St);
 } /* End of atomic statement */
}
```

Listing 2.26: Code-generated version of the `ReturnCard` operation.

8. Enabling and disabling invariant checking

Declare in a code-generated module `M` a globally accessible JML **`ghost`** field `invChecksOn` to control invariant checking:

```
/*@ public ghost static
boolean invChecksOn = true; @*/
```

Before executing a code-generated atomic statement (in any of the code-generated modules) invariant checking is disabled using the following JML **`set`** statement:

```
//@ set M.invChecksOn = false;
```

After the code-generated atomic block has finished executing invariant checking is re-enabled using:

```
//@ set M.invChecksOn = true;
```

When all the statements have been executed it must be ensured that no invariants have been violated. For the example in Listing 2.26, the only thing that needs to be checked is that the state component of the `ATM` class, i.e. `St` does not violate its invariant. This is checked by asserting that `\invariant_for(St)` holds.

9. Resuming invariant checking

Let $d_1, \ldots, d_n$ be state designators of records that have been updated, or affected by an update, during execution of a code-generated atomic statement block. Further assume that $d_1, \ldots, d_n$ have been updated in the given order, i.e. d_i was updated (for the first time) before d_{i+1} and that d_i may be of one of m_i record types $D_{i1}, \ldots, D_{im_i}$. Immediately after executing the code-generated atomic statement block, it is checked that the state designators $d_1, \ldots, d_n$ do not violate any invariants using the following sequence of **assert** statements:

```
//@ assert d1 instance of D11 ==>
   \invariant_for((D11) d1);
...
//@ assert d1 instance of D1m1 ==>
   \invariant_for((D1m1) d1);
...
//@ assert dn instance of Dn1 ==>
   \invariant_for((Dn1) dn);
...
//@ assert dn instance of Dnmn ==>
   \invariant_for((Dnmn) dn);
```

The **\invariant_for** construct is not currently implemented in OpenJML. Instead this check can be inlined as a method call (rather than explicitly using **\invariant_for**). However, throughout this report we use **\invariant_for** to check record invariants as we believe it makes the examples easier to understand.

The JML translator keeps track of state designators of records that potentially have been updated as part of executing the code-generated atomic statement block. This is done by analysing the left-hand sides of the assignment statements. Immediately after invariant checking is re-enabled, i.e. the code-generated atomic statement block has finished execution, it is checked that no record violates its invariant.

There are a few things related to rule 9 that are worth clarifying. First, for assignments to composite state designators such as `a.b.c:=42`, the invariants of the individual state designators `a`, `b` and `c`, have to be checked, if these are defined. For this particular example we say that `c` was updated, and that `a` and `b` were affected by the update. Second, the order in which the invariants are checked follows that used by the Overture VDM interpreter. Third, regardless of how many times a state designator is updated, the corresponding invariant is only checked once (for each state designator) since this is how atomic execution works in VDM, i.e. the update(s) are performed atomically, and afterwards the constraints that the state designators are subjected to are checked. Fourth, rule 9 includes all the state designators that have been updated or affected by an update. No particularly complex situations can arise that makes it difficult to identify these state designators since all VDM-SL's data types use call-by-value semantics, and therefore no aliasing can occur. Essentially this means that for the assignment statement `a.b.c:=42`, the only invariants (if defined) that have to be checked are those of the state designators `a`, `b` and `c` since aliases do not exist. Therefore, the JML translator can determine (using static analysis) that assertions only have to be generated for these state designators. Naturally this simplifies the translation process, since the JML translator does not have to identify additional state designators (other than those that appear on the left-hand side of the assignment) that are affected by the assignment.

A state designator can be "masked" as a union type and in such situations it cannot always be statically determined what the runtime type of a state designator will be. To demonstrate this, consider the record types `R1` and `R2` and a state designator declared as **`dcl`** `r : R1 | R2 := ...`. Further assume that `R1` and `R2` code-generate to classes $\texttt{R1}_\texttt{C}$ and $\texttt{R2}_\texttt{C}$. After updating `r` atomically in the generated code, it is ensured that `\`**`invariant_for`**$\texttt{((R1}_\texttt{C}\texttt{) r)}$ holds if `r` is of type $\texttt{R1}_\texttt{C}$, and similarly that the equivalent condition is true if `r` is of type $\texttt{R2}_\texttt{C}$. Since rule 9 has to take all possible types into account, the invariant checks are formulated as implications.

Although the VDM type system allows state designators to be "masked" as union types, most of the time it is possible to statically determine the runtime type of a state designator. For example, in Listing 2.26 no **`instanceof`** check is needed since the static type of the state component is `St`. This is an example where the JML translator simplifies the checks proposed by rule 9.

There are more aspects to rule 9 worth discussing – especially when state designators are based on arbitrarily complex data structures such as nested records. These will be addressed in subsection 2.7.1.

2.5.7 Translating module state to JML

As described in subsection 2.2.1, a module state invariant constrains the record type used to represent the state component of the enclosing module. Therefore, a module state invariant can essentially be seen as a record invariant that can be translated into JML-annotated Java without introducing additional translation rules. This subsection instead explains how a VDM-SL state definition is translated into a form that allows the rules related to record invariants to be applied (see subsection 2.5.5).

In our example each account can be accessed from an ATM using one of the debit cards associated with it. In addition to the bank accounts, the state of the ATM also keeps track of the debit cards that the system considers valid, the debit card that is currently inserted into the ATM, and whether the PIN code entered by the user is valid. The state (including the state invariant) as specified in VDM-SL is shown in Listing 2.10 and described in section 2.4. Based on the state definition, a record class is generated that represents the state type as shown in Listing 2.27. Recall that the fields in this class are nullable according to rule 1.

```java
final public class St implements Record {
 public VDMSet validCards;
 public atm.ATMtypes.Card currentCard;
 public Boolean pinOk;
 public VDMMap accounts;

 //@ public instance invariant atm.ATM.invChecksOn ==> inv_St(validCards,
     currentCard,pinOk,accounts);
 /* Record methods omitted */
}
```

Listing 2.27: The record class used to represent the state type.

In addition, an instance of the record class is created to represent the state component as shown in Listing 2.28. The state component is annotated with the **`spec_public`** modifier so that it can be referred to from the **`requires`** and **`ensures`** clauses of **`public`** methods. Also note that the module is not constrained by an invariant. This is handled entirely by the record invariant shown in Listing 2.27.

```java
final public class ATM {
 /* Fields omitted */

 /*@ spec_public @*/
 private static atm.ATMtypes.St St = new atm.ATMtypes.St(SetUtil.set(),
             null, false, MapUtil.map());
```

```
  /* Module methods omitted */
}
```

Listing 2.28: The state component in the `ATM` module.

10. Translating the state component
Annotate state components of module classes with the **`spec_public`** modifier to ensure that the state components can be referred to from the **`requires`** and **`ensures`** clauses of **`public`** methods.

2.6 Checking VDM types using JML

In this section we describe how the translator uses JML to check the consistency of VDM types when they are code-generated.

Throughout this section we construct a function called `Is(v,T)` that takes as input a Java value `v` and a VDM type `T` and produces a JML expression that can be used to check whether `v` represents a value of type `T`. We use `Is(v,T)` to check whether a Java value remains consistent with the VDM type that produces it. The check produced by `Is(v,T)` can be added to the generated Java code to ensure that no type violations occur.

This section covers some of the different classes of VDM types that the JML translator supports, and explains using our case study example, how JML is used to check a Java value against the VDM type that produces it. Finally, we summarise and provide the complete definition of `Is(v,T)` in Figure 2.1.

2.6.1 Where to generate dynamic type checks

Most of the types available in VDM are also present in Java in some form or other. The VDM and Java type systems do, however, have some differences that require us to generate extra checks to ensure that a Java value remains consistent with the VDM type that produces it.

In addition to producing the JML expression needed to check the consistency of a type, i.e. `Is(v,T)`, we also need to consider where to add the check to the generated code. The description below summarises the VDM-SL constructs that must be considered when adding these checks to the generated Java code. We use the term *parameter* to refer to an identifier whose value does not change. A parameter can be defined using a **`let`** construct, which is different from a state designator or variable that can be locally defined using a **`dcl`** statement or globally using a state definition (see section 2.2). The constructs to be considered are:

- **`return`** statement: If a functional description has a specified result type in its signature, then the returned value must be checked against the specified type.
- Parameters of functions and operations: The arguments passed to a functional description must be checked against the specified types of the corresponding formal parameters upon entry to the functional description.
- State designators: After updating a local or global state designator, the new value assigned must respect the type of the state designator.
- Variable or parameter declaration: After initialising a variable or parameter it must be checked against its declared type.

Is(v,T) =

Utils.is_bool(v)	**if** T = **bool**
Utils.is_nat(v)	**if** T = **nat**
Utils.is_nat1(v)	**if** T = **nat1**
Utils.is_int(v)	**if** T = **int**
Utils.is_rat(v)	**if** T = **rat**
Utils.is_real(v)	**if** T = **real**
Utils.is_char(v)	**if** T = **char**
Utils.is_token(v)	**if** T = **token**
Utils.is_(v,String.**class**)	**if** T = **seq of char**
Utils.is_(v,S_{CG}.**class**)	**if** T is a record or quote type S that generates to a Java class with the fully qualified name S_{CG}
(v == **null** || Is(v,S))	**if** T = [S]
V2J.isTup(v,n) && Is(v,T_1) &&...&& Is(v,T_n)	**if** T = T_1*...*T_n
Is(v,T_1) ||...|| Is(v,T_n)	**if** T = T_1|...|T_n
V2J.isSet(v) && (**\forall int** i; 0 <= i && i < V2J.size(v); Is(V2J.get(v,i),S))	**if** T = **set of** S
V2J.isSeq(v) && (**\forall int** i; 0 <= i && i < V2J.size(v); Is(V2J.get(v,i),S))	**if** T = **seq of** S
V2J.isSeq1(v) && (**\forall int** i; 0 <= i && i < V2J.size(v); Is(V2J.get(v,i),S))	**if** T = **seq1 of** S
V2J.isMap(v) && (**\forall int** i; 0 <= i && i < V2J.size(v); Is(V2J.getDom(v,i),D) && Is(V2J.getRng(v,i),R))	**if** T = **map** D **to** R
V2J.isInjMap(v) && (**\forall int** i; 0 <= i && i < V2J.size(v); Is(V2J.getDom(v,i),D) && Is(V2J.getRng(v,i),R))	**if** T = **inmap** D **to** R
Is(v,D) && inv_T(v)	**if** T is a named invariant type with domain type D and invariant method inv_T

Figure 2.1: Complete definition of `Is(v,T)`.

- Value definition: An explicitly typed value definition must specify a value consistent with its type.

All of the constructs in the list above – with the exception of the value definition – can be checked using a JML **`assert`** statement. The reason for this is that the code-generated versions of these constructs appear inside methods in the generated code. Since a VDM value definition code-generates to a **`public static final`** field (a constant) it is checked using a **`static`** invariant.

2.6.2 Translating basic types

In our example we may wish to check that the amount being withdrawn from an account is valid – for example by requiring that it is a natural number larger than zero, as shown in Listing 2.29.

```
let amount : nat1 = expense - profit
in
  Withdraw(accId, amount);
```

Listing 2.29: Use of explicit type annotation to ensure that a valid amount is being withdrawn.

In the generated Java code, shown in Listing 2.30, this is checked by analysing the value of the `amount` variable using the `Utils.is_nat1` method available from the Java code-generator's runtime library. This method is invoked from a JML annotation in order to check that `amount` is different from **`null`** and that it represents an integer larger than zero.

```
Number amount = expense.longValue() - profit.longValue();
//@ assert Utils.is_nat1(amount);
return Withdraw(accId, amount);
```

Listing 2.30: Use of JML to check that a valid amount is being withdrawn.

11. Checking of the `nat1` type

Let `v` be a value or object reference in the generated code that originates from a variable or pattern of type **`nat1`** and further define `Is(v,`**`nat1`**`) = Utils.is_nat1(v).`
To ensure that `v` represents a value of type **`nat1`**, generate a JML check to ensure that `Is(v,` **`nat1`**`)` holds.

The approach used to check other basic types follows the principles demonstrated using Listing 2.29 and Listing 2.30 – the main difference being that each basic type uses a dedicated method from the Java code-generator's runtime library. Therefore, we omit the details of how other basic types of VDM are checked using JML, and instead provide the complete set of rules in Figure 2.1.

We note that a record type or a quote type can be checked in a way similar to that of a basic type. The reason for this is that the Java code-generator produces a Java class for each of the record definitions and quote types in the VDM model. Therefore, all there is to checking whether an object reference represents a given record or quote class is to check whether the object reference is an instance of said class. The rules for checking record and quote types are included in Figure 2.1.

2.6.3 Translating optional types

To demonstrate how the JML translator handles optional types consider the `GetCurrentCardId` operation in Listing 2.31. This operation returns the identification of the debit card currently inserted into the machine, if any. Otherwise the operation returns **`nil`** to indicate the absence of a debit card. To allow **`null`** as a return value, the optional type operator is used to specify the return type of the operation as `[`**`nat`**`]`.

```
GetCurrentCardId : () ==> [nat]
GetCurrentCardId () ==
 if currentCard <> nil then
   return currentCard.id
 else
   return nil;
```

Listing 2.31: Operation for getting the id of the debit card currently inserted into the ATM.

Considering solely the signature of the code-generated version of this operation, shown in Listing 2.32, there is no way to tell that the return type represents a `[`**`nat`**`]`.

```
public static Number GetCurrentCardId(){...}
```

Listing 2.32: Signature of the code-generated version of the `GetCurrentCardId` operation.

The reason for this is that the Java code-generator uses the `Number` class (which is part of the Java standard library) to represent all numeric VDM types. That the return type of the operation is `[nat]` only becomes apparent when we start using the corresponding method.

To demonstrate this, Listing 2.33 uses the result of invoking the `GetCurrentCardId` method to initialise a variable named `id`. The initialisation of `id` is immediately followed by a check that ensures that it represents either **`null`** or a natural number. The approach of allowing **`null`** values like this is the same for all optional types.

```
Number id = GetCurrentCardId();
//@ assert id == null || Utils.is_nat(id);
```

Listing 2.33: Use of the `GetCurrentCardId` method in the generated code.

12. Checking of optional types

Let `v` be a value or object reference in the generated code that originates from a variable or pattern of the VDM type `[T]` and further define
`Is(v,[T]) = (v == null || Is(v,T))`
To ensure that `v` represents a value of type `[T]`, generate a JML check to ensure that `Is(v,[T])` holds.

2.6.4 Translating tuple types

In our case study example we use a tuple type to represent the status of the ATM: the first field is a **`boolean`** flag that indicates if the ATM is currently awaiting a debit card to be inserted, and the second field is a human-readable description of the current state of the ATM, e.g. “transaction in progress”. The signature of the operation that retrieves the status of the ATM is shown in Listing 2.34. Note in particular that the status returned is represented using the tuple type **`bool`** `*` **`seq of char`**.

```
GetStatus : () ==> bool * seq of char
GetStatus () == ...
```

Listing 2.34: The signature of the `GetStatus` operation.

In the generated Java code, every tuple value is represented as an instance of the `Tuple` class available from the Java code-generator runtime library. Since the `Tuple` class represents tuple values in general, each instance of this class must be checked against the specific tuple type that it originates from.

After the status of the ATM has been retrieved using the `GetStatus` method in the generated code, the status is checked as shown in Listing 2.35. First it is checked that `status` is a tuple of size two. Afterwards it is checked that the first field is a **`boolean`** and that the second field is a Java `String` (which represents the **`seq of char`** type).

```
Tuple status = GetStatus();
//@ assert (V2J.isTup(status,2) && Utils.is_bool(V2J.field(status,0)) &&
    Utils.is_(V2J.field(status,1),String.class));
```

Listing 2.35: Checking the ATM status in the generated code.

13. Checking of tuple types
Let `v` be a value or object reference in the generated code that originates from a variable or pattern of the VDM tuple type $T_1 * ... * T_n$ and further define $Is(v,T_1*...*T_n) = V2J.isTup(v,n)$ && $Is(v,T_1)$ &&...&& $Is(v,T_n)$ To ensure that `v` represents a value of type $T_1 * ... * T_n$, generate a JML check to ensure that $Is(v,T_1*...*T_n)$ holds.

2.6.5 Translating union types

Attempting to insert a debit card into the ATM results in the debit card being accepted, if no card is currently inserted and it is considered a valid card by the system. Otherwise the card is rejected. Based on the outcome of this the `NotifyUser` operation, shown in Listing 2.36, displays a message to inform the card holder about the current status of the session. This operation uses a union type, formed by the three quote types `<Accept>`, `<Busy>` and `<Reject>`, to represent one of three outcomes of the card holder attempting to insert a debit card into the ATM.

```
NotifyUser : <Accept>|<Busy>|<Reject> ==> ()
NotifyUser (outcome) ==
if outcome = <Accept> then
  Display("Card accepted")
elseif outcome = <Busy> then
  ...
```

Listing 2.36: Operation used to notify a ATM user.

The code-generated version of the `NotifyUser` operation is shown in Listing 2.37. Since the `outcome` parameter originates from the union type formed by the three quote types, it must be checked that `outcome` equals one of the three possible values. This check is performed immediately after entering the `NotifyUser` method, as shown in Listing 2.37.

```
public static void NotifyUser(final Object outcome) {
 //@ assert (Utils.is_(outcome,atm.quotes.AcceptQuote.class) || Utils.is_(
     outcome,atm.quotes.BusyQuote.class) || Utils.is_(outcome,atm.quotes.
     RejectQuote.class));
 if (Utils.equals(outcome, atm.quotes.AcceptQuote.getInstance())) {
   Display("Card␣accepted");
 } else if (Utils.equals(outcome, atm.quotes.BusyQuote.getInstance())){
   ...
}
```

Listing 2.37: Code-generated version of the `NotifyUser` operation.

14. Checking of union types
Let `v` be a value or object reference in the generated code that originates from a variable or pattern of the VDM union type $T_1|...|T_n$ and further define $Is(v,T_1|...|T_n) = Is(v,T_1) \|\|...\|\| Is(v,T_n)$ To ensure that `v` represents a value of type $T_1|...|T_n$, generate a JML check to ensure that $Is(v,T_1|...|T_n)$ holds.

2.6.6 Translating collections

In the generated code the `VDMSet`, `VDMSeq` and `VDMMap` collection classes are used as raw types. Therefore the code-generator does not take advantage of Java generics to make compile-time guarantees about the types of the objects a collection stores. This approach has the advantage of making it easier to store Java objects and values of different types in the same collection without having to introduce additional types. Although this allows the type system of VDM to be represented in Java it has the disadvantage that no compile-time guarantees can be made about the types of the objects that a collection stores.

In the ATM example we use the `TotalBalance` function, shown Listing 2.38, to calculate the total balance available from a set of accounts.

```
TotalBalance : set of Account -> real
TotalBalance (acs) ==
 if acs = {} then
   0
 else
  let a in set acs
  in
    a.balance + TotalBalance(acs \ {a});
```

Listing 2.38: Function that calculates the total balance available from a set of accounts.

When the `TotalBalance` function is code-generated to JML-annotated Java, the code-generator adds JML assertions to ensure that the set of accounts is consistent with the collection type used in VDM. Since an `Account` record is represented using a Java class with the same name, we have to check that every element in the set is an instance of said Java class. As shown in Listing 2.39, this is checked using a quantified expression. This expression uses a bound variable `i` to iterate over all the accounts and check that each element is an instance of the `Account` record class. Although sets are unordered collections, the quantified expression takes advantage of `VDMset` being implemented as an ordered collection. The formulation of the range expression in the quantified expression further ensures that the assertion can be checked using a tool such as the OpenJML runtime assertion checker, i.e. the assertion is executable.

```
/*@ pure @*/
public static Number TotalBalance(final VDMSet acs) {
 //@ assert (V2J.isSet(acs) && (\forall int i; 0 <= i && i < V2J.size(acs);
     Utils.is_(V2J.get(acs,i),atm.ATMtypes.Account.class)));
 if (Utils.empty(acs)) {
   Number ret_1 = 0L;
   //@ assert Utils.is_real(ret_1);
   return ret_1;
 } else { ... /*Compute sum recursively */}
}
```

Listing 2.39: Code-generated version of the `TotalBalance` operation.

The JML translator only uses Java 7 features since OpenJML did not support Java 8 at the time the JML translator was developed. Iterating over collections (as shown in Listing 2.39) may also be achieved using Java 8 features such as lambda expressions. For example, one could imagine a method used to check collection types that would take as input two arguments (1) the collection itself and (2) a predicate method (e.g. lambda expression) that would be evaluated for each of the elements in the collection. In that way the generated JML annotations would not have to rely on sets implemented as ordered collections. Since lambda expressions in Java are mostly syntactic sugar for anonymous inner classes, lambda expressions could in principle be represented solely using Java 7 features. However, using this approach, the generated JML annotations would not be concise, although this is only a concern if a human will read them.

15. Checking of sets

Let `v` be a value or object reference in the generated code that originates from a variable or pattern of the VDM set type **`set of`** `T` and further define

```
Is(v,set of T) = V2J.isSet(v) &&
(\forall int i; 0 <= i &&
i < V2J.size(v); Is(V2J.get(v,i),T))
```

To ensure that `v` represents a value of type **`set of`** `T`, generate a JML check to ensure that `Is(v,`**`set of`**` T)` holds.

The VDM sequence types **`seq`** and **`seq1`** are checked in a way similar to sets. The difference between checking the **`seq`** and **`seq1`** collection types is that the **`seq1`** type requires at least one element to be present in the sequence. Checking a map, which like a set is an unordered collection, takes advantage of `VDMMap` imposing an order on the domain and range values. The main difference between checking a map and a set is that both the domain and range values of a map have to be checked. Checking the injective map type **`inmap`** is similar to checking a standard map, except that the injectivity property must hold. We refrain from providing examples of how to check each of the collection types in VDM since they are similar to what has already been shown. Instead we summarise the rules for checking all of the collection types in Figure 2.1.

2.6.7 Translating named invariant types to JML

Since the Java code-generator does not generate additional class definitions for named invariant types, the invariant imposed on such a type cannot be expressed as a JML invariant. This is only possible for a record since it translates to a class definition.

Instead, we identify places in the generated code where a named invariant type may be violated, as described in subsection 2.6.1, and check that the invariant holds. Also, it is worth noting that a named invariant type, unlike a record type, does not have an explicit type constructor. Therefore, an expression can only violate a named invariant type if the expression is explicitly declared to be of that type.

The ATM in our example is not capable of dispensing cents and also imposes a limit on the amount of money that can be withdrawn. Therefore, the amount of money can be represented as a named invariant type. An attempt to withdraw an amount of money that exceeds 2000 will yield a runtime error. The named invariant type used to represent the amount withdrawn from an account is shown together with the `Withdraw` operation in Listing 2.40.

```
types
Amount = nat1
inv a == a < 2000;

operations
Withdraw : AccountId * Amount ==> real
Withdraw (id, amount) == ...
```

Listing 2.40: The amount to withdraw modelled using a named invariant type.

On entering the code-generated version of `Withdraw`, shown in Listing 2.41, we assert that `amount` meets the named invariant type `Amount`. The assertion does two things: First it performs a dynamic type check to ensure that `amount` is a valid domain type of `Amount` and secondly, it checks that the invariant predicate holds. For the example in Listing 2.41 this means checking that `amount` is of type **`nat1`** and smaller than 2000. Note that meeting the invariant condition does not imply compatibility with the domain type of the named invariant type and vice versa. For example, -1 is smaller than 2000 but it is not of type **`nat1`**. Likewise, 2001 is of type **`nat1`** but it exceeds 2000 so neither -1 nor 2001 are of type `Amount`.

```
public static Number Withdraw(final Number id, final Number amount){
...
//@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));
 ...
}
```

Listing 2.41: Checking a named invariant type of an operation parameter in JML.

The code-generated invariant method for type `Amount` is shown in Listing 2.42. Since the named invariant type check, shown in Listing 2.41, is evaluated from left to right using short-circuit evaluation semantics [26], the invariant method is only invoked if the value subject to checking is compatible with the domain type of the named invariant type. Therefore, it is safe to narrow (or cast) the type of the argument passed to the invariant method before performing the invariant check.

```
/*@ pure @*/
/*@ helper @*/
public static Boolean inv_ATM_Amount(final Object check_a) {
 Number a = ((Number) check_a);
 return a.longValue() < 2000L;
}
```

Listing 2.42: The named invariant type method for `Amount`.

16. Checking of named invariant types

Let `v` be a value or object reference in the generated code that originates from a variable or pattern of the VDM named invariant type `T` based on the domain type `D` and constrained by invariant predicate `e(p)`, i.e. `T` is defined as

```
types
T = D
inv p == e(p)
```

Then `T` has an invariant method, responsible for running the code-generated version of the `e(p)` check, with a signature defined as:

```
public static boolean inv_T(Object o)
```

Further define `Is(v,T) = Is(v,D) && inv_T(v)`

To ensure that `v` represents a value of type `T`, generate a JML check to ensure that `Is(v,T)` holds.

Note that the invariant method `inv_T` in rule 16 defines the input parameter `o` to be of type `Object`, thus allowing `inv_T` to accept inputs of any type. Therefore, `inv_T` must narrow the type of the input parameter `o` before performing the invariant check (see the example in Listing 2.42). This approach has the advantage that it allows simpler JML checks since the argument type does not need to be narrowed before the invariant method is invoked. Had the input parameter of the invariant method been defined using the smallest possible type, then the argument type would need to be narrowed for situations where the argument is masked as a union type. Although this would complicate the JML checks, it would have the advantage of allowing type narrowing to be removed from the invariant methods.

2.7 Other aspects of VDM-SL affecting the JML-generation

There are other aspects of VDM-SL that further complicate the generation of VDM-SL models to JML-annotated Java. In this section we use examples to demonstrate these issues and explain how they may be overcome.

2.7.1 Complex state designators

State designators may be composite data structures such as records with fields that themselves are records. Such a data type forms *complex state designators* that when modified require careful handling during the translation process. To demonstrate this, consider the three VDM-SL record definitions `R1`, `R2` and `R3` in Listing 2.43. Note in particular how the invariants of `R1` and `R2` depend on the field of `R3`. This transitive dependency complicates checking of invariants in the generated code. To demonstrate this, the operation in Listing 2.43 instantiates `R1` as `r1` and modifies it to violate the `R1` invariant, which causes a runtime-error to be reported.

```
types
R1 :: r2 : R2
inv r1 == r1.r2.r3.x <> -1;
R2 :: r3 : R3
inv r2 == r2.r3.x <> -2;
R3 :: x : int
inv r3 == r3.x <> -3;

operations
```

```
op: () ==> nat
op () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(5)));
 r1.r2.r3.x := -1;
 return 0;
)
```

Listing 2.43: Record nesting in VDM-SL.

The operation `op` in Listing 2.43 produces the method in Listing 2.44. For this example, `r1` is the same in both listings, `r2` is the same as `stateDes_1` in Listing 2.44, and `r3` is the same as `stateDes_2`. Note that in Listing 2.44 we have removed fully qualified names of record classes and other JML checks that are not relevant.

```
public static Number op() {
 R1 r1 = new R1(new R2(new R3(5L)));
 R2 stateDes_1 = r1.get_r2();
 R3 stateDes_2 = stateDes_1.get_r3();
 stateDes_2.set_x(-1L);
 //@ assert \invariant_for(stateDes_1);
 //@ assert \invariant_for(r1);
 Number ret_1 = 0L;
 return ret_1;
}
```

Listing 2.44: Code-generated version of the operation from Listing 2.43.

Immediately after completing the state update, i.e. invoking `stateDes_2.set_x(-1L)`, the following things happen:

1. The state of `stateDes_2` becomes *visible* thus triggering the `stateDes_2` invariant check.
2. The invariant check of `stateDes_1` is run as `\invariant_for(stateDes_1)` and finally,
3. the invariant check of `r1` is run by asserting `\invariant_for(r1)`, which causes a runtime-error to be reported.

Strictly speaking the objects pointed to by `stateDes_1` and `r` are also in visible states after executing the update to `stateDes_2` and therefore the invariants of those objects should also hold. In particular a state is *visible* for an object `o` *"when no constructor, destructor, non-static method invocation with* `o` *as receiver, or static method invocation for a method in* `o`*'s class or some superclass of* `o`*'s class is in progress [24]"*. So in theory the invariant checks should not have to be run explicitly (step 2 and step 3). The reason that the JML translator generates these checks anyway has to do with the strategies JML tools use to check invariants.

Tools such as JML runtime checkers may assume no problems with ownership aliasing to avoid having to keep track of what objects and types are in visible states. Although this reduces the overhead of checking invariants, it also means that some invariant violations might go unnoticed. Alternatively, tools can check every applicable invariant for classes and objects in visible states but this adds a significant overhead to the program execution.

Since aliasing can never occur in VDM-SL, it becomes simpler to keep track of what objects are in a visible state in the generated code and thus generate JML checks that explicitly trigger the invariants checks. This has the advantage that invariant violations do not go unnoticed even though a JML tool adopts a more practical approach to checking invariants.

For the example in Listing 2.44, the important thing is to ensure that the violation of the invariant of `R1` is reported after executing the state update. This is done by asserting the entire chain of state designators. The JML translator is able to generate these checks since it keeps track of state designators of records that may have been affected by updates to other state designators.

17. Checking transitive dependencies

Let d_n be a state designator of a record in the generated code that has been updated non-atomically, and let $d_k, ..., d_1$, for $k = n-1$, be state designators that were affected by the update to d_n. Further assume that d_i may be of one of m_i record types $D_{i1}, ..., D_{im_i}$. Immediately after executing the update to d_n the state of d_n becomes visible. To ensure that the invariant is evaluated for all affected state designators, execute the following sequence of assertions:

```
//@ assert dk instance of Dk1 ==>
   \invariant_for((Dk1) dk);
...
//@ assert dk instance of Dkmk ==>
   \invariant_for((Dkmk) dk);
....
//@ assert d1 instance of D11 ==>
   \invariant_for((D11) d1);
...
//@ assert d1 instance of D1m1 ==>
   \invariant_for((D1m1) d1);
```

Note that the code in Listing 2.44 omits the **`instance of`** checks, proposed by rule 17, since the types of the affected state designators can be determined statically.

Regarding rule 9, similar issues with transitive dependencies may occur in the generated code when dealing with atomic execution. Recall that invariant checking is disabled before a code-generated atomic statement block is executed. Once the atomic execution has completed, invariant checking is re-enabled, and therefore rule 9 must also take into account all the state designators that were affected by the atomic execution.

2.7.2 Recursive types

It is possible to formulate recursive types for which the generated JML checks can only perform limited type checking. To demonstrate this, consider the recursive VDM type definition in Listing 2.45. For this example, `S` represents an infinite number of types including **`nat1`** as well as all possible dimensions of sequences that store elements of type **`nat1`**, i.e. **`seq of nat1`**, **`seq of seq of nat1`** and so on.

```
types
S = nat1 | seq of S;
```

Listing 2.45: Example of recursive type definition in VDM.

The issue with this kind of type definition is that `Is(v,S)` in theory becomes an expression of infinite length. The JML translator stops generating type checks whenever it encounters type cycles. For the particular example in Listing 2.45 this means that a Java value or object reference `v` is only considered to respect `S` if `Utils.is_nat1(v)` holds. For the rest of this section, we discuss the current limitations of type checking recursive types, and describe how these limitations may be addressed.

The approach used to check types could be changed to also take the depth of the recursion `n` into account, i.e. use `Is(v,T,n)` to generate the type checks. The current approach used by the JML translator thus corresponds to generating checks using `Is(v,T,1)`. `Is(v,S,2)` then generates checks for types **`nat1`** and **`seq of nat`**, whereas `Is(v,S,3)` additionally generates a check for the type **`seq of seq of nat1`**.

Alternatively, checking a recursive type `T` (such as `S` shown in Listing 2.45) can be done using a code-generated recursive method that is constructed in a way that allows a value `v` to be validated against `T`. Although static provers may not be able to perform checking of such types it should be possible using runtime assertion checking. However, in order to enable this style of type checking, the JML translator would have to be extended with functionality that enables these methods to be generated such that they can be invoked from the generated JML assertions.

The limitation of the JML translator for the example shown in Listing 2.45 is a consequence of `S` being defined using the union type constructor "`|`". However, it is possible to check more practical examples of recursively defined types such as the linked list `LL` shown in Listing 2.46.

To demonstrate this, consider the construction of a linked list value in VDM that contains the numbers 1, 2 and 3 as shown in Listing 2.47. In the generated code this value is represented using the code shown in Listing 2.48.

```
types
LL ::
  element : nat
  tail : [LL]
```

Listing 2.46: Example of a linked list defined using a record type.

```
mk_LL(1, mk_LL(2, mk_LL(3, nil)))
```

Listing 2.47: Example of a linked list value in VDM.

```
new LL(1L,new LL(2L,new LL(3L, null)));
```

Listing 2.48: Example of a linked list value in Java.

Each time an object of type `LL` is instantiated in Java the constructor checks the types of the current `element` and the `tail` – see Listing 2.49. For this linked list example, it is therefore possible to type check `LL` since the VDM type is represented using a recursively defined class in the generated code.

```
public LL(final Number _element, final LL _tail) {
 //@ assert Utils.is_nat(_element);
 //@ assert (_tail == null || Utils.is_(_tail,LL.class));
 ...
}
```

Listing 2.49: Type checking a linked list using JML.

2.7.3 Detecting problems with the generated code

As explained in subsection 2.5.4 deep copying objects may significantly affect the performance of the generated code. Therefore, the user may not always want to have these copy calls generated. However, from a general perspective this may result in code that does not preserve the semantics across the translation. JML specifications can help detect such problems. To demonstrate this consider the VDM-SL operation in Listing 2.50. This operation assumes the existence of a two-dimensional vector `Vector2D`, defined as a record (a value type). In Listing 2.50 `v2` is created as a deep copy of `v1`, and therefore the assignment to `v1` has no affect on `v2`, and `op` therefore returns 1 (see the postcondition).

If this example is translated to Java with deep copying *disabled* the code shown in Listing 2.51 is produced. Note that this listing omits the generated JML assertions to focus on the postcondition.

```
op : () ==> nat
op () == (
dcl v1 : Vector2D := mk_Vector2D(1,2);
dcl v2 : Vector2D := v1; -- Copy value
v1.x := 2;
return v2.x;)
post RESULT = 1
```

Listing 2.50: Use of value types in VDM.

```
//@ ensures post_op(\result);
public static Number op() {
  Vector2D v1 = new Vector2D(1L,2L);
  Vector2D v2 = v1;
  v1.set_x(2L);
  Number ret_1 = v2.get_x();
  return ret_1;
}
```

Listing 2.51: Generated Java code without copy calls.

If this code is executed using the OpenJML runtime assertion checker an error is reported because the method returns 2, which is different from the result obtained by executing the corresponding VDM-SL operation. Since deep copying is disabled only the `v1` reference is copied, and therefore the update to `v1`, i.e. `v1.set_x(2L)`, also affects `v2`.

The detection of the postcondition violation as reported by the OpenJML runtime assertion checker is shown in Listing 2.52. However, if the code is generated with deep copying enabled (at the cost of performance) then `v2` will be constructed as `Utils.copy(v1)` and the method will change to return 1, as expected.

```
Ex/DEFAULT.java:17: JML postcondition is false
    public static Number op() {
Ex/DEFAULT.java:16: Associated declaration: Ex/DEFAULT.java:17:
    //@ ensures post_op(\result);
```

Listing 2.52: Detection of a postcondition violation.

2.8 Translation assessment

In this section we provide an assessment of the translation. We first describe how the correctness of the translation was assessed, and afterwards we discuss the scope and treated feature set in relation to existing JML tools.

2.8.1 Translation correctness

The translation rules have been validated by running examples through the JML translator and analysing the generated Java/JML using the OpenJML runtime assertion checker. Some of the examples used to test the tool constitute *integration tests* that have been developed by the authors. In addition, we have used the tool to analyse an *external specification* (originally used as part of an industrial case study) that the authors have not been involved in the development of. A summary of the different examples used to test the translation is given below. Additional details about the examples can be found via the references provided.

The *integration tests* currently consist of 85 examples that cover testing of all the translation rules. Each test (typically) forms a minimal example that exercises a small part of the entire translation (such as a single rule). The workflow for running these tests is as follows: First, the test model is translated to JML-annotated Java using the JML translator. Next, the generated Java/JML is compiled and executed using the OpenJML runtime assertion checker. Finally, the (actual) output reported by the OpenJML runtime assertion checker is compared to the expected output in order to confirm that the behaviour of the test model is preserved across the translation. For example, if the execution of a test model produces a precondition violation then the equivalent error is expected to be produced when the generated Java/JML is executed using the OpenJML runtime assertion checker. All the examples used to test the JML translator are available via Overture's GitHub page [31] or can be found in appendix C.

Compared to the *integration tests*, the *external specification* is a large example that is rich in terms of DbC elements. The model was originally developed to study the properties of an algorithm used to obfuscate Financial Accounting District (FAD) codes, which are six digit numbers used to identify branches of a retailer. The customer required that obfuscated FAD codes were still six digit numbers, remained unique (per branch), and that the entire range of FAD codes (0-999999) was still available. In addition, the obfuscation had to be a light-weight calculation (rather than a look-up in a table). The properties of the algorithm were described using VDM contracts to allow the algorithm to be validated using VDM's test automation features [20].

Investigating whether the algorithm met the requirements necessitated the generation and execution of one million tests that initially could not be handled by any of the VDM tools (either due to intractable execution times, or because the VDM interpreter ran out of memory). Motivated by this, the specification was translated into a JML-annotated Java program [35], and all one million tests were executed using a code-generated version of the VDM specification. In that way, the properties of the obfuscation algorithm could be validated by executing a code-generated version of the VDM specification using the OpenJML runtime assertion checker.

2.8.2 Translation scope and treated feature set

As explained in subsection 2.7.2 it is possible to formulate recursive types that currently are not supported by the JML translator. Aside from that, all VDM-SL's types and contract-based elements are supported. However, the JML translator does not currently support the object-oriented and real-time dialects of VDM, called VDM++ [12] and VDM-RT [23].

The Java code-generator that we extend currently only uses Java 7 features in the generated code. OpenJML is the only JML tool that we are aware of that supports this version of Java.

Specifically, as of December 2016, OpenJML version 0.8.5 was released with support for Java 8, i.e. the latest official Java version (at the current time of writing). Other JML tools, on the other hand, lack support for recent Java versions (in particular Java 7 and 8). Therefore, these tools cannot currently be used to analyse the generated Java/JML.

The JML translation is only valuable if the JML features that it relies on are supported by JML tools. Specifically, we have aimed to develop a translation that generates Java/JML that can be analysed using OpenJML. However, the translation would benefit from the `\invariant_for` construct, which OpenJML does not currently support. Instead we offer an alternative way to represent this construct in order to achieve compatibility with OpenJML (see subsection 2.5.6 for details).

2.9 Related work

In [37] Vilhena considers the possibilities for automatically converting between VDM++ and JML and the approach is demonstrated using a proof-of-concept implementation. That work considers a bi-directional mapping, whereas we only consider a one-way translation from VDM-SL to Java/JML. The bi-directional mapping proposed by Vilhena only produces the JML specification files (where non-model methods do not define bodies). Therefore, Vilhena's mapping does not generate annotations at the statement level, which is an essential part of our work. The implementation of the bi-directional mapping was originally targeting the Overture tool, but it never reached maturity to be included in the release of the tool.

Rules for translating from a subset of VDM-SL to JML are proposed by Jin et al. in [14]. Their approach also considers implicit functional descriptions but it provides limited support for translation of record definitions and named invariant types. In the early phases of the software development process the authors propose to formulate requirements in natural language or using the Unified Modeling Language (UML) [34] and then formalise them in VDM-SL to eliminate ambiguity. Subsequently the authors manually apply their rules to the VDM-SL specification to produce an initial version of the software implementation. Their work does, however, not take generation of the bodies of functions and operations into account. Therefore, the authors only produce the method signatures for the Java methods when translating the functional descriptions of the VDM-SL model.

The translation rules proposed by Jin et al. have been implemented as an Eclipse plugin by Zhou et al. in [42]. The plugin takes a VDM-SL specification as input, which is type-checked using VDMTools [18], and outputs JML-annotated Java classes that must be completed manually by the developer.

Translations from other formal notations or modelling languages to JML-annotated Java have also been developed. As an example, Rivera et al. present the EventB2Java tool [33] – a code-generator, which is capable of translating both abstract and refinement Event-B [1] models into JML-annotated Java. EventB2Java has the advantage over other Event-B code-generators that it does not require user intervention as part of the code-generation process, which is similar to our approach.

In [25] Lensink et al. present a prototype code-generator that translates a subset of the Prototype Verification System (PVS) [32] to an intermediate representation in Why [9] suitable for program verification. Subsequently the Why representation is translated to JML-annotated Java. In their work the authors focus on translating executable PVS constructs, which is similar to what we do for VDM-SL. A key feature of their code-generator is that it, in addition to specification code, also translates proven properties, which is outside the scope of our work.

Hubbers et. al propose AutoJML [13] – a tool for translating UML state diagrams into JML-annotated Java Card code [41]. A state diagram describes a Java Card applet from which AutoJML produces Java skeleton code annotated with JML. In the generated code the different states are represented as constant values, and an additional Java field is used to represent the current state of the applet. A JML **`invariant`** is used to specify the valid state values for this field, and a JML **`constraint`** is used to describe the valid state transitions. This is comparable to the way we enable and disable invariant checking, which we do by toggling the `invChecksOn` **`ghost`** field using **`set`** statements.

In [17] Klebanov proposes an approach similar to that of Hubbers et al. Instead of using UML state diagrams, Klebanov uses automata-based programming to describe the behaviour of a smart card application, which is generated to JML-annotated Java Card code. Klebanov argues that use of automata-based programming over UML state diagrams is a better way to describe application-specific behaviour. A similar argument can be made for VDM-SL, which is suitable for capturing the dynamic aspects of a system.

2.10 Conclusion and future plans

In this report we have demonstrated how VDM-SL models can be translated to JML-annotated Java programs that can be checked for correctness using JML tools. The JML translator uses JML to represent the DbC elements of VDM-SL, and generates checks that help ensure the consistency of VDM-SL types across the translation.

The principles for pre- and postconditions in VDM-SL and JML are similar although there are subtle semantic differences between the two notations. These differences are mostly caused by the fact that JML is built on top of Java, where object types use reference semantics. VDM-SL, on the other hand, solely uses value types. Therefore, it is necessary to employ deep cloning principles when representing value types in JML-annotated Java code.

Checking state and record invariants in the generated code is complicated due to two reasons: First, atomic execution in VDM requires a way to control when invariant checking must be done. We achieve this by using a **`ghost`** field to indicate when invariant checking is enabled, and update it before entering and leaving the **`atomic`** statement. Secondly, we have demonstrated that transitive dependencies between records sometimes require extra JML checks to be generated to ensure that the invariant checks are evaluated when they should.

The differences between the type systems of VDM-SL and Java further necessitate extra checks to be produced. These checks are needed to ensure that the generated code does not violate any of the constraints imposed by the types in the VDM-SL model. Overture performs these dynamic type checks internally, whereas they must be made explicit in Java.

Although DbC languages often support many of the same DbC concepts, it is the semantic differences between the languages that make developing a translation challenging. In this report we have shown several examples of such differences and how they can be addressed. Naturally, translating between other specification language pairs may reveal other differences and design details that are of interest to researchers and practitioners working on comparable tasks. However, based on the experiences gained by developing the VDM-SL-to-JML translation, we list some of the design details that we believe are likely to challenge the development of translations between other specification language pairs:

Invariants: The times when invariants are evaluated varies across specification languages. For example, in VDM they have to hold at all times (except inside **`atomic`** statements), whereas in JML they must hold in visible states. When invariants have different semantics the translation must find a way to either produce or reduce the number of invariant checks at the appropriate places in the code.

Type systems: The differences between type systems require careful attention when developing a translation. Especially, when the destination language (e.g. JML) uses a more "coarse-grained" type system than the source language (e.g. VDM-SL). For such situations extra checks must be produced to ensure that types are used consistently across the translation. In our work we use the function `Is(v,T)` to produce these extra checks.

Atomic execution: Languages may use dedicated constructs to represent atomic execution (e.g. VDM) or by allowing invariants not to hold at certain times (e.g. JML). In this report an example was given of how a dedicated construct can be emulated in a language that does not support one natively.

Old state: Despite pre- and postconditions being similar concepts in different specification languages it is likely that the notion of old state may require careful handling when developing a translation between two specification languages. In our work, a deep cloning principle was employed to ensure the correct construction of the old state.

In the future we plan to use this work in the context of test automation. In VDM it is possible to specify a trace definition in a way similar to that of a regular expression. This trace can then be expanded into a large collection of tests that can be executed against the model. This is a useful way to detect deficiencies in the model, such as missing preconditions, postconditions and invariants [20].

We plan to code-generate the trace expansion such that the tests can be executed against the code-generated version of the model. The work presented in this report can then be used to detect contract or type violations and give verdicts to the code-generated trace tests. We believe that this will be particularly advantageous for execution of large collections of tests. We expect this approach to significantly increase execution speed for test cases and also allow more tests to be executed. In addition, we plan to look into JML-generation for other VDM dialects such as VDM++. However, since VDM++ is object-oriented and supports concurrency, we envisage that this will give rise to a completely new set of challenges not addressed by the work in this report.

So far the analysis of the generated Java/JML has primarily been limited to runtime assertion checking. Another item of future work is to formally verify the generated code against the JML specification. In particular, by investigating to what extent this is possible, and whether the JML translation can be optimised in a way that better supports formal verification through static analysis. For example, currently the translation produces auxiliary methods for invariants and pre- and postconditions that are used as part of the JML specification. However, use of method calls in specifications complicates static analysis due to, for example, the possibility of exceptions or non-terminating behaviour [7].

We hope that our work will serve as inspiration for other researchers who seek to bridge the gap between other specification notations and implementation technologies that support the DbC approach. We believe that the rules proposed in this report can be useful for others who want to translate between specification languages such as ASM, B and Z and implementation technologies such as Spec#, Sparc-Ada and Eiffel.

A

The ATM model

This appendix contains the complete version of the ATM model that is used in this report to demonstrate the JML translation.

```
module ATM

imports from IO all
exports all

definitions

state St of
 validCards : set of Card
 currentCard : [Card]
 pinOk : bool
 accounts : map AccountId to Account
 init St == St = mk_St({},nil,false,{|->})
 inv mk_St(v,c,p,a) ==
  (p or c <> nil => c in set v)
  and
  forall id1, id2 in set dom a &
   id1 <> id2 =>
   a(id1).cards inter a(id2).cards = {}
end

types

Card ::
 id : nat
 pin : Pin;

Pin = nat
inv p == 0 <= p and p <= 9999;

AccountId = nat
inv id == id > 0;

Account ::
 cards : set of Card
 balance : real
 inv a == a.balance >= -1000;

Amount = nat1
```

```
inv a == a < 2000;

functions

TotalBalance : set of Account -> real
TotalBalance (acs) ==
 if acs = {} then
   0
 else
  let a in set acs
  in
    a.balance + TotalBalance(acs \ {a})
measure TotalBalanceMes;

TotalBalanceMes: set of Account +> nat
TotalBalanceMes(acs) == card acs;

operations

GetStatus : () ==> bool * seq of char
GetStatus () ==
if currentCard <> nil then
  if pinOk then
    return mk_(false, "transaction␣in␣progress.")
  else
    return mk_(false, "debit␣card␣inserted.␣Awaiting␣pin␣code.")
else
  return mk_(true,"no␣debit␣card␣is␣currently␣inserted␣into␣the␣machine.");

OpenAccount : set of Card * AccountId ==> ()
OpenAccount (cards,id) ==
 accounts := accounts munion {id |-> mk_Account(cards,0.0)}
pre id not in set dom accounts
post id in set dom accounts and
     accounts(id).balance = 0;

AddCard : Card ==> ()
AddCard (c) ==
 validCards := validCards union {c}
pre c not in set validCards
post c in set validCards;

RemoveCard : Card ==> ()
RemoveCard (c) ==
 validCards := validCards \ {c}
pre c in set validCards
post c not in set validCards;

InsertCard : Card ==> <Accept>|<Busy>|<Reject>
InsertCard (c) ==
if c in set validCards then
(
 currentCard := c;
 return <Accept>;
)
elseif currentCard <> nil then
  return <Busy>
else
  return <Reject>
pre currentCard = nil
```

```
post
 if RESULT = <Accept> then
  currentCard = c
 else if RESULT = <Busy> then
   currentCard = currentCard~
 else currentCard = nil;

Display : seq of char ==> ()
Display (msg) ==
  IO`println(msg);

NotifyUser : <Accept>|<Busy>|<Reject> ==> ()
NotifyUser (outcome) ==
if outcome = <Accept> then
  Display("Card␣accepted")
elseif outcome = <Busy> then
  Display("Another␣card␣has␣already␣been␣inserted")
else if outcome = <Reject> then
  Display("Unknown␣card")
else
  error;

EnterPin : Pin ==> ()
EnterPin (pin) ==
 pinOk := (currentCard.pin = pin)
pre currentCard <> nil;

ReturnCard : () ==> ()
ReturnCard () ==
atomic
(
 currentCard := nil;
 pinOk := false;
)
pre currentCard <> nil
post currentCard = nil and not pinOk;

Withdraw : AccountId * Amount ==> real
Withdraw (id, amount) ==
let newBalance = accounts(id).balance - amount
in
(
 accounts(id).balance := newBalance;
 return newBalance;
)
pre currentCard in set validCards and pinOk and
    currentCard in set accounts(id).cards and
    id in set dom accounts
post
let accountPre = accounts~(id),
    accountPost = accounts(id)
in
 accountPre.balance = accountPost.balance + amount and
 accountPost.balance = RESULT;

Deposit : AccountId * Amount ==> real
Deposit (id, amount) ==
let newBalance = accounts(id).balance + amount
in
(
 accounts(id).balance := newBalance;
 return newBalance;
)
pre pre_Withdraw(id,amount,St)
post
let accountPre = accounts~(id),
    accountPost = accounts(id)
in
 accountPre.balance + amount = accountPost.balance and
 accountPost.balance = RESULT;

PrintAccount: AccountId ==> ()
PrintAccount(id) ==
let balance = accounts(id).balance
in
 IO`printf("Balance␣is␣for␣account␣%s␣is␣%s\n", [id,balance]);

GetCurrentCardId : () ==> [nat]
GetCurrentCardId () ==
 if currentCard <> nil then
   return currentCard.id
 else
   return nil;

--
-- Test operations
--

TestCurrentCardId : () ==> [nat]
TestCurrentCardId () ==
let id = GetCurrentCardId()
in
  return id;

TestStatus : () ==> real
TestStatus () ==
let accId = 1,
   c1 = mk_Card(1,1234)
in
(

 AddCard(c1);
 OpenAccount({mk_Card(1,1234)},accId);

 let status = GetStatus(),
     awaitingCard = status.#1,
     msg = status.#2
  in
  (
    IO`println("Message:␣" ^ msg);
    if awaitingCard and <Accept> = InsertCard(c1) then
    (
      NotifyUser(<Accept>);
      EnterPin(1234);
      Deposit(accId,100);
    );
  );

  return 0;
);
```

```

TestWithdraw : () ==> real
TestWithdraw () ==
let accId = 1,
    cardId = 1,
    pin = 1234,
   c1 = mk_Card(cardId,pin)
in
(

 AddCard(c1);
 OpenAccount({mk_Card(1,1234)},accId);

 if InsertCard(c1) = <Accept> then
 (
   EnterPin(pin);
   let expense = 600,
     profit = 100
   in
     let amount : nat1 = expense - profit
     in
       Withdraw(accId, amount);
 );

 error;
);

TestTotalBalance : () ==> real
TestTotalBalance () ==
let card1 = mk_Card(1,1234),
    card2 = mk_Card(2,5678),
    ac1 = mk_Account({card1}, 1000),
    ac2 = mk_Account({card2}, 500)
in
  TotalBalance({ac1,ac2});

TestScenario : () ==> ()
TestScenario() ==
let accId1 : AccountId = 1,
    pin1 = 1234,
    card1 = mk_Card(1, pin1),
    pin2 = 2345,
    card2 = mk_Card(2, pin2)
in
(
 AddCard(card1);
 AddCard(card2);
 OpenAccount({card1, card2},accId1);
 let - = InsertCard(card2) in skip;
 PrintAccount(accId1);
 EnterPin(2345);
 let - = Deposit(accId1, 200) in skip;

 PrintAccount(accId1);

 ReturnCard();
 RemoveCard(card1);
 RemoveCard(card2);
);

```

```
end ATM
```

B

The code-generated ATM model

This appendix contains the code-generated version of the ATM model in appendix A.

```
package atm;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class ATM implements java.io.Serializable {
  /*@ spec_public @*/

  private static atm.ATMtypes.St St =
      new atm.ATMtypes.St(SetUtil.set(), null, false, MapUtil.map());
  /*@ public ghost static boolean invChecksOn = true; @*/

  private ATM() {}

  public static Tuple GetStatus() {

    if (!(Utils.equals(Utils.copy(St.get_currentCard()), null))) {
      if (St.get_pinOk()) {
        Tuple ret_1 = Tuple.mk_(false, "transaction␣in␣progress.");
        //@ assert (V2J.isTup(ret_1,2) && Utils.is_bool(V2J.field(ret_1,0))
            && Utils.is_(V2J.field(ret_1,1),String.class));

        return Utils.copy(ret_1);

      } else {
        Tuple ret_2 = Tuple.mk_(false, "debit␣card␣inserted.␣Awaiting␣pin␣
            code.");
        //@ assert (V2J.isTup(ret_2,2) && Utils.is_bool(V2J.field(ret_2,0))
            && Utils.is_(V2J.field(ret_2,1),String.class));

        return Utils.copy(ret_2);
      }

    } else {
      Tuple ret_3 = Tuple.mk_(true, "no␣debit␣card␣is␣currently␣inserted␣
          into␣the␣machine.");
      //@ assert (V2J.isTup(ret_3,2) && Utils.is_bool(V2J.field(ret_3,0)) &&
          Utils.is_(V2J.field(ret_3,1),String.class));

      return Utils.copy(ret_3);
    }
  }
  //@ requires pre_OpenAccount(cards,id,St);
  //@ ensures post_OpenAccount(cards,id,\old(St.copy()),St);

  public static void OpenAccount(final VDMSet cards, final Number id) {

    //@ assert (V2J.isSet(cards) && (\forall int i; 0 <= i && i < V2J.size(
        cards); Utils.is_(V2J.get(cards,i),atm.ATMtypes.Card.class)));

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    //@ assert St != null;

    St.set_accounts(
        MapUtil.munion(
            Utils.copy(St.get_accounts()),
            MapUtil.map(new Maplet(id, new atm.ATMtypes.Account(cards, 0.0))
                )));
  }
  //@ requires pre_AddCard(c,St);
  //@ ensures post_AddCard(c,\old(St.copy()),St);

  public static void AddCard(final atm.ATMtypes.Card c) {

    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    //@ assert St != null;

    St.set_validCards(SetUtil.union(Utils.copy(St.get_validCards()), SetUtil
        .set(Utils.copy(c))));
  }
  //@ requires pre_RemoveCard(c,St);
  //@ ensures post_RemoveCard(c,\old(St.copy()),St);

  public static void RemoveCard(final atm.ATMtypes.Card c) {

    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    //@ assert St != null;

    St.set_validCards(SetUtil.diff(Utils.copy(St.get_validCards()), SetUtil.
        set(Utils.copy(c))));
  }
  //@ requires pre_InsertCard(c,St);
  //@ ensures post_InsertCard(c,\result,\old(St.copy()),St);

  public static Object InsertCard(final atm.ATMtypes.Card c) {

    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    if (SetUtil.inSet(c, Utils.copy(St.get_validCards()))) {
      //@ assert St != null;

      St.set_currentCard(Utils.copy(c));

      Object ret_4 = atm.quotes.AcceptQuote.getInstance();
      //@ assert (Utils.is_(ret_4,atm.quotes.AcceptQuote.class) || Utils.is_(ret_4,atm.quotes.BusyQuote.class) || Utils.is_(ret_4,atm.quotes.RejectQuote.class));

      return ret_4;

    } else if (!(Utils.equals(Utils.copy(St.get_currentCard()), null))) {
      Object ret_5 = atm.quotes.BusyQuote.getInstance();
      //@ assert (Utils.is_(ret_5,atm.quotes.AcceptQuote.class) || Utils.is_(ret_5,atm.quotes.BusyQuote.class) || Utils.is_(ret_5,atm.quotes.RejectQuote.class));

      return ret_5;

    } else {
      Object ret_6 = atm.quotes.RejectQuote.getInstance();
      //@ assert (Utils.is_(ret_6,atm.quotes.AcceptQuote.class) || Utils.is_(ret_6,atm.quotes.BusyQuote.class) || Utils.is_(ret_6,atm.quotes.RejectQuote.class));

      return ret_6;
    }
  }

  public static void Display(final String msg) {

    //@ assert Utils.is_(msg,String.class);

    IO.println(msg);
  }

  public static void NotifyUser(final Object outcome) {

    //@ assert (Utils.is_(outcome,atm.quotes.AcceptQuote.class) || Utils.is_(outcome,atm.quotes.BusyQuote.class) || Utils.is_(outcome,atm.quotes.RejectQuote.class));

    if (Utils.equals(outcome, atm.quotes.AcceptQuote.getInstance())) {
      Display("Card␣accepted");
    } else if (Utils.equals(outcome, atm.quotes.BusyQuote.getInstance())) {
      Display("Another␣card␣has␣already␣been␣inserted");
    } else {
      if (Utils.equals(outcome, atm.quotes.RejectQuote.getInstance())) {
        Display("Unknown␣card");
      } else {
        throw new RuntimeException("ERROR␣statement␣reached");
      }
    }
  }
  //@ requires pre_EnterPin(pin,St);

  public static void EnterPin(final Number pin) {

    //@ assert (Utils.is_nat(pin) && inv_ATM_Pin(pin));

    //@ assert St != null;

    St.set_pinOk(Utils.equals(St.get_currentCard().get_pin(), pin));
  }
```

```
//@ requires pre_ReturnCard(St);
//@ ensures post_ReturnCard(\old(St.copy()),St);

public static void ReturnCard() {

  atm.ATMtypes.Card atomicTmp_1 = null;
  //@ assert ((atomicTmp_1 == null) || Utils.is_(atomicTmp_1,atm.ATMtypes.
      Card.class));

  Boolean atomicTmp_2 = false;
  //@ assert Utils.is_bool(atomicTmp_2);

  {
      /* Start of atomic statement */
    //@ set invChecksOn = false;

    //@ assert St != null;

    St.set_currentCard(Utils.copy(atomicTmp_1));

    //@ assert St != null;

    St.set_pinOk(atomicTmp_2);

    //@ set invChecksOn = true;

    //@ assert St.valid();

  } /* End of atomic statement */
}
//@ requires pre_Withdraw(id,amount,St);
//@ ensures post_Withdraw(id,amount,\result,\old(St.copy()),St);

public static Number Withdraw(final Number id, final Number amount) {

  //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

  //@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));

  final Number newBalance =
      ((atm.ATMtypes.Account) Utils.get(St.accounts, id)).get_balance().
          doubleValue()
          - amount.longValue();
  //@ assert Utils.is_real(newBalance);

  {
    VDMMap stateDes_1 = St.get_accounts();

    atm.ATMtypes.Account stateDes_2 = ((atm.ATMtypes.Account) Utils.get(
        stateDes_1, id));

    //@ assert stateDes_2 != null;

    stateDes_2.set_balance(newBalance);
    //@ assert (V2J.isMap(stateDes_1) && (\forall int i; 0 <= i && i < V2J
        .size(stateDes_1); (Utils.is_nat(V2J.getDom(stateDes_1,i)) &&
        inv_ATM_AccountId(V2J.getDom(stateDes_1,i))) && Utils.is_(V2J.
        getRng(stateDes_1,i),atm.ATMtypes.Account.class)));

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);
```

```

      //@ assert St.valid();

      Number ret_7 = newBalance;
      //@ assert Utils.is_real(ret_7);

      return ret_7;
    }
  }
  //@ requires pre_Deposit(id,amount,St);
  //@ ensures post_Deposit(id,amount,\result,\old(St.copy()),St);

  public static Number Deposit(final Number id, final Number amount) {

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    //@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));

    final Number newBalance =
        ((atm.ATMtypes.Account) Utils.get(St.accounts, id)).get_balance().
            doubleValue()
            + amount.longValue();
    //@ assert Utils.is_real(newBalance);

    {
      VDMMap stateDes_3 = St.get_accounts();

      atm.ATMtypes.Account stateDes_4 = ((atm.ATMtypes.Account) Utils.get(
          stateDes_3, id));

      //@ assert stateDes_4 != null;

      stateDes_4.set_balance(newBalance);
      //@ assert (V2J.isMap(stateDes_3) && (\forall int i; 0 <= i && i < V2J
          .size(stateDes_3); (Utils.is_nat(V2J.getDom(stateDes_3,i)) &&
          inv_ATM_AccountId(V2J.getDom(stateDes_3,i))) && Utils.is_(V2J.
          getRng(stateDes_3,i),atm.ATMtypes.Account.class)));

      //@ assert Utils.is_(St,atm.ATMtypes.St.class);

      //@ assert St.valid();

      Number ret_8 = newBalance;
      //@ assert Utils.is_real(ret_8);

      return ret_8;
    }
  }

  public static void PrintAccount(final Number id) {

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    final Number balance = ((atm.ATMtypes.Account) Utils.get(St.accounts, id
        )).get_balance();
    //@ assert Utils.is_real(balance);

    IO.printf("Balance␣is␣for␣account␣%s␣is␣%s\n", SeqUtil.seq(id, balance))
        ;
  }

  public static Number GetCurrentCardId() {

    if (!(Utils.equals(Utils.copy(St.get_currentCard()), null))) {
      Number ret_9 = St.get_currentCard().get_id();
      //@ assert ((ret_9 == null) || Utils.is_nat(ret_9));

      return ret_9;

    } else {
      Number ret_10 = null;
      //@ assert ((ret_10 == null) || Utils.is_nat(ret_10));

      return ret_10;
    }
  }

  public static Number TestCurrentCardId() {

    final Number id = GetCurrentCardId();
    //@ assert ((id == null) || Utils.is_nat(id));

    Number ret_11 = id;
    //@ assert ((ret_11 == null) || Utils.is_nat(ret_11));

    return ret_11;
  }

  public static Number TestStatus() {

    final Number accId = 1L;
    //@ assert Utils.is_nat1(accId);

    final atm.ATMtypes.Card c1 = new atm.ATMtypes.Card(1L, 1234L);
    //@ assert Utils.is_(c1,atm.ATMtypes.Card.class);

    {
      AddCard(Utils.copy(c1));
      OpenAccount(SetUtil.set(new atm.ATMtypes.Card(1L, 1234L)), accId);
      {
        final Tuple status = GetStatus();
        //@ assert (V2J.isTup(status,2) && Utils.is_bool(V2J.field(status,0)
            ) && Utils.is_(V2J.field(status,1),String.class));

        final Boolean awaitingCard = ((Boolean) status.get(0));
        //@ assert Utils.is_bool(awaitingCard);

        final String msg = SeqUtil.toStr(status.get(1));
        //@ assert Utils.is_(msg,String.class);

        {
          IO.println("Message:␣" + msg);
          Boolean andResult_8 = false;
          //@ assert Utils.is_bool(andResult_8);

          if (awaitingCard) {
            if (Utils.equals(atm.quotes.AcceptQuote.getInstance(),
                InsertCard(Utils.copy(c1)))) {
              andResult_8 = true;
              //@ assert Utils.is_bool(andResult_8);

            }
          }

          if (andResult_8) {
            NotifyUser(atm.quotes.AcceptQuote.getInstance());
            EnterPin(1234L);
            return Deposit(accId, 100L);
          }
        }
      }

      Number ret_12 = 0L;
      //@ assert Utils.is_real(ret_12);

      return ret_12;
    }
  }

  public static Number TestWithdraw() {

    final Number accId = 1L;
    //@ assert Utils.is_nat1(accId);

    final Number cardId = 1L;
    //@ assert Utils.is_nat1(cardId);

    final Number pin = 1234L;
    //@ assert Utils.is_nat1(pin);

    final atm.ATMtypes.Card c1 = new atm.ATMtypes.Card(cardId, pin);
    //@ assert Utils.is_(c1,atm.ATMtypes.Card.class);

    {
      AddCard(Utils.copy(c1));
      OpenAccount(SetUtil.set(new atm.ATMtypes.Card(1L, 1234L)), accId);
      if (Utils.equals(InsertCard(Utils.copy(c1)), atm.quotes.AcceptQuote.
          getInstance())) {
        EnterPin(pin);
        {
          final Number expense = 600L;
          //@ assert Utils.is_nat1(expense);

          final Number profit = 100L;
          //@ assert Utils.is_nat1(profit);

          {
            final Number amount = expense.longValue() - profit.longValue();
            //@ assert Utils.is_nat1(amount);

            return Withdraw(accId, amount);
          }
        }
      }

      throw new RuntimeException("ERROR␣statement␣reached");
    }
  }

  public static Number TestTotalBalance() {

    final atm.ATMtypes.Card card1 = new atm.ATMtypes.Card(1L, 1234L);
    //@ assert Utils.is_(card1,atm.ATMtypes.Card.class);

    final atm.ATMtypes.Card card2 = new atm.ATMtypes.Card(2L, 5678L);
    //@ assert Utils.is_(card2,atm.ATMtypes.Card.class);

    final atm.ATMtypes.Account ac1 =
        new atm.ATMtypes.Account(SetUtil.set(Utils.copy(card1)), 1000L);
    //@ assert Utils.is_(ac1,atm.ATMtypes.Account.class);

    final atm.ATMtypes.Account ac2 = new atm.ATMtypes.Account(SetUtil.set(
        Utils.copy(card2)), 500L);
    //@ assert Utils.is_(ac2,atm.ATMtypes.Account.class);

    return TotalBalance(SetUtil.set(Utils.copy(ac1), Utils.copy(ac2)));
  }

  public static void TestScenario() {

    final Number accId1 = 1L;
    //@ assert (Utils.is_nat(accId1) && inv_ATM_AccountId(accId1));

    final Number pin1 = 1234L;
    //@ assert Utils.is_nat1(pin1);

    final atm.ATMtypes.Card card1 = new atm.ATMtypes.Card(1L, pin1);
    //@ assert Utils.is_(card1,atm.ATMtypes.Card.class);

    final Number pin2 = 2345L;
    //@ assert Utils.is_nat1(pin2);

    final atm.ATMtypes.Card card2 = new atm.ATMtypes.Card(2L, pin2);
    //@ assert Utils.is_(card2,atm.ATMtypes.Card.class);

    {
      AddCard(Utils.copy(card1));
      AddCard(Utils.copy(card2));
      OpenAccount(SetUtil.set(Utils.copy(card1), Utils.copy(card2)), accId1)
          ;
      {
        final Object ignorePattern_1 = InsertCard(Utils.copy(card2));
        //@ assert (Utils.is_(ignorePattern_1,atm.quotes.AcceptQuote.class)
            || Utils.is_(ignorePattern_1,atm.quotes.BusyQuote.class) ||
            Utils.is_(ignorePattern_1,atm.quotes.RejectQuote.class));

        /* skip */
      }

      PrintAccount(accId1);
      EnterPin(2345L);
      {
        final Number ignorePattern_2 = Deposit(accId1, 200L);
        //@ assert Utils.is_real(ignorePattern_2);

        /* skip */
      }

      PrintAccount(accId1);
      ReturnCard();
      RemoveCard(Utils.copy(card1));
      RemoveCard(Utils.copy(card2));
    }
  }
  /*@ pure @*/

  public static Number TotalBalance(final VDMSet acs) {

    //@ assert (V2J.isSet(acs) && (\forall int i; 0 <= i && i < V2J.size(acs
        ); Utils.is_(V2J.get(acs,i),atm.ATMtypes.Account.class)));

    if (Utils.empty(acs)) {
      Number ret_13 = 0L;
      //@ assert Utils.is_real(ret_13);

      return ret_13;

    } else {
      Number letBeStExp_1 = null;
      atm.ATMtypes.Account a = null;

      Boolean success_1 = false;
      //@ assert Utils.is_bool(success_1);

      VDMSet set_1 = Utils.copy(acs);
      //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size
          (set_1); Utils.is_(V2J.get(set_1,i),atm.ATMtypes.Account.class)));

      for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() &&
          !(success_1); ) {
        a = ((atm.ATMtypes.Account) iterator_1.next());
        success_1 = true;
        //@ assert Utils.is_bool(success_1);

      }
      if (!(success_1)) {
        throw new RuntimeException("Let␣Be␣St␣found␣no␣applicable␣bindings")
            ;
      }

      letBeStExp_1 =
          a.get_balance().doubleValue()
              + TotalBalance(SetUtil.diff(Utils.copy(acs), SetUtil.set(Utils
                  .copy(a))))
                  .doubleValue();
      //@ assert Utils.is_real(letBeStExp_1);

      Number ret_14 = letBeStExp_1;
      //@ assert Utils.is_real(ret_14);

      return ret_14;
    }
  }
  /*@ pure @*/

  public static Number TotalBalanceMes(final VDMSet acs) {

    //@ assert (V2J.isSet(acs) && (\forall int i; 0 <= i && i < V2J.size(acs
        ); Utils.is_(V2J.get(acs,i),atm.ATMtypes.Account.class)));

    Number ret_15 = acs.size();
    //@ assert Utils.is_nat(ret_15);

    return ret_15;
  }
  /*@ pure @*/

  public static Boolean pre_OpenAccount(
      final VDMSet cards, final Number id, final atm.ATMtypes.St St) {

    //@ assert (V2J.isSet(cards) && (\forall int i; 0 <= i && i < V2J.size(
        cards); Utils.is_(V2J.get(cards,i),atm.ATMtypes.Card.class)));

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_16 = !(SetUtil.inSet(id, MapUtil.dom(Utils.copy(St.
        get_accounts()))));
    //@ assert Utils.is_bool(ret_16);

    return ret_16;
  }
  /*@ pure @*/

  public static Boolean post_OpenAccount(
      final VDMSet cards, final Number id, final atm.ATMtypes.St _St, final
          atm.ATMtypes.St St) {

    //@ assert (V2J.isSet(cards) && (\forall int i; 0 <= i && i < V2J.size(
        cards); Utils.is_(V2J.get(cards,i),atm.ATMtypes.Card.class)));

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    //@ assert Utils.is_(_St,atm.ATMtypes.St.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean andResult_1 = false;
    //@ assert Utils.is_bool(andResult_1);

    if (SetUtil.inSet(id, MapUtil.dom(Utils.copy(St.get_accounts())))) {
      if (Utils.equals(((atm.ATMtypes.Account) Utils.get(St.accounts, id)).
          get_balance(), 0L)) {
        andResult_1 = true;
        //@ assert Utils.is_bool(andResult_1);

      }
    }

    Boolean ret_17 = andResult_1;
    //@ assert Utils.is_bool(ret_17);

    return ret_17;
  }
  /*@ pure @*/

  public static Boolean pre_AddCard(final atm.ATMtypes.Card c, final atm.
      ATMtypes.St St) {

```

```
    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_18 = !(SetUtil.inSet(c, Utils.copy(St.get_validCards())));
    //@ assert Utils.is_bool(ret_18);

    return ret_18;
  }
  /*@ pure @*/

  public static Boolean post_AddCard(
      final atm.ATMtypes.Card c, final atm.ATMtypes.St _St, final atm.
          ATMtypes.St St) {

    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    //@ assert Utils.is_(_St,atm.ATMtypes.St.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_19 = SetUtil.inSet(c, Utils.copy(St.get_validCards()));
    //@ assert Utils.is_bool(ret_19);

    return ret_19;
  }
  /*@ pure @*/

  public static Boolean pre_RemoveCard(final atm.ATMtypes.Card c, final atm.
      ATMtypes.St St) {

    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_20 = SetUtil.inSet(c, Utils.copy(St.get_validCards()));
    //@ assert Utils.is_bool(ret_20);

    return ret_20;
  }
  /*@ pure @*/

  public static Boolean post_RemoveCard(
      final atm.ATMtypes.Card c, final atm.ATMtypes.St _St, final atm.
          ATMtypes.St St) {

    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    //@ assert Utils.is_(_St,atm.ATMtypes.St.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_21 = !(SetUtil.inSet(c, Utils.copy(St.get_validCards())));
    //@ assert Utils.is_bool(ret_21);

    return ret_21;
  }
  /*@ pure @*/

  public static Boolean pre_InsertCard(final atm.ATMtypes.Card c, final atm.
      ATMtypes.St St) {

    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_22 = Utils.equals(Utils.copy(St.get_currentCard()), null);
    //@ assert Utils.is_bool(ret_22);

    return ret_22;
  }
  /*@ pure @*/

  public static Boolean post_InsertCard(
      final atm.ATMtypes.Card c,
      final Object RESULT,
      final atm.ATMtypes.St _St,
      final atm.ATMtypes.St St) {

    //@ assert Utils.is_(c,atm.ATMtypes.Card.class);

    //@ assert (Utils.is_(RESULT,atm.quotes.AcceptQuote.class) || Utils.is_(
        RESULT,atm.quotes.BusyQuote.class) || Utils.is_(RESULT,atm.quotes.
        RejectQuote.class));

    //@ assert Utils.is_(_St,atm.ATMtypes.St.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    if (Utils.equals(RESULT, atm.quotes.AcceptQuote.getInstance())) {
      Boolean ret_23 = Utils.equals(Utils.copy(St.get_currentCard()), c);
      //@ assert Utils.is_bool(ret_23);

      return ret_23;

    } else {
      if (Utils.equals(RESULT, atm.quotes.BusyQuote.getInstance())) {
        Boolean ret_24 =
            Utils.equals(Utils.copy(St.get_currentCard()), Utils.copy(_St.
                get_currentCard()));
        //@ assert Utils.is_bool(ret_24);

        return ret_24;

      } else {
        Boolean ret_25 = Utils.equals(Utils.copy(St.get_currentCard()), null
            );
        //@ assert Utils.is_bool(ret_25);

        return ret_25;
      }
    }
  }
  /*@ pure @*/

  public static Boolean pre_EnterPin(final Number pin, final atm.ATMtypes.St
       St) {

    //@ assert (Utils.is_nat(pin) && inv_ATM_Pin(pin));

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_26 = !(Utils.equals(Utils.copy(St.get_currentCard()), null))
        ;
    //@ assert Utils.is_bool(ret_26);

    return ret_26;
  }
  /*@ pure @*/

  public static Boolean pre_ReturnCard(final atm.ATMtypes.St St) {

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_27 = !(Utils.equals(Utils.copy(St.get_currentCard()), null))
        ;
    //@ assert Utils.is_bool(ret_27);

    return ret_27;
  }
  /*@ pure @*/

  public static Boolean post_ReturnCard(final atm.ATMtypes.St _St, final atm
      .ATMtypes.St St) {

    //@ assert Utils.is_(_St,atm.ATMtypes.St.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean andResult_2 = false;
    //@ assert Utils.is_bool(andResult_2);

    if (Utils.equals(Utils.copy(St.get_currentCard()), null)) {
      if (!(St.get_pinOk())) {
        andResult_2 = true;
        //@ assert Utils.is_bool(andResult_2);

      }
    }

    Boolean ret_28 = andResult_2;
    //@ assert Utils.is_bool(ret_28);

    return ret_28;
  }
  /*@ pure @*/

  public static Boolean pre_Withdraw(
      final Number id, final Number amount, final atm.ATMtypes.St St) {

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    //@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean andResult_3 = false;
    //@ assert Utils.is_bool(andResult_3);

    if (SetUtil.inSet(Utils.copy(St.get_currentCard()), Utils.copy(St.
        get_validCards()))) {
      Boolean andResult_4 = false;
      //@ assert Utils.is_bool(andResult_4);

      if (St.get_pinOk()) {
        Boolean andResult_5 = false;
        //@ assert Utils.is_bool(andResult_5);

        if (SetUtil.inSet(
            Utils.copy(St.get_currentCard()),
            Utils.copy(((atm.ATMtypes.Account) Utils.get(St.accounts, id)).
                get_cards()))) {
          if (SetUtil.inSet(id, MapUtil.dom(Utils.copy(St.get_accounts()))))
              {
            andResult_5 = true;
            //@ assert Utils.is_bool(andResult_5);

          }
        }

        if (andResult_5) {
          andResult_4 = true;
          //@ assert Utils.is_bool(andResult_4);

        }
      }

      if (andResult_4) {
        andResult_3 = true;
        //@ assert Utils.is_bool(andResult_3);

      }
    }

    Boolean ret_29 = andResult_3;
    //@ assert Utils.is_bool(ret_29);

    return ret_29;
  }
  /*@ pure @*/

  public static Boolean post_Withdraw(
      final Number id,
      final Number amount,
      final Number RESULT,
      final atm.ATMtypes.St _St,
      final atm.ATMtypes.St St) {

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    //@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));

    //@ assert Utils.is_real(RESULT);

    //@ assert Utils.is_(_St,atm.ATMtypes.St.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    final atm.ATMtypes.Account accountPre =
        Utils.copy(((atm.ATMtypes.Account) Utils.get(_St.accounts, id)));
    //@ assert Utils.is_(accountPre,atm.ATMtypes.Account.class);

    final atm.ATMtypes.Account accountPost =
        Utils.copy(((atm.ATMtypes.Account) Utils.get(St.accounts, id)));
    //@ assert Utils.is_(accountPost,atm.ATMtypes.Account.class);

    Boolean andResult_6 = false;
    //@ assert Utils.is_bool(andResult_6);

    if (Utils.equals(
        accountPre.get_balance(), accountPost.get_balance().doubleValue() +
            amount.longValue())) {
      if (Utils.equals(accountPost.get_balance(), RESULT)) {
        andResult_6 = true;
        //@ assert Utils.is_bool(andResult_6);

      }
    }

    Boolean ret_30 = andResult_6;
    //@ assert Utils.is_bool(ret_30);

    return ret_30;
  }
  /*@ pure @*/

  public static Boolean pre_Deposit(
      final Number id, final Number amount, final atm.ATMtypes.St St) {

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    //@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    Boolean ret_31 = pre_Withdraw(id, amount, Utils.copy(St));
    //@ assert Utils.is_bool(ret_31);

    return ret_31;
  }
  /*@ pure @*/

  public static Boolean post_Deposit(
      final Number id,
      final Number amount,
      final Number RESULT,
      final atm.ATMtypes.St _St,
      final atm.ATMtypes.St St) {

    //@ assert (Utils.is_nat(id) && inv_ATM_AccountId(id));

    //@ assert (Utils.is_nat1(amount) && inv_ATM_Amount(amount));

    //@ assert Utils.is_real(RESULT);

    //@ assert Utils.is_(_St,atm.ATMtypes.St.class);

    //@ assert Utils.is_(St,atm.ATMtypes.St.class);

    final atm.ATMtypes.Account accountPre =
        Utils.copy(((atm.ATMtypes.Account) Utils.get(_St.accounts, id)));
    //@ assert Utils.is_(accountPre,atm.ATMtypes.Account.class);

    final atm.ATMtypes.Account accountPost =
        Utils.copy(((atm.ATMtypes.Account) Utils.get(St.accounts, id)));
    //@ assert Utils.is_(accountPost,atm.ATMtypes.Account.class);

    Boolean andResult_7 = false;
    //@ assert Utils.is_bool(andResult_7);

    if (Utils.equals(
        accountPre.get_balance().doubleValue() + amount.longValue(),
            accountPost.get_balance())) {
      if (Utils.equals(accountPost.get_balance(), RESULT)) {
        andResult_7 = true;
        //@ assert Utils.is_bool(andResult_7);

      }
    }

    Boolean ret_32 = andResult_7;
    //@ assert Utils.is_bool(ret_32);

    return ret_32;
  }

  public String toString() {

    return "ATM{" + "St␣:=␣" + Utils.toString(St) + "}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_Pin(final Object check_p) {

    Number p = ((Number) check_p);

    Boolean andResult_9 = false;

    if (0L <= p.longValue()) {
      if (p.longValue() <= 9999L) {
        andResult_9 = true;
      }
    }

    return andResult_9;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_AccountId(final Object check_id) {

    Number id = ((Number) check_id);

    return id.longValue() > 0L;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_Amount(final Object check_a) {

    Number a = ((Number) check_a);

    return a.longValue() < 2000L;
  }
}
```

```
package atm.ATMtypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class St implements Record, java.io.Serializable {
  public VDMSet validCards;
  public atm.ATMtypes.Card currentCard;
  public Boolean pinOk;
  public VDMMap accounts;
  //@ public instance invariant atm.ATM.invChecksOn ==> inv_St(validCards,
      currentCard,pinOk,accounts);

  public St(
      final VDMSet _validCards,
      final atm.ATMtypes.Card _currentCard,
      final Boolean _pinOk,
      final VDMMap _accounts) {

    //@ assert (V2J.isSet(_validCards) && (\forall int i; 0 <= i && i < V2J.
        size(_validCards); Utils.is_(V2J.get(_validCards,i),atm.ATMtypes.
        Card.class)));

    //@ assert ((_currentCard == null) || Utils.is_(_currentCard,atm.
        ATMtypes.Card.class));

    //@ assert Utils.is_bool(_pinOk);

    //@ assert (V2J.isMap(_accounts) && (\forall int i; 0 <= i && i < V2J.
        size(_accounts); (Utils.is_nat(V2J.getDom(_accounts,i)) &&
        inv_ATM_AccountId(V2J.getDom(_accounts,i))) && Utils.is_(V2J.getRng(
        _accounts,i),atm.ATMtypes.Account.class)));

    validCards = _validCards != null ? Utils.copy(_validCards) : null;
    //@ assert (V2J.isSet(validCards) && (\forall int i; 0 <= i && i < V2J.
        size(validCards); Utils.is_(V2J.get(validCards,i),atm.ATMtypes.Card.
        class)));

    currentCard = _currentCard != null ? Utils.copy(_currentCard) : null;
    //@ assert ((currentCard == null) || Utils.is_(currentCard,atm.ATMtypes.
        Card.class));

    pinOk = _pinOk;
    //@ assert Utils.is_bool(pinOk);

    accounts = _accounts != null ? Utils.copy(_accounts) : null;
    //@ assert (V2J.isMap(accounts) && (\forall int i; 0 <= i && i < V2J.
        size(accounts); (Utils.is_nat(V2J.getDom(accounts,i)) &&
        inv_ATM_AccountId(V2J.getDom(accounts,i))) && Utils.is_(V2J.getRng(
        accounts,i),atm.ATMtypes.Account.class)));

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof atm.ATMtypes.St)) {
      return false;
    }

    atm.ATMtypes.St other = ((atm.ATMtypes.St) obj);

    return (Utils.equals(validCards, other.validCards))
        && (Utils.equals(currentCard, other.currentCard))
        && (Utils.equals(pinOk, other.pinOk))
        && (Utils.equals(accounts, other.accounts));
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(validCards, currentCard, pinOk, accounts);
  }
  /*@ pure @*/

  public atm.ATMtypes.St copy() {

    return new atm.ATMtypes.St(validCards, currentCard, pinOk, accounts);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_ATM`St" + Utils.formatFields(validCards, currentCard, pinOk,
        accounts);
  }
  /*@ pure @*/

  public VDMSet get_validCards() {

    VDMSet ret_37 = validCards;
    //@ assert atm.ATM.invChecksOn ==> ((V2J.isSet(ret_37) && (\forall int i
        ; 0 <= i && i < V2J.size(ret_37); Utils.is_(V2J.get(ret_37,i),atm.
        ATMtypes.Card.class))));

    return ret_37;
  }

  public void set_validCards(final VDMSet _validCards) {

    //@ assert atm.ATM.invChecksOn ==> ((V2J.isSet(_validCards) && (\forall
        int i; 0 <= i && i < V2J.size(_validCards); Utils.is_(V2J.get(
        _validCards,i),atm.ATMtypes.Card.class))));

    validCards = _validCards;
    //@ assert atm.ATM.invChecksOn ==> ((V2J.isSet(validCards) && (\forall
      int i; 0 <= i && i < V2J.size(validCards); Utils.is_(V2J.get(
      validCards,i),atm.ATMtypes.Card.class))));

  }
  /*@ pure @*/

  public atm.ATMtypes.Card get_currentCard() {

    atm.ATMtypes.Card ret_38 = currentCard;
    //@ assert atm.ATM.invChecksOn ==> (((ret_38 == null) || Utils.is_(
        ret_38,atm.ATMtypes.Card.class)));

    return ret_38;
  }

  public void set_currentCard(final atm.ATMtypes.Card _currentCard) {

    //@ assert atm.ATM.invChecksOn ==> (((_currentCard == null) || Utils.is_
        (_currentCard,atm.ATMtypes.Card.class)));

    currentCard = _currentCard;
    //@ assert atm.ATM.invChecksOn ==> (((currentCard == null) || Utils.is_(
        currentCard,atm.ATMtypes.Card.class)));

  }
  /*@ pure @*/

  public Boolean get_pinOk() {

    Boolean ret_39 = pinOk;
    //@ assert atm.ATM.invChecksOn ==> (Utils.is_bool(ret_39));

    return ret_39;
  }

  public void set_pinOk(final Boolean _pinOk) {

    //@ assert atm.ATM.invChecksOn ==> (Utils.is_bool(_pinOk));

    pinOk = _pinOk;
    //@ assert atm.ATM.invChecksOn ==> (Utils.is_bool(pinOk));

  }
  /*@ pure @*/

  public VDMMap get_accounts() {

    VDMMap ret_40 = accounts;
    //@ assert atm.ATM.invChecksOn ==> ((V2J.isMap(ret_40) && (\forall int i
        ; 0 <= i && i < V2J.size(ret_40); (Utils.is_nat(V2J.getDom(ret_40,i)
        ) && inv_ATM_AccountId(V2J.getDom(ret_40,i))) && Utils.is_(V2J.
        getRng(ret_40,i),atm.ATMtypes.Account.class))));

    return ret_40;
  }

  public void set_accounts(final VDMMap _accounts) {

    //@ assert atm.ATM.invChecksOn ==> ((V2J.isMap(_accounts) && (\forall
        int i; 0 <= i && i < V2J.size(_accounts); (Utils.is_nat(V2J.getDom(
        _accounts,i)) && inv_ATM_AccountId(V2J.getDom(_accounts,i))) &&
        Utils.is_(V2J.getRng(_accounts,i),atm.ATMtypes.Account.class))));

    accounts = _accounts;
    //@ assert atm.ATM.invChecksOn ==> ((V2J.isMap(accounts) && (\forall int
         i; 0 <= i && i < V2J.size(accounts); (Utils.is_nat(V2J.getDom(
        accounts,i)) && inv_ATM_AccountId(V2J.getDom(accounts,i))) && Utils.
        is_(V2J.getRng(accounts,i),atm.ATMtypes.Account.class))));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_St(
      final VDMSet _validCards,
      final atm.ATMtypes.Card _currentCard,
      final Boolean _pinOk,
      final VDMMap _accounts) {

    Boolean success_2 = true;
    VDMSet v = null;
    atm.ATMtypes.Card c = null;
    Boolean p = null;
    VDMMap a = null;
    v = _validCards;
    c = _currentCard;
    p = _pinOk;
    a = _accounts;

    if (!(success_2)) {
      throw new RuntimeException("Record␣pattern␣match␣failed");
    }

    Boolean andResult_10 = false;

    Boolean orResult_1 = false;

    Boolean orResult_2 = false;

    if (p) {
      orResult_2 = true;
    } else {
      orResult_2 = !(Utils.equals(c, null));
    }

    if (!(orResult_2)) {
      orResult_1 = true;
    } else {
      orResult_1 = SetUtil.inSet(c, v);
    }

    if (orResult_1) {
      Boolean forAllExpResult_1 = true;
      VDMSet set_2 = MapUtil.dom(a);
      for (Iterator iterator_2 = set_2.iterator(); iterator_2.hasNext() && forAllExpResult_1; ) {
        Number id1 = ((Number) iterator_2.next());
        for (Iterator iterator_3 = set_2.iterator(); iterator_3.hasNext() && forAllExpResult_1; ) {
          Number id2 = ((Number) iterator_3.next());
          Boolean orResult_3 = false;

          if (!(!(Utils.equals(id1, id2)))) {
            orResult_3 = true;
          } else {
            orResult_3 =
                Utils.empty(
                    SetUtil.intersect(
                        Utils.copy(((atm.ATMtypes.Account) Utils.get(a, id1)).cards),
                        Utils.copy(((atm.ATMtypes.Account) Utils.get(a, id2)).cards)));
          }

          forAllExpResult_1 = orResult_3;
        }
      }
      if (forAllExpResult_1) {
        andResult_10 = true;
      }
    }

    return andResult_10;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_Pin(final Object check_p) {

    Number p = ((Number) check_p);

    Boolean andResult_9 = false;

    if (0L <= p.longValue()) {
      if (p.longValue() <= 9999L) {
        andResult_9 = true;
      }
    }

    return andResult_9;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_AccountId(final Object check_id) {

    Number id = ((Number) check_id);

    return id.longValue() > 0L;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_Amount(final Object check_a) {

    Number a = ((Number) check_a);

    return a.longValue() < 2000L;
  }
}
```

```
package atm.ATMtypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Account implements Record, java.io.Serializable {
  public VDMSet cards;
  public Number balance;
  //@ public instance invariant atm.ATM.invChecksOn ==> inv_Account(cards,
      balance);

  public Account(final VDMSet _cards, final Number _balance) {

    //@ assert (V2J.isSet(_cards) && (\forall int i; 0 <= i && i < V2J.size(
        _cards); Utils.is_(V2J.get(_cards,i),atm.ATMtypes.Card.class)));

    //@ assert Utils.is_real(_balance);

    cards = _cards != null ? Utils.copy(_cards) : null;
    //@ assert (V2J.isSet(cards) && (\forall int i; 0 <= i && i < V2J.size(
        cards); Utils.is_(V2J.get(cards,i),atm.ATMtypes.Card.class)));

    balance = _balance;
    //@ assert Utils.is_real(balance);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof atm.ATMtypes.Account)) {
      return false;
    }

    atm.ATMtypes.Account other = ((atm.ATMtypes.Account) obj);

    return (Utils.equals(cards, other.cards)) && (Utils.equals(balance,
        other.balance));
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(cards, balance);
  }
  /*@ pure @*/

  public atm.ATMtypes.Account copy() {

    return new atm.ATMtypes.Account(cards, balance);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_ATM`Account" + Utils.formatFields(cards, balance);
  }
  /*@ pure @*/

  public VDMSet get_cards() {

    VDMSet ret_35 = cards;
    //@ assert atm.ATM.invChecksOn ==> ((V2J.isSet(ret_35) && (\forall int i
        ; 0 <= i && i < V2J.size(ret_35); Utils.is_(V2J.get(ret_35,i),atm.
        ATMtypes.Card.class))));

    return ret_35;
  }

  public void set_cards(final VDMSet _cards) {

    //@ assert atm.ATM.invChecksOn ==> ((V2J.isSet(_cards) && (\forall int i
        ; 0 <= i && i < V2J.size(_cards); Utils.is_(V2J.get(_cards,i),atm.
        ATMtypes.Card.class))));

    cards = _cards;
    //@ assert atm.ATM.invChecksOn ==> ((V2J.isSet(cards) && (\forall int i;
         0 <= i && i < V2J.size(cards); Utils.is_(V2J.get(cards,i),atm.
        ATMtypes.Card.class))));

  }
  /*@ pure @*/

  public Number get_balance() {

    Number ret_36 = balance;
    //@ assert atm.ATM.invChecksOn ==> (Utils.is_real(ret_36));

    return ret_36;
  }

  public void set_balance(final Number _balance) {

    //@ assert atm.ATM.invChecksOn ==> (Utils.is_real(_balance));

    balance = _balance;
    //@ assert atm.ATM.invChecksOn ==> (Utils.is_real(balance));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Account(final VDMSet _cards, final Number
      _balance) {

    return _balance.doubleValue() >= -1000L;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_Pin(final Object check_p) {

    Number p = ((Number) check_p);

    Boolean andResult_9 = false;

    if (0L <= p.longValue()) {
      if (p.longValue() <= 9999L) {
        andResult_9 = true;
      }
    }

    return andResult_9;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_AccountId(final Object check_id) {

    Number id = ((Number) check_id);

    return id.longValue() > 0L;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_Amount(final Object check_a) {

    Number a = ((Number) check_a);

    return a.longValue() < 2000L;
  }
}
```

```
package atm.ATMtypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Card implements Record, java.io.Serializable {
  public Number id;
  public Number pin;

  public Card(final Number _id, final Number _pin) {

    //@ assert Utils.is_nat(_id);

    //@ assert (Utils.is_nat(_pin) && inv_ATM_Pin(_pin));

    id = _id;
    //@ assert Utils.is_nat(id);

    pin = _pin;
    //@ assert (Utils.is_nat(pin) && inv_ATM_Pin(pin));

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof atm.ATMtypes.Card)) {
      return false;
    }

    atm.ATMtypes.Card other = ((atm.ATMtypes.Card) obj);

    return (Utils.equals(id, other.id)) && (Utils.equals(pin, other.pin));
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(id, pin);
  }
  /*@ pure @*/

  public atm.ATMtypes.Card copy() {

    return new atm.ATMtypes.Card(id, pin);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_ATM`Card" + Utils.formatFields(id, pin);
  }
  /*@ pure @*/

  public Number get_id() {

    Number ret_33 = id;
    //@ assert atm.ATM.invChecksOn ==> (Utils.is_nat(ret_33));

    return ret_33;
  }

  public void set_id(final Number _id) {

    //@ assert atm.ATM.invChecksOn ==> (Utils.is_nat(_id));

    id = _id;
    //@ assert atm.ATM.invChecksOn ==> (Utils.is_nat(id));

  }
  /*@ pure @*/

  public Number get_pin() {

    Number ret_34 = pin;
    //@ assert atm.ATM.invChecksOn ==> ((Utils.is_nat(ret_34) && inv_ATM_Pin
        (ret_34)));

    return ret_34;
  }

  public void set_pin(final Number _pin) {

    //@ assert atm.ATM.invChecksOn ==> ((Utils.is_nat(_pin) && inv_ATM_Pin(
        _pin)));

    pin = _pin;
    //@ assert atm.ATM.invChecksOn ==> ((Utils.is_nat(pin) && inv_ATM_Pin(
        pin)));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_Pin(final Object check_p) {

    Number p = ((Number) check_p);

    Boolean andResult_9 = false;

    if (0L <= p.longValue()) {
      if (p.longValue() <= 9999L) {
        andResult_9 = true;
      }
    }

    return andResult_9;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_AccountId(final Object check_id) {

    Number id = ((Number) check_id);

    return id.longValue() > 0L;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_ATM_Amount(final Object check_a) {

    Number a = ((Number) check_a);

    return a.longValue() < 2000L;
  }
}
```

```
package atm.quotes;

import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class startQuote implements java.io.Serializable {
  private static int hc = 0;
  private static startQuote instance = null;

  public startQuote() {

    if (Utils.equals(hc, 0)) {
      hc = super.hashCode();
    }
  }

  public static startQuote getInstance() {

    if (Utils.equals(instance, null)) {
      instance = new startQuote();
    }

    return instance;
  }

  public int hashCode() {

    return hc;
  }

  public boolean equals(final Object obj) {

    return obj instanceof startQuote;
  }

  public String toString() {

    return "<start>";
  }
}
```

```
package atm.quotes;

import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default
```

```java

final public class RejectQuote implements java.io.Serializable {
  private static int hc = 0;
  private static RejectQuote instance = null;

  public RejectQuote() {

    if (Utils.equals(hc, 0)) {
      hc = super.hashCode();
    }
  }

  public static RejectQuote getInstance() {

    if (Utils.equals(instance, null)) {
      instance = new RejectQuote();
    }

    return instance;
  }

  public int hashCode() {

    return hc;
  }

  public boolean equals(final Object obj) {

    return obj instanceof RejectQuote;
  }

  public String toString() {

    return "<Reject>";
  }
}
```

```java
package atm.quotes;

import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class AcceptQuote implements java.io.Serializable {
  private static int hc = 0;
  private static AcceptQuote instance = null;

  public AcceptQuote() {

    if (Utils.equals(hc, 0)) {
      hc = super.hashCode();
    }
  }

  public static AcceptQuote getInstance() {

    if (Utils.equals(instance, null)) {
      instance = new AcceptQuote();
    }

    return instance;
  }

  public int hashCode() {

    return hc;
  }

  public boolean equals(final Object obj) {

    return obj instanceof AcceptQuote;
  }

  public String toString() {

    return "<Accept>";
  }
}
```

```
package atm.quotes;

import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class appendQuote implements java.io.Serializable {
  private static int hc = 0;
  private static appendQuote instance = null;

  public appendQuote() {

    if (Utils.equals(hc, 0)) {
      hc = super.hashCode();
    }
  }

  public static appendQuote getInstance() {

    if (Utils.equals(instance, null)) {
      instance = new appendQuote();
    }

    return instance;
  }

  public int hashCode() {

    return hc;
  }

  public boolean equals(final Object obj) {

    return obj instanceof appendQuote;
  }

  public String toString() {

    return "<append>";
  }
}
```

```
package atm.quotes;

import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class BusyQuote implements java.io.Serializable {
  private static int hc = 0;
  private static BusyQuote instance = null;

  public BusyQuote() {

    if (Utils.equals(hc, 0)) {
      hc = super.hashCode();
    }
  }

  public static BusyQuote getInstance() {

    if (Utils.equals(instance, null)) {
      instance = new BusyQuote();
    }

    return instance;
  }

  public int hashCode() {

    return hc;
  }

  public boolean equals(final Object obj) {

    return obj instanceof BusyQuote;
  }

  public String toString() {

    return "<Busy>";
  }
}
```

C

Validation of the translation rules

Since the translation is not formally defined all the translation rules have been validated by running examples, or *regression tests*, through the tool (see section 2.8 for details). This appendix contains all the regression tests that are used to test the translation rules. For each test, the input model, the corresponding Java/JML and the OpenJML runtime assertion checker output is shown, respectively. Alternatively, all the examples, including the generated output, can be downloaded via [2].

C.1 Map.vdmsl

C.1.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣legal␣use");
  let - : inmap nat to nat = {x |-> x | x in set {1,2,3} & x > 0} in skip;
  let - : inmap nat to nat = {x |-> x | x in set {1,2,3}} in skip;
  IO`println("After␣legal␣use");
  IO`println("Before␣violations");
  let - : inmap nat to nat = {x |-> 2 | x in set {1,2,3} & x > 1} in skip;
  let - : inmap nat to nat = {x |-> 2 | x in set {1,2,3}} in skip;
  IO`println("After␣violations");
  return 0;
);

end Entry
```

C.1.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    VDMMap mapCompResult_1 = MapUtil.map();
    //@ assert (V2J.isMap(mapCompResult_1) && (\forall int i; 0 <= i && i <
        V2J.size(mapCompResult_1); Utils.is_nat1(V2J.getDom(mapCompResult_1,
        i)) && Utils.is_nat1(V2J.getRng(mapCompResult_1,i))));

    VDMSet set_1 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_nat1(V2J.get(set_1,i))));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext(); ) {
      Number x = ((Number) iterator_1.next());
      //@ assert Utils.is_nat1(x);

      if (x.longValue() > 0L) {
        MapUtil.mapAdd(mapCompResult_1, new Maplet(x, x));
      }
    }
    {
      final VDMMap ignorePattern_1 = Utils.copy(mapCompResult_1);
      //@ assert (V2J.isInjMap(ignorePattern_1) && (\forall int i; 0 <= i &&
           i < V2J.size(ignorePattern_1); Utils.is_nat(V2J.getDom(
          ignorePattern_1,i)) && Utils.is_nat(V2J.getRng(ignorePattern_1,i))
          ));

      /* skip */
    }

    VDMMap mapCompResult_2 = MapUtil.map();
    //@ assert (V2J.isMap(mapCompResult_2) && (\forall int i; 0 <= i && i <
        V2J.size(mapCompResult_2); Utils.is_nat1(V2J.getDom(mapCompResult_2,
        i)) && Utils.is_nat1(V2J.getRng(mapCompResult_2,i))));

    VDMSet set_2 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_2) && (\forall int i; 0 <= i && i < V2J.size(
        set_2); Utils.is_nat1(V2J.get(set_2,i))));

    for (Iterator iterator_2 = set_2.iterator(); iterator_2.hasNext(); ) {
      Number x = ((Number) iterator_2.next());
      //@ assert Utils.is_nat1(x);

      MapUtil.mapAdd(mapCompResult_2, new Maplet(x, x));
    }
    {
      final VDMMap ignorePattern_2 = Utils.copy(mapCompResult_2);
      //@ assert (V2J.isInjMap(ignorePattern_2) && (\forall int i; 0 <= i &&
           i < V2J.size(ignorePattern_2); Utils.is_nat(V2J.getDom(
          ignorePattern_2,i)) && Utils.is_nat(V2J.getRng(ignorePattern_2,i))
          ));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣violations");
    VDMMap mapCompResult_3 = MapUtil.map();
    //@ assert (V2J.isMap(mapCompResult_3) && (\forall int i; 0 <= i && i <
        V2J.size(mapCompResult_3); Utils.is_nat1(V2J.getDom(mapCompResult_3,
        i)) && Utils.is_nat1(V2J.getRng(mapCompResult_3,i))));

    VDMSet set_3 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_3) && (\forall int i; 0 <= i && i < V2J.size(
        set_3); Utils.is_nat1(V2J.get(set_3,i))));

    for (Iterator iterator_3 = set_3.iterator(); iterator_3.hasNext(); ) {
      Number x = ((Number) iterator_3.next());
      //@ assert Utils.is_nat1(x);

      if (x.longValue() > 1L) {
        MapUtil.mapAdd(mapCompResult_3, new Maplet(x, 2L));
      }
    }
    {
      final VDMMap ignorePattern_3 = Utils.copy(mapCompResult_3);
      //@ assert (V2J.isInjMap(ignorePattern_3) && (\forall int i; 0 <= i &&
           i < V2J.size(ignorePattern_3); Utils.is_nat(V2J.getDom(
          ignorePattern_3,i)) && Utils.is_nat(V2J.getRng(ignorePattern_3,i))
          ));

      /* skip */
    }

    VDMMap mapCompResult_4 = MapUtil.map();
    //@ assert (V2J.isMap(mapCompResult_4) && (\forall int i; 0 <= i && i <
        V2J.size(mapCompResult_4); Utils.is_nat1(V2J.getDom(mapCompResult_4,
        i)) && Utils.is_nat1(V2J.getRng(mapCompResult_4,i))));

    VDMSet set_4 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_4) && (\forall int i; 0 <= i && i < V2J.size(
        set_4); Utils.is_nat1(V2J.get(set_4,i))));

    for (Iterator iterator_4 = set_4.iterator(); iterator_4.hasNext(); ) {
      Number x = ((Number) iterator_4.next());
      //@ assert Utils.is_nat1(x);

      MapUtil.mapAdd(mapCompResult_4, new Maplet(x, 2L));
    }
    {
      final VDMMap ignorePattern_4 = Utils.copy(mapCompResult_4);
      //@ assert (V2J.isInjMap(ignorePattern_4) && (\forall int i; 0 <= i &&
           i < V2J.size(ignorePattern_4); Utils.is_nat(V2J.getDom(
          ignorePattern_4,i)) && Utils.is_nat(V2J.getRng(ignorePattern_4,i))
        ));

      /* skip */
    }

    IO.println("After␣violations");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.1.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before violations"
Entry.java:76: JML assertion is false
      //@ assert (V2J.isInjMap(ignorePattern_3) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_3); Utils.is_nat(V2J.getDom(ignorePattern_3
          ,i)) && Utils.is_nat(V2J.getRng(ignorePattern_3,i))));
          ^
Entry.java:95: JML assertion is false
      //@ assert (V2J.isInjMap(ignorePattern_4) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_4); Utils.is_nat(V2J.getDom(ignorePattern_4
          ,i)) && Utils.is_nat(V2J.getRng(ignorePattern_4,i))));
          ^
"After violations"
```

C.2 Seq.vdmsl

C.2.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣legal␣use");
  let - : seq1 of nat = [x | x in set {1,2,3} & x > 0] in skip;
  let - : seq1 of nat = [x | x in set {1,2,3}] in skip;
  IO`println("After␣legal␣use");
  IO`println("Before␣violations");
  let - : seq1 of nat = [x | x in set {1,2,3} & x > 4] in skip;
  let - : seq1 of nat = [x | x in set xs()] in skip;
  IO`println("After␣violations");
  return 0;
);

functions

xs :  () -> set of nat
xs () == {};

end Entry
```

C.2.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    VDMSeq seqCompResult_1 = SeqUtil.seq();
    //@ assert (V2J.isSeq(seqCompResult_1) && (\forall int i; 0 <= i && i <
        V2J.size(seqCompResult_1); Utils.is_nat1(V2J.get(seqCompResult_1,i))
        ));

    VDMSet set_1 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_nat1(V2J.get(set_1,i))));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext(); ) {
      Number x = ((Number) iterator_1.next());
      //@ assert Utils.is_nat1(x);

      if (x.longValue() > 0L) {
        seqCompResult_1.add(x);
      }
    }
    {
      final VDMSeq ignorePattern_1 = Utils.copy(seqCompResult_1);
      //@ assert (V2J.isSeq1(ignorePattern_1) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_1); Utils.is_nat(V2J.get(ignorePattern_1
          ,i))));

      /* skip */
    }

    VDMSeq seqCompResult_2 = SeqUtil.seq();
    //@ assert (V2J.isSeq(seqCompResult_2) && (\forall int i; 0 <= i && i <
        V2J.size(seqCompResult_2); Utils.is_nat1(V2J.get(seqCompResult_2,i))
        ));

    VDMSet set_2 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_2) && (\forall int i; 0 <= i && i < V2J.size(
        set_2); Utils.is_nat1(V2J.get(set_2,i))));

    for (Iterator iterator_2 = set_2.iterator(); iterator_2.hasNext(); ) {
      Number x = ((Number) iterator_2.next());
      //@ assert Utils.is_nat1(x);

      seqCompResult_2.add(x);
    }
    {
      final VDMSeq ignorePattern_2 = Utils.copy(seqCompResult_2);
      //@ assert (V2J.isSeq1(ignorePattern_2) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(ignorePattern_2
          ,i))));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣violations");
    VDMSeq seqCompResult_3 = SeqUtil.seq();
    //@ assert (V2J.isSeq(seqCompResult_3) && (\forall int i; 0 <= i && i <
        V2J.size(seqCompResult_3); Utils.is_nat1(V2J.get(seqCompResult_3,i))
        ));

    VDMSet set_3 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_3) && (\forall int i; 0 <= i && i < V2J.size(
        set_3); Utils.is_nat1(V2J.get(set_3,i))));

    for (Iterator iterator_3 = set_3.iterator(); iterator_3.hasNext(); ) {
      Number x = ((Number) iterator_3.next());
      //@ assert Utils.is_nat1(x);

      if (x.longValue() > 4L) {
        seqCompResult_3.add(x);
      }
    }
    {
      final VDMSeq ignorePattern_3 = Utils.copy(seqCompResult_3);
      //@ assert (V2J.isSeq1(ignorePattern_3) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_3); Utils.is_nat(V2J.get(ignorePattern_3
          ,i))));

      /* skip */
    }

    VDMSeq seqCompResult_4 = SeqUtil.seq();
    //@ assert (V2J.isSeq(seqCompResult_4) && (\forall int i; 0 <= i && i <
        V2J.size(seqCompResult_4); Utils.is_nat(V2J.get(seqCompResult_4,i)))
        );

    VDMSet set_4 = xs();
    //@ assert (V2J.isSet(set_4) && (\forall int i; 0 <= i && i < V2J.size(
        set_4); Utils.is_nat(V2J.get(set_4,i))));

    for (Iterator iterator_4 = set_4.iterator(); iterator_4.hasNext(); ) {
      Number x = ((Number) iterator_4.next());
      //@ assert Utils.is_nat(x);

      seqCompResult_4.add(x);
    }
    {
      final VDMSeq ignorePattern_4 = Utils.copy(seqCompResult_4);
      //@ assert (V2J.isSeq1(ignorePattern_4) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_4); Utils.is_nat(V2J.get(ignorePattern_4
          ,i))));

      /* skip */
    }

    IO.println("After violations");
    return 0L;
  }
  /*@ pure @*/

  public static VDMSet xs() {

    VDMSet ret_1 = SetUtil.set();
    //@ assert (V2J.isSet(ret_1) && (\forall int i; 0 <= i && i < V2J.size(
        ret_1); Utils.is_nat(V2J.get(ret_1,i))));

    return Utils.copy(ret_1);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.2.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before violations"
Entry.java:76: JML assertion is false
      //@ assert (V2J.isSeq1(ignorePattern_3) && (\forall int i; 0 <= i && i <
           V2J.size(ignorePattern_3); Utils.is_nat(V2J.get(ignorePattern_3,i)))
          );
          ^
Entry.java:95: JML assertion is false
      //@ assert (V2J.isSeq1(ignorePattern_4) && (\forall int i; 0 <= i && i <
           V2J.size(ignorePattern_4); Utils.is_nat(V2J.get(ignorePattern_4,i)))
          );
          ^
"After violations"
```

C.3 Set.vdmsl

C.3.1 The VDM-SL model

```
module Entry
```

```

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣legal␣use");
  let - : set of nat1 = {x| x in set {1,2,3} & x > 0} in skip;
  let - : set of nat1 = {x| x in set {1,2,3}} in skip;
  IO`println("After␣legal␣use");
  IO`println("Before␣violations");
  let - : set of nat1 = {x| x in set {0,1,2} & x > -1} in skip;
  let - : set of nat1 = {x| x in set {0,1,2}} in skip;
  IO`println("After␣violations");
  return 0;
);

end Entry
```

C.3.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    VDMSet setCompResult_1 = SetUtil.set();
    //@ assert (V2J.isSet(setCompResult_1) && (\forall int i; 0 <= i && i <
        V2J.size(setCompResult_1); Utils.is_nat1(V2J.get(setCompResult_1,i))
        ));

    VDMSet set_1 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_nat1(V2J.get(set_1,i))));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext(); ) {
      Number x = ((Number) iterator_1.next());
      //@ assert Utils.is_nat1(x);

      if (x.longValue() > 0L) {
        setCompResult_1.add(x);
      }
    }
    {
      final VDMSet ignorePattern_1 = Utils.copy(setCompResult_1);
      //@ assert (V2J.isSet(ignorePattern_1) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_1); Utils.is_nat1(V2J.get(ignorePattern_1
          ,i))));

      /* skip */
    }

    VDMSet setCompResult_2 = SetUtil.set();
    //@ assert (V2J.isSet(setCompResult_2) && (\forall int i; 0 <= i && i <
        V2J.size(setCompResult_2); Utils.is_nat1(V2J.get(setCompResult_2,i))
        ));

    VDMSet set_2 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_2) && (\forall int i; 0 <= i && i < V2J.size(
        set_2); Utils.is_nat1(V2J.get(set_2,i))));

    for (Iterator iterator_2 = set_2.iterator(); iterator_2.hasNext(); ) {
      Number x = ((Number) iterator_2.next());
      //@ assert Utils.is_nat1(x);

      setCompResult_2.add(x);
    }
    {
      final VDMSet ignorePattern_2 = Utils.copy(setCompResult_2);
      //@ assert (V2J.isSet(ignorePattern_2) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_2); Utils.is_nat1(V2J.get(ignorePattern_2
          ,i))));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣violations");
    VDMSet setCompResult_3 = SetUtil.set();
    //@ assert (V2J.isSet(setCompResult_3) && (\forall int i; 0 <= i && i <
        V2J.size(setCompResult_3); Utils.is_nat(V2J.get(setCompResult_3,i)))
        );

    VDMSet set_3 = SetUtil.set(0L, 1L, 2L);
    //@ assert (V2J.isSet(set_3) && (\forall int i; 0 <= i && i < V2J.size(
        set_3); Utils.is_nat(V2J.get(set_3,i))));

    for (Iterator iterator_3 = set_3.iterator(); iterator_3.hasNext(); ) {
      Number x = ((Number) iterator_3.next());
      //@ assert Utils.is_nat(x);

      if (x.longValue() > -1L) {
        setCompResult_3.add(x);
      }
    }
    {
      final VDMSet ignorePattern_3 = Utils.copy(setCompResult_3);
      //@ assert (V2J.isSet(ignorePattern_3) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_3); Utils.is_nat1(V2J.get(ignorePattern_3
          ,i))));

      /* skip */
```

```
79      }
80
81      VDMSet setCompResult_4 = SetUtil.set();
82      //@ assert (V2J.isSet(setCompResult_4) && (\forall int i; 0 <= i && i <
            V2J.size(setCompResult_4); Utils.is_nat(V2J.get(setCompResult_4,i)))
            );
83
84      VDMSet set_4 = SetUtil.set(0L, 1L, 2L);
85      //@ assert (V2J.isSet(set_4) && (\forall int i; 0 <= i && i < V2J.size(
            set_4); Utils.is_nat(V2J.get(set_4,i))));
86
87      for (Iterator iterator_4 = set_4.iterator(); iterator_4.hasNext(); ) {
88        Number x = ((Number) iterator_4.next());
89        //@ assert Utils.is_nat(x);
90
91        setCompResult_4.add(x);
92      }
93      {
94        final VDMSet ignorePattern_4 = Utils.copy(setCompResult_4);
95        //@ assert (V2J.isSet(ignorePattern_4) && (\forall int i; 0 <= i && i
              < V2J.size(ignorePattern_4); Utils.is_nat1(V2J.get(ignorePattern_4
              ,i))));
96
97        /* skip */
98      }
99
100     IO.println("After␣violations");
101     return 0L;
102   }
103
104   public String toString() {
105
106     return "Entry{}";
107   }
108 }
```

C.3.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before violations"
Entry.java:76: JML assertion is false
      //@ assert (V2J.isSet(ignorePattern_3) && (\forall int i; 0 <= i && i <
          V2J.size(ignorePattern_3); Utils.is_nat1(V2J.get(ignorePattern_3,i)))
          );
          ^
Entry.java:95: JML assertion is false
      //@ assert (V2J.isSet(ignorePattern_4) && (\forall int i; 0 <= i && i <
          V2J.size(ignorePattern_4); Utils.is_nat1(V2J.get(ignorePattern_4,i)))
          );
          ^
"After violations"
```

C.4 AtomicStateInvViolation.vdmsl

C.4.1 The VDM-SL model

```
module Entry

imports from IO all
definitions

state St of
  x : nat
  init s == s = mk_St(1)
  inv s == s.x = 1
end

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣first␣atomic␣(expecting␣violation␣after␣atomic)");
  atomic
  (
    x := 2;
  );
  IO`println("After␣first␣atomic␣(expected␣violation␣before␣this␣print␣
      statement)");
  IO`println("Before␣second␣atomic");
  atomic
  (
    x := 1;
  );
  IO`println("After␣second␣atomic");
  return 2;
);

end Entry
```

C.4.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class St implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);

  public St(final Number _x) {

    //@ assert Utils.is_nat(_x);

    x = _x;
    //@ assert Utils.is_nat(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.St)) {
      return false;
    }

    project.Entrytypes.St other = ((project.Entrytypes.St) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.St copy() {

    return new project.Entrytypes.St(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`St" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_1 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_1));

    return ret_1;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_St(final Number _x) {

    return Utils.equals(_x, 1L);
  }
}
```

C.4.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ spec_public @*/

  private static project.Entrytypes.St St = new project.Entrytypes.St(1L);
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣first␣atomic␣(expecting␣violation␣after␣atomic)");
    Number atomicTmp_1 = 2L;
    //@ assert Utils.is_nat(atomicTmp_1);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      //@ assert St != null;

      St.set_x(atomicTmp_1);

      //@ set invChecksOn = true;

      //@ assert St.valid();

    } /* End of atomic statement */

    IO.println("After␣first␣atomic␣(expected␣violation␣before␣this␣print␣
       statement)");
    IO.println("Before␣second␣atomic");
    Number atomicTmp_2 = 1L;
    //@ assert Utils.is_nat(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      //@ assert St != null;

      St.set_x(atomicTmp_2);

      //@ set invChecksOn = true;

      //@ assert St.valid();

    } /* End of atomic statement */

    IO.println("After␣second␣atomic");
    return 2L;
  }

  public String toString() {

    return "Entry{" + "St␣:=␣" + Utils.toString(St) + "}";
  }
}
```

C.4.4 The OpenJML runtime assertion checker output

```
"Before first atomic (expecting violation after atomic)"
St.java:72: JML invariant is false on leaving method project.Entrytypes.St.
    valid()
  public Boolean valid() {
                 ^
St.java:12: Associated declaration
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);
                      ^
"After first atomic (expected violation before this print statement)"
"Before second atomic"
"After second atomic"
```

C.5 AtomicStateInvNoViolation.vdmsl

C.5.1 The VDM-SL model

```
module Entry

imports from IO all
definitions

state St of
  x : nat
  init s == s = mk_St(1)
  inv s == s.x = 1
end

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣atomic");
  atomic
  (
    x := 2;
    x := 1;
  );
  IO`println("After␣atomic");

  return x;
);

end Entry
```

C.5.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class St implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);

  public St(final Number _x) {

    //@ assert Utils.is_nat(_x);

    x = _x;
    //@ assert Utils.is_nat(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.St)) {
      return false;
    }

    project.Entrytypes.St other = ((project.Entrytypes.St) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.St copy() {

    return new project.Entrytypes.St(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`St" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_1 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_1));

    return ret_1;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_St(final Number _x) {

    return Utils.equals(_x, 1L);
  }
}
```

C.5.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ spec_public @*/

  private static project.Entrytypes.St St = new project.Entrytypes.St(1L);
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣atomic");
    Number atomicTmp_1 = 2L;
    //@ assert Utils.is_nat(atomicTmp_1);

    Number atomicTmp_2 = 1L;
    //@ assert Utils.is_nat(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      //@ assert St != null;

      St.set_x(atomicTmp_1);

      //@ assert St != null;

      St.set_x(atomicTmp_2);

      //@ set invChecksOn = true;

      //@ assert St.valid();

    } /* End of atomic statement */

    IO.println("After␣atomic");
    return St.get_x();
  }

  public String toString() {

    return "Entry{" + "St␣:=␣" + Utils.toString(St) + "}";
  }
}
```

C.5.4 The OpenJML runtime assertion checker output

```
"Before atomic"
"After atomic"
```

C.6 InvChecksOnFlagInOtherModule.vdmsl

C.6.1 The VDM-SL model

```
module Mod

exports all
definitions
types

M :: x : int
inv m == m.x > 0;

operations
```

```

op : () ==> ()
op () ==
(
  dcl m : M := mk_M(1);
  atomic
  (
    m.x := -20;
    m.x := 20;
  );
);

end Mod

module Entry

exports all
imports from IO all
definitions
types

E :: x : int
inv e == e.x > 0;

operations

Run : () ==> ()
Run () ==
(
  dcl e : E := mk_E(1);
  atomic
  (
    e.x := -20;
    e.x := 20;
  );
  IO`println("Done!␣Expected␣to␣exit␣without␣any␣errors");
);

end Entry
```

C.6.2 The generated Java/JML

```
package project.Modtypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class M implements Record {
  public Number x;
  //@ public instance invariant project.Mod.invChecksOn ==> inv_M(x);

  public M(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Modtypes.M)) {
      return false;
    }

    project.Modtypes.M other = ((project.Modtypes.M) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Modtypes.M copy() {

    return new project.Modtypes.M(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Mod`M" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_1 = x;
    //@ assert project.Mod.invChecksOn ==> (Utils.is_int(ret_1));

    return ret_1;
  }

  public void set_x(final Number _x) {

    //@ assert project.Mod.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Mod.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_M(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.6.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Mod {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Mod() {}

  public static void op() {

    project.Modtypes.M m = new project.Modtypes.M(1L);
    //@ assert Utils.is_(m,project.Modtypes.M.class);

    Number atomicTmp_1 = -20L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 20L;
    //@ assert Utils.is_int(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      //@ assert m != null;

      m.set_x(atomicTmp_1);

      //@ assert m != null;

      m.set_x(atomicTmp_2);

      //@ set invChecksOn = true;

      //@ assert m.valid();

    } /* End of atomic statement */
  }

  public String toString() {

    return "Mod{}";
  }
}
```

C.6.4 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class E implements Record {
  public Number x;
  //@ public instance invariant project.Mod.invChecksOn ==> inv_E(x);

  public E(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.E)) {
      return false;
    }

    project.Entrytypes.E other = ((project.Entrytypes.E) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.E copy() {

    return new project.Entrytypes.E(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`E" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Mod.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Mod.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Mod.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_E(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.6.5 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  private Entry() {}

  public static void Run() {

    project.Entrytypes.E e = new project.Entrytypes.E(1L);
    //@ assert Utils.is_(e,project.Entrytypes.E.class);

    Number atomicTmp_3 = -20L;
    //@ assert Utils.is_int(atomicTmp_3);

    Number atomicTmp_4 = 20L;
    //@ assert Utils.is_int(atomicTmp_4);

    {
        /* Start of atomic statement */
      //@ set project.Mod.invChecksOn = false;

      //@ assert e != null;

      e.set_x(atomicTmp_3);

      //@ assert e != null;

      e.set_x(atomicTmp_4);

      //@ set project.Mod.invChecksOn = true;

      //@ assert e.valid();

    } /* End of atomic statement */

    IO.println("Done!␣Expected␣to␣exit␣without␣any␣errors");
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.6.6 The OpenJML runtime assertion checker output

```
"Done! Expected to exit without any errors"
```

C.7 AtomicRecUnion.vdmsl

C.7.1 The VDM-SL model

```
module Entry

imports from IO all
exports all

definitions

types
R1 :: x : int
inv r1 == r1.x > 0;

R2 :: x : int
inv r2 == r2.x > 0;

operations

Run : () ==> ?
Run () ==
(dcl r : R1 | R2 := mk_R1(1);

 IO`println("Before␣valid␣use");
 atomic
 (
   r.x := -5;
   r.x := 5;
 );
 IO`println("After␣valid␣use");

 IO`println("Before␣illegal␣use");
 atomic
 (
   r.x := -5;
 );
 IO`println("After␣illegal␣use");

 return 0;
)

end Entry
```

C.7.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(x);

  public R1(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R1)) {
      return false;
    }

    project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R1 copy() {

    return new project.Entrytypes.R1(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R1" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_1 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_1));

    return ret_1;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R1(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.7.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public Number x;
```

```
//@ public instance invariant project.Entry.invChecksOn ==> inv_R2(x);

public R2(final Number _x) {

  //@ assert Utils.is_int(_x);

  x = _x;
  //@ assert Utils.is_int(x);

}
/*@ pure @*/

public boolean equals(final Object obj) {

  if (!(obj instanceof project.Entrytypes.R2)) {
    return false;
  }

  project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

  return Utils.equals(x, other.x);
}
/*@ pure @*/

public int hashCode() {

  return Utils.hashCode(x);
}
/*@ pure @*/

public project.Entrytypes.R2 copy() {

  return new project.Entrytypes.R2(x);
}
/*@ pure @*/

public String toString() {

  return "mk_Entry`R2" + Utils.formatFields(x);
}
/*@ pure @*/

public Number get_x() {

  Number ret_2 = x;
  //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

  return ret_2;
}

public void set_x(final Number _x) {

  //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

  x = _x;
  //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

}
/*@ pure @*/

```

```
  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R2(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.7.4 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Object r = new project.Entrytypes.R1(1L);
    //@ assert (Utils.is_(r,project.Entrytypes.R1.class) || Utils.is_(r,
        project.Entrytypes.R2.class));

    IO.println("Before␣valid␣use");
    Number atomicTmp_1 = -5L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 5L;
    //@ assert Utils.is_int(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      if (r instanceof project.Entrytypes.R1) {
        //@ assert r != null;

        ((project.Entrytypes.R1) r).set_x(atomicTmp_1);

      } else if (r instanceof project.Entrytypes.R2) {
        //@ assert r != null;

        ((project.Entrytypes.R2) r).set_x(atomicTmp_1);

      } else {
        throw new RuntimeException("Missing␣member:␣x");
      }

      if (r instanceof project.Entrytypes.R1) {
        //@ assert r != null;

        ((project.Entrytypes.R1) r).set_x(atomicTmp_2);

      } else if (r instanceof project.Entrytypes.R2) {
        //@ assert r != null;

        ((project.Entrytypes.R2) r).set_x(atomicTmp_2);

      } else {
        throw new RuntimeException("Missing␣member:␣x");
      }

      //@ set invChecksOn = true;

      //@ assert r instanceof project.Entrytypes.R1 ==> ((project.Entrytypes
          .R1) r).valid();

      //@ assert r instanceof project.Entrytypes.R2 ==> ((project.Entrytypes
          .R2) r).valid();

    } /* End of atomic statement */

    IO.println("After␣valid␣use");
    IO.println("Before␣illegal␣use");
    Number atomicTmp_3 = -5L;
    //@ assert Utils.is_int(atomicTmp_3);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      if (r instanceof project.Entrytypes.R1) {
        //@ assert r != null;

        ((project.Entrytypes.R1) r).set_x(atomicTmp_3);

      } else if (r instanceof project.Entrytypes.R2) {
        //@ assert r != null;

        ((project.Entrytypes.R2) r).set_x(atomicTmp_3);

      } else {
        throw new RuntimeException("Missing␣member:␣x");
      }

      //@ set invChecksOn = true;

      //@ assert r instanceof project.Entrytypes.R1 ==> ((project.Entrytypes
          .R1) r).valid();

      //@ assert r instanceof project.Entrytypes.R2 ==> ((project.Entrytypes
          .R2) r).valid();

    } /* End of atomic statement */

    IO.println("After␣illegal␣use");
```

```
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.7.5 The OpenJML runtime assertion checker output

```
"Before valid use"
"After valid use"
"Before illegal use"
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/AtomicRecUnion/
    project/Entrytypes/R1.java:72: JML invariant is false on leaving method
    project.Entrytypes.R1.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/AtomicRecUnion/
    project/Entrytypes/R1.java:12: Associated declaration: /home/peter/git-
    repos/ovt/core/codegen/vdm2jml/target/jml/code/AtomicRecUnion/project/
    Entrytypes/R1.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(x);
                      ^
"After illegal use"
```

C.8 NamedTypeInvValues.vdmsl

C.8.1 The VDM-SL model

```
module Entry

exports all
definitions

values

fOk : CN = 'a';
fBreak : CN = 'b';

types

CN = C|N
inv cn == is_char(cn) => cn = 'a';
N = nat;
C = char;

operations

Run : () ==> ?
Run () ==
  return 0;

end Entry
```

C.8.2 The generated Java/JML

```java
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  //@ public static invariant ((fOk == null) || ((Utils.is_char(fOk) &&
      inv_Entry_C(fOk)) || (Utils.is_nat(fOk) && inv_Entry_N(fOk))) &&
      inv_Entry_CN(fOk));

  public static final Object fOk = 'a';
  //@ public static invariant ((fBreak == null) || ((Utils.is_char(fBreak)
      && inv_Entry_C(fBreak)) || (Utils.is_nat(fBreak) && inv_Entry_N(fBreak
      ))) && inv_Entry_CN(fBreak));

  public static final Object fBreak = 'b';
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    return 0L;
  }

  public String toString() {

    return "Entry{" + "fOk␣=␣" + Utils.toString(fOk) + ",␣fBreak␣=␣" + Utils
        .toString(fBreak) + "}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_CN(final Object check_cn) {

    Object cn = ((Object) check_cn);

    Boolean orResult_1 = false;

    if (!(Utils.is_char(cn))) {
      orResult_1 = true;
    } else {
      orResult_1 = Utils.equals(cn, 'a');
    }

    return orResult_1;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_N(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_C(final Object check_elem) {

    return true;
  }
}
```

C.8.3 The OpenJML runtime assertion checker output

```
Entry.java:10: JML static initialization may not be correct
final public class Entry {
             ^
Entry.java:14: Associated declaration
  //@ public static invariant ((fBreak == null) || ((Utils.is_char(fBreak) &&
      inv_Entry_C(fBreak)) || (Utils.is_nat(fBreak) && inv_Entry_N(fBreak))) &&
       inv_Entry_CN(fBreak));
                    ^
Main.java:7: JML invariant is false on entering method project.Entry.Run()
    from Main.main(java.lang.String[])
                        project.Entry.Run();
                                         ^
Entry.java:14: Associated declaration
  //@ public static invariant ((fBreak == null) || ((Utils.is_char(fBreak) &&
      inv_Entry_C(fBreak)) || (Utils.is_nat(fBreak) && inv_Entry_N(fBreak))) &&
       inv_Entry_CN(fBreak));
                    ^
Entry.java:21: JML invariant is false on leaving method project.Entry.Run()
  public static Object Run() {
                       ^
Entry.java:14: Associated declaration
  //@ public static invariant ((fBreak == null) || ((Utils.is_char(fBreak) &&
      inv_Entry_C(fBreak)) || (Utils.is_nat(fBreak) && inv_Entry_N(fBreak))) &&
       inv_Entry_CN(fBreak));
                    ^
```

C.9 NamedTypeInvMapUpdate.vdmsl

C.9.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

M = map ? to ?
inv m == forall x in set dom m & (is_nat(x) and is_nat(m(x))) => x + 1 = m(x
    )
```

```

operations

Run : () ==> ?
Run () ==
(
  dcl m : M := {'a' |-> 1, 1 |-> 2};
  m('a') := 2;
  m(1) := 2;
  IO`println("Breaking␣named␣type␣invariant␣for␣sequence");
  m(2) := 10;
  return 0;
);

end Entry
```

C.9.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    VDMMap m = MapUtil.map(new Maplet('a', 1L), new Maplet(1L, 2L));
    //@ assert ((V2J.isMap(m) && (\forall int i; 0 <= i && i < V2J.size(m);
        true && true)) && inv_Entry_M(m));

    //@ assert m != null;

    Utils.mapSeqUpdate(m, 'a', 2L);
    //@ assert ((V2J.isMap(m) && (\forall int i; 0 <= i && i < V2J.size(m);
        true && true)) && inv_Entry_M(m));

    //@ assert m != null;

    Utils.mapSeqUpdate(m, 1L, 2L);
    //@ assert ((V2J.isMap(m) && (\forall int i; 0 <= i && i < V2J.size(m);
        true && true)) && inv_Entry_M(m));

    IO.println("Breaking␣named␣type␣invariant␣for␣sequence");
    //@ assert m != null;

    Utils.mapSeqUpdate(m, 2L, 10L);
    //@ assert ((V2J.isMap(m) && (\forall int i; 0 <= i && i < V2J.size(m);
        true && true)) && inv_Entry_M(m));

    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_M(final Object check_m) {

    VDMMap m = ((VDMMap) check_m);

    Boolean forAllExpResult_1 = true;
    VDMSet set_1 = MapUtil.dom(Utils.copy(m));
    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() &&
        forAllExpResult_1; ) {
      Object x = ((Object) iterator_1.next());
      Boolean orResult_1 = false;

      Boolean andResult_1 = false;

      if (Utils.is_nat(x)) {
        if (Utils.is_nat(Utils.get(m, x))) {
          andResult_1 = true;
        }
      }

      if (!(andResult_1)) {
        orResult_1 = true;
      } else {
        orResult_1 = Utils.equals(((Number) x).doubleValue() + 1L, Utils.get
            (m, x));
      }

      forAllExpResult_1 = orResult_1;
    }
    return forAllExpResult_1;
  }
}
```

C.9.3 The OpenJML runtime assertion checker output

```
"Breaking named type invariant for sequence"
Entry.java:34: JML assertion is false
    //@ assert ((V2J.isMap(m) && (\forall int i; 0 <= i && i < V2J.size(m);
        true && true)) && inv_Entry_M(m));
        ^
```

C.10 NamedTypeInvSeqUpdate.vdmsl

C.10.1 The VDM-SL model

```
module Entry
```

```

exports all
imports from IO all
definitions

types

S = seq of ?
inv s == forall x in set elems s & is_nat(x) => x > 5;

operations

Run : () ==> ?
Run () ==
(
  dcl s : S := [10,11,12];
  s(1) := 'a';
  s(2) := nil;
  IO`println("Breaking␣named␣type␣invariant␣for␣sequence");
  s(3) := 4;
  return 0;
);

end Entry
```

C.10.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    VDMSeq s = SeqUtil.seq(10L, 11L, 12L);
    //@ assert ((V2J.isSeq(s) && (\forall int i; 0 <= i && i < V2J.size(s);
        true)) && inv_Entry_S(s));

    //@ assert s != null;

    Utils.mapSeqUpdate(s, 1L, 'a');
    //@ assert ((V2J.isSeq(s) && (\forall int i; 0 <= i && i < V2J.size(s);
        true)) && inv_Entry_S(s));

    //@ assert s != null;

    Utils.mapSeqUpdate(s, 2L, null);
    //@ assert ((V2J.isSeq(s) && (\forall int i; 0 <= i && i < V2J.size(s);
        true)) && inv_Entry_S(s));
29
30      IO.println("Breaking␣named␣type␣invariant␣for␣sequence");
31      //@ assert s != null;
32
33      Utils.mapSeqUpdate(s, 3L, 4L);
34      //@ assert ((V2J.isSeq(s) && (\forall int i; 0 <= i && i < V2J.size(s);
            true)) && inv_Entry_S(s));
35
36      return 0L;
37    }
38
39    public String toString() {
40
41      return "Entry{}";
42    }
43
44    /*@ pure @*/
45    /*@ helper @*/
46
47    public static Boolean inv_Entry_S(final Object check_s) {
48
49      VDMSeq s = ((VDMSeq) check_s);
50
51      Boolean forAllExpResult_1 = true;
52      VDMSet set_1 = SeqUtil.elems(Utils.copy(s));
53      for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() &&
            forAllExpResult_1; ) {
54        Object x = ((Object) iterator_1.next());
55        Boolean orResult_1 = false;
56
57        if (!(Utils.is_nat(x))) {
58          orResult_1 = true;
59        } else {
60          orResult_1 = ((Number) x).doubleValue() > 5L;
61        }
62
63        forAllExpResult_1 = orResult_1;
64      }
65      return forAllExpResult_1;
66    }
67  }
```

C.10.3 The OpenJML runtime assertion checker output

```
"Breaking named type invariant for sequence"
Entry.java:34: JML assertion is false
    //@ assert ((V2J.isSeq(s) && (\forall int i; 0 <= i && i < V2J.size(s);
        true)) && inv_Entry_S(s));
        ^
```

C.11 RecursionConservativeChecking.vdmsl

C.11.1 The VDM-SL model

```
module Entry
```

```

exports all
imports from IO all
definitions
types

T = nat | seq of T;

T1 = nat | nat * T1;

operations

tNat : () ==> T
tNat () == return 1;

tSeq : () ==> T
tSeq () == return [[[1]]];

t1Nat : () ==> T1
t1Nat () == return 1;

t1Tup : () ==> T1
t1Tup () == return mk_(1,2);

Run : () ==> ?
Run () ==
(
  IO`println("Before␣legal␣use");
  let - = tNat() in skip;
  let - = t1Nat() in skip;
  IO`println("Before␣legal␣use");
  IO`println("Before␣illegal␣use");
  let - = tSeq() in skip;
  let - = t1Tup() in skip;
  IO`println("After␣illegal␣use");
  return 0;
);

end Entry
```

C.11.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object tNat() {

    Object ret_1 = 1L;
    //@ assert ((Utils.is_nat(ret_1) || (V2J.isSeq(ret_1) && (\forall int i;
         0 <= i && i < V2J.size(ret_1); false))) && inv_Entry_T(ret_1));

    return Utils.copy(ret_1);
  }

  public static Object tSeq() {

    Object ret_2 = SeqUtil.seq(SeqUtil.seq(SeqUtil.seq(1L)));
    //@ assert ((Utils.is_nat(ret_2) || (V2J.isSeq(ret_2) && (\forall int i;
         0 <= i && i < V2J.size(ret_2); false))) && inv_Entry_T(ret_2));

    return Utils.copy(ret_2);
  }

  public static Object t1Nat() {

    Object ret_3 = 1L;
    //@ assert (((V2J.isTup(ret_3,2) && Utils.is_nat(V2J.field(ret_3,0)) &&
        false) || Utils.is_nat(ret_3)) && inv_Entry_T1(ret_3));

    return Utils.copy(ret_3);
  }

  public static Object t1Tup() {

    Object ret_4 = Tuple.mk_(1L, 2L);
    //@ assert (((V2J.isTup(ret_4,2) && Utils.is_nat(V2J.field(ret_4,0)) &&
        false) || Utils.is_nat(ret_4)) && inv_Entry_T1(ret_4));

    return Utils.copy(ret_4);
  }

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final Object ignorePattern_1 = tNat();
      //@ assert ((Utils.is_nat(ignorePattern_1) || (V2J.isSeq(
          ignorePattern_1) && (\forall int i; 0 <= i && i < V2J.size(
          ignorePattern_1); false))) && inv_Entry_T(ignorePattern_1));

      /* skip */
    }


    {
      final Object ignorePattern_2 = t1Nat();
      //@ assert (((V2J.isTup(ignorePattern_2,2) && Utils.is_nat(V2J.field(
          ignorePattern_2,0)) && false) || Utils.is_nat(ignorePattern_2)) &&
           inv_Entry_T1(ignorePattern_2));

      /* skip */
    }

    IO.println("Before␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final Object ignorePattern_3 = tSeq();
      //@ assert ((Utils.is_nat(ignorePattern_3) || (V2J.isSeq(
          ignorePattern_3) && (\forall int i; 0 <= i && i < V2J.size(
          ignorePattern_3); false))) && inv_Entry_T(ignorePattern_3));

      /* skip */
    }

    {
      final Object ignorePattern_4 = t1Tup();
      //@ assert (((V2J.isTup(ignorePattern_4,2) && Utils.is_nat(V2J.field(
          ignorePattern_4,0)) && false) || Utils.is_nat(ignorePattern_4)) &&
           inv_Entry_T1(ignorePattern_4));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T1(final Object check_elem) {

    return true;
  }
}
```

C.11.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"Before legal use"
"Before illegal use"
Entry.java:26: JML assertion is false
    //@ assert ((Utils.is_nat(ret_2) || (V2J.isSeq(ret_2) && (\forall int i; 0
        <= i && i < V2J.size(ret_2); false))) && inv_Entry_T(ret_2));
        ^
Entry.java:68: JML assertion is false
      //@ assert ((Utils.is_nat(ignorePattern_3) || (V2J.isSeq(ignorePattern_3
          ) && (\forall int i; 0 <= i && i < V2J.size(ignorePattern_3); false))
          ) && inv_Entry_T(ignorePattern_3));
          ^
Entry.java:42: JML assertion is false
```

```
    //@ assert (((V2J.isTup(ret_4,2) && Utils.is_nat(V2J.field(ret_4,0)) &&
        false) || Utils.is_nat(ret_4)) && inv_Entry_T1(ret_4));
        ^
Entry.java:75: JML assertion is false
      //@ assert (((V2J.isTup(ignorePattern_4,2) && Utils.is_nat(V2J.field(
          ignorePattern_4,0)) && false) || Utils.is_nat(ignorePattern_4)) &&
          inv_Entry_T1(ignorePattern_4));
          ^
"After illegal use"
```

C.12 NamedTypeInvLocalDecls.vdmsl

C.12.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

No = Even | Large;

Even = nat
inv ev == ev mod 2 = 0;

Large = real
inv la == la > 1000;

operations

typeUseOk : () ==> ()
typeUseOk () ==
let  - = 1,
     even : No = 2,
     - = 3
in
  skip;

typeUseNotOk : () ==> ()
typeUseNotOk () ==
(
  IO`println("Before␣breaking␣named␣type␣invariant");
  (
    dcl notLarge : No := 999;
    IO`println("After␣breaking␣named␣type␣invariant");
    skip;
  );
);

Run : () ==> ?
Run () ==
(
  typeUseOk();
  typeUseNotOk();
  return 0;
);

end Entry
```

C.12.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static void typeUseOk() {

    final Number ignorePattern_1 = 1L;
    //@ assert Utils.is_nat1(ignorePattern_1);

    final Object even = 2L;
    //@ assert (((Utils.is_nat(even) && inv_Entry_Even(even)) || (Utils.
        is_real(even) && inv_Entry_Large(even))) && inv_Entry_No(even));

    final Number ignorePattern_2 = 3L;
    //@ assert Utils.is_nat1(ignorePattern_2);

    /* skip */

  }

  public static void typeUseNotOk() {

    IO.println("Before␣breaking␣named␣type␣invariant");
    {
      Object notLarge = 999L;
      //@ assert (((Utils.is_nat(notLarge) && inv_Entry_Even(notLarge)) || (
          Utils.is_real(notLarge) && inv_Entry_Large(notLarge))) &&
          inv_Entry_No(notLarge));

      IO.println("After␣breaking␣named␣type␣invariant");
      /* skip */
    }
  }

  public static Object Run() {

    typeUseOk();
    typeUseNotOk();
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_No(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Even(final Object check_ev) {

    Number ev = ((Number) check_ev);

    return Utils.equals(Utils.mod(ev.longValue(), 2L), 0L);
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Large(final Object check_la) {

    Number la = ((Number) check_la);

    return la.doubleValue() > 1000L;
  }
}
```

C.12.3 The OpenJML runtime assertion checker output

```
"Before breaking named type invariant"
Entry.java:35: JML assertion is false
      //@ assert (((Utils.is_nat(notLarge) && inv_Entry_Even(notLarge)) || (
          Utils.is_real(notLarge) && inv_Entry_Large(notLarge))) &&
          inv_Entry_No(notLarge));
          ^
"After breaking named type invariant"
```

C.13 NamedTypeInvReturn.vdmsl

C.13.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

```

```
A = [B | C]
inv c == is_char(c) => c = 'a';
B = real;
C = char
inv c == c = 'a' or c = 'b';

operations

Run : () ==> ?
Run () ==
let - = idC('b'),
    - = idC('a'),
    - = idA(nil),
    - = idA(2.1),
    - = constFunc()
in
(
  IO`println("Breaking␣named␣type␣invariant␣for␣return␣value");
  let - = idA('b') in skip;
  return 0;
);

functions

idC : C -> C
idC (c) ==
  c;

idA : A -> A
idA (a) ==
  a;

constFunc : () -> A
constFunc () ==
  'a';

end Entry
```

C.13.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    final Character ignorePattern_1 = idC('b');
    //@ assert (Utils.is_char(ignorePattern_1) && inv_Entry_C(
        ignorePattern_1));

    final Character ignorePattern_2 = idC('a');
    //@ assert (Utils.is_char(ignorePattern_2) && inv_Entry_C(
        ignorePattern_2));

    final Object ignorePattern_3 = idA(null);
    //@ assert ((ignorePattern_3 == null) || ((ignorePattern_3 == null) || (
        Utils.is_real(ignorePattern_3) && inv_Entry_B(ignorePattern_3)) || (
        Utils.is_char(ignorePattern_3) && inv_Entry_C(ignorePattern_3))) &&
        inv_Entry_A(ignorePattern_3));

    final Object ignorePattern_4 = idA(2.1);
    //@ assert ((ignorePattern_4 == null) || ((ignorePattern_4 == null) || (
        Utils.is_real(ignorePattern_4) && inv_Entry_B(ignorePattern_4)) || (
        Utils.is_char(ignorePattern_4) && inv_Entry_C(ignorePattern_4))) &&
        inv_Entry_A(ignorePattern_4));

    final Object ignorePattern_5 = constFunc();
    //@ assert ((ignorePattern_5 == null) || ((ignorePattern_5 == null) || (
        Utils.is_real(ignorePattern_5) && inv_Entry_B(ignorePattern_5)) || (
        Utils.is_char(ignorePattern_5) && inv_Entry_C(ignorePattern_5))) &&
        inv_Entry_A(ignorePattern_5));

    {
      IO.println("Breaking␣named␣type␣invariant␣for␣return␣value");
      {
        final Object ignorePattern_6 = idA('b');
        //@ assert ((ignorePattern_6 == null) || ((ignorePattern_6 == null)
            || (Utils.is_real(ignorePattern_6) && inv_Entry_B(
            ignorePattern_6)) || (Utils.is_char(ignorePattern_6) &&
            inv_Entry_C(ignorePattern_6))) && inv_Entry_A(ignorePattern_6));

        /* skip */
      }

      return 0L;
    }
  }
  /*@ pure @*/

  public static Character idC(final Character c) {

    //@ assert (Utils.is_char(c) && inv_Entry_C(c));

    Character ret_1 = c;
    //@ assert (Utils.is_char(ret_1) && inv_Entry_C(ret_1));

    return ret_1;
  }
  /*@ pure @*/

  public static Object idA(final Object a) {

    //@ assert ((a == null) || ((a == null) || (Utils.is_real(a) &&
        inv_Entry_B(a)) || (Utils.is_char(a) && inv_Entry_C(a))) &&
        inv_Entry_A(a));

    Object ret_2 = a;
    //@ assert ((ret_2 == null) || ((ret_2 == null) || (Utils.is_real(ret_2)
         && inv_Entry_B(ret_2)) || (Utils.is_char(ret_2) && inv_Entry_C(
        ret_2))) && inv_Entry_A(ret_2));

    return ret_2;
  }
  /*@ pure @*/

  public static Object constFunc() {

    Object ret_3 = 'a';
    //@ assert ((ret_3 == null) || ((ret_3 == null) || (Utils.is_real(ret_3)
         && inv_Entry_B(ret_3)) || (Utils.is_char(ret_3) && inv_Entry_C(
        ret_3))) && inv_Entry_A(ret_3));

    return ret_3;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_A(final Object check_c) {

    Object c = ((Object) check_c);

    Boolean orResult_1 = false;

    if (!(Utils.is_char(c))) {
      orResult_1 = true;
    } else {
      orResult_1 = Utils.equals(c, 'a');
    }

    return orResult_1;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_B(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_C(final Object check_c) {

    Character c = ((Character) check_c);

    Boolean orResult_2 = false;

    if (Utils.equals(c, 'a')) {
      orResult_2 = true;
    } else {
      orResult_2 = Utils.equals(c, 'b');
    }

    return orResult_2;
  }
}
```

C.13.3 The OpenJML runtime assertion checker output

```
"Breaking named type invariant for return value"
Entry.java:59: JML assertion is false
    //@ assert ((a == null) || ((a == null) || (Utils.is_real(a) &&
        inv_Entry_B(a)) || (Utils.is_char(a) && inv_Entry_C(a))) && inv_Entry_A
        (a));
        ^
Entry.java:62: JML assertion is false
    //@ assert ((ret_2 == null) || ((ret_2 == null) || (Utils.is_real(ret_2)
        && inv_Entry_B(ret_2)) || (Utils.is_char(ret_2) && inv_Entry_C(ret_2)))
         && inv_Entry_A(ret_2));
        ^
Entry.java:36: JML assertion is false
        //@ assert ((ignorePattern_6 == null) || ((ignorePattern_6 == null) ||
             (Utils.is_real(ignorePattern_6) && inv_Entry_B(ignorePattern_6))
            || (Utils.is_char(ignorePattern_6) && inv_Entry_C(ignorePattern_6))
            ) && inv_Entry_A(ignorePattern_6));
            ^
```

C.14 NamedTypeInvMethodParam.vdmsl

C.14.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

Even = nat
inv n == n mod 2 = 0;

operations

Run : () ==> ?
Run () ==
let n1 = 2,
    n2 = 3
in
(
  let - = op(n1, 5, n1) in skip;
  IO`println("Breaking␣named␣type␣invariant␣for␣method␣parameter");
  let - = op(n1, 6, n2) in skip;
  return 0;
);

op : Even * nat * Even ==> Even
op (a,b,c) ==
  return b * (a + c);

end Entry
```

C.14.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    final Number n1 = 2L;
    //@ assert Utils.is_nat1(n1);

    final Number n2 = 3L;
    //@ assert Utils.is_nat1(n2);

    {
      {
        final Number ignorePattern_1 = op(n1, 5L, n1);
        //@ assert (Utils.is_nat(ignorePattern_1) && inv_Entry_Even(
            ignorePattern_1));

        /* skip */
      }

      IO.println("Breaking␣named␣type␣invariant␣for␣method␣parameter");
      {
        final Number ignorePattern_2 = op(n1, 6L, n2);
        //@ assert (Utils.is_nat(ignorePattern_2) && inv_Entry_Even(
            ignorePattern_2));

        /* skip */
      }

      return 0L;
    }
  }

  public static Number op(final Number a, final Number b, final Number c) {

    //@ assert (Utils.is_nat(a) && inv_Entry_Even(a));

    //@ assert Utils.is_nat(b);

    //@ assert (Utils.is_nat(c) && inv_Entry_Even(c));

    Number ret_1 = b.longValue() * (a.longValue() + c.longValue());
    //@ assert (Utils.is_nat(ret_1) && inv_Entry_Even(ret_1));

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Even(final Object check_n) {

    Number n = ((Number) check_n);

    return Utils.equals(Utils.mod(n.longValue(), 2L), 0L);
  }
}
```

C.14.3 The OpenJML runtime assertion checker output

```
"Breaking named type invariant for method parameter"
Entry.java:49: JML assertion is false
    //@ assert (Utils.is_nat(c) && inv_Entry_Even(c));
        ^
```

C.15 NamedTypeInvNullAllowed.vdmsl

C.15.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

N = [X | Y];

X = nat;
Y = char;

operations

Run : () ==> ?
Run () ==
let e : N = nil
```

```
in
  return e;

end Entry
```

C.15.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    final Object e = null;
    //@ assert ((e == null) || ((e == null) || (Utils.is_nat(e) &&
        inv_Entry_X(e)) || (Utils.is_char(e) && inv_Entry_Y(e))) &&
        inv_Entry_N(e));

    return e;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_N(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_X(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Y(final Object check_elem) {

    return true;
  }
}
```

C.15.3 The OpenJML runtime assertion checker output

```
```

C.16 NamedTypeMadeOptional.vdmsl

C.16.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

Even = nat
inv e == e mod 2 = 0;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣valid␣use");
 (
  dcl e : [Even] := 2;
  e := nil;
 );
 IO`println("After␣valid␣use");

 IO`println("Before␣invalid␣use");
 (
  dcl e : Even := 2;
  e := Nil();
 );
 IO`println("After␣invalid␣use");
  return 0;
);

functions

Nil : () -> [Even]
Nil () == nil;

end Entry
```

C.16.2 The generated Java/JML

```
package project;

```

```
import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣valid␣use");
    {
      Number e = 2L;
      //@ assert ((e == null) || Utils.is_nat(e) && inv_Entry_Even(e));

      e = null;
      //@ assert ((e == null) || Utils.is_nat(e) && inv_Entry_Even(e));

    }

    IO.println("After␣valid␣use");
    IO.println("Before␣invalid␣use");
    {
      Number e = 2L;
      //@ assert (Utils.is_nat(e) && inv_Entry_Even(e));

      e = Nil();
      //@ assert (Utils.is_nat(e) && inv_Entry_Even(e));

    }

    IO.println("After␣invalid␣use");
    return 0L;
  }
  /*@ pure @*/

  public static Number Nil() {

    Number ret_1 = null;
    //@ assert ((ret_1 == null) || Utils.is_nat(ret_1) && inv_Entry_Even(
        ret_1));

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Even(final Object check_e) {

    Number e = ((Number) check_e);

    return Utils.equals(Utils.mod(e.longValue(), 2L), 0L);
  }
}
```

C.16.3 The OpenJML runtime assertion checker output

```
"Before valid use"
"After valid use"
"Before invalid use"
Entry.java:34: JML assertion is false
      //@ assert (Utils.is_nat(e) && inv_Entry_Even(e));
          ^
"After invalid use"
```

C.17 NamedTypeInvAsssignments.vdmsl

C.17.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

state St of
  x : PT
  init s == s = mk_St(1)
end

types

PT = PossiblyOne | True;
PossiblyOne = [nat]
inv p == p <> nil => p = 1;
True = bool
inv b == b;

operations

op1 : () ==> ()
op1 () ==
(
  dcl p : PT := nil;
  p := 1;
  p := true;
  St.x := nil;
  St.x := 1;
  St.x := true;
  IO`println("Breaking␣named␣type␣invariant␣(assigning␣record␣field)");
  St.x := false;
);

op2 : () ==> ()
op2 () ==
(
  dcl p1 : PT := nil;
  St.x := true;
  IO`println("Breaking␣named␣type␣invariant␣(assigning␣local␣variable)");
  p1 := false;
);

Run : () ==> ?
Run () ==
(
  op1();
  op2();
  return 0;
);

end Entry
```

C.17.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class St implements Record {
  public Object x;

  public St(final Object _x) {

    //@ assert ((((_x == null) || Utils.is_nat(_x)) &&
        inv_Entry_PossiblyOne(_x)) || (Utils.is_bool(_x) && inv_Entry_True(
        _x))) && inv_Entry_PT(_x));

    x = _x != null ? _x : null;
    //@ assert ((((x == null) || Utils.is_nat(x)) && inv_Entry_PossiblyOne(
        x)) || (Utils.is_bool(x) && inv_Entry_True(x))) && inv_Entry_PT(x));

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.St)) {
      return false;
    }

    project.Entrytypes.St other = ((project.Entrytypes.St) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.St copy() {

    return new project.Entrytypes.St(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`St" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Object get_x() {

    Object ret_1 = x;
    //@ assert project.Entry.invChecksOn ==> ((((((ret_1 == null) || Utils.
        is_nat(ret_1)) && inv_Entry_PossiblyOne(ret_1)) || (Utils.is_bool(
        ret_1) && inv_Entry_True(ret_1))) && inv_Entry_PT(ret_1)));

    return ret_1;
  }

  public void set_x(final Object _x) {

    //@ assert project.Entry.invChecksOn ==> ((((((_x == null) || Utils.
        is_nat(_x)) && inv_Entry_PossiblyOne(_x)) || (Utils.is_bool(_x) &&
        inv_Entry_True(_x))) && inv_Entry_PT(_x)));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> ((((((x == null) || Utils.
        is_nat(x)) && inv_Entry_PossiblyOne(x)) || (Utils.is_bool(x) &&
        inv_Entry_True(x))) && inv_Entry_PT(x)));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_PT(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_PossiblyOne(final Object check_p) {

    Number p = ((Number) check_p);

    Boolean orResult_1 = false;

    if (!(!(Utils.equals(p, null)))) {
      orResult_1 = true;
    } else {
      orResult_1 = Utils.equals(p, 1L);
    }

    return orResult_1;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_True(final Object check_b) {

    Boolean b = ((Boolean) check_b);

    return b;
  }
}
```

C.17.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ spec_public @*/

  private static project.Entrytypes.St St = new project.Entrytypes.St(1L);
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static void op1() {

    Object p = null;
    //@ assert ((((p == null) || Utils.is_nat(p)) && inv_Entry_PossiblyOne(
        p)) || (Utils.is_bool(p) && inv_Entry_True(p))) && inv_Entry_PT(p));

    p = 1L;
    //@ assert ((((p == null) || Utils.is_nat(p)) && inv_Entry_PossiblyOne(
        p)) || (Utils.is_bool(p) && inv_Entry_True(p))) && inv_Entry_PT(p));

    p = true;
    //@ assert ((((p == null) || Utils.is_nat(p)) && inv_Entry_PossiblyOne(
        p)) || (Utils.is_bool(p) && inv_Entry_True(p))) && inv_Entry_PT(p));

    //@ assert St != null;

    St.set_x(null);

    //@ assert St != null;

    St.set_x(1L);

    //@ assert St != null;

    St.set_x(true);

    IO.println("Breaking␣named␣type␣invariant␣(assigning␣record␣field)");
    //@ assert St != null;

    St.set_x(false);
  }

  public static void op2() {

    Object p1 = null;
    //@ assert ((((p1 == null) || Utils.is_nat(p1)) &&
        inv_Entry_PossiblyOne(p1)) || (Utils.is_bool(p1) && inv_Entry_True(
        p1))) && inv_Entry_PT(p1));

    //@ assert St != null;

    St.set_x(true);

    IO.println("Breaking␣named␣type␣invariant␣(assigning␣local␣variable)");
    p1 = false;
    //@ assert ((((p1 == null) || Utils.is_nat(p1)) &&
        inv_Entry_PossiblyOne(p1)) || (Utils.is_bool(p1) && inv_Entry_True(
        p1))) && inv_Entry_PT(p1));

  }

  public static Object Run() {

    op1();
    op2();
    return 0L;
  }

  public String toString() {

    return "Entry{" + "St␣:=␣" + Utils.toString(St) + "}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_PT(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_PossiblyOne(final Object check_p) {

    Number p = ((Number) check_p);

    Boolean orResult_1 = false;

    if (!(!(Utils.equals(p, null)))) {
      orResult_1 = true;
    } else {
      orResult_1 = Utils.equals(p, 1L);
    }

    return orResult_1;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_True(final Object check_b) {

    Boolean b = ((Boolean) check_b);

    return b;
  }
}
```

C.17.4 The OpenJML runtime assertion checker output

```
"Breaking named type invariant (assigning record field)"
St.java:63: JML assertion is false
    //@ assert project.Entry.invChecksOn ==> ((((((_x == null) || Utils.is_nat
        (_x)) && inv_Entry_PossiblyOne(_x)) || (Utils.is_bool(_x) &&
        inv_Entry_True(_x))) && inv_Entry_PT(_x)));
        ^
St.java:66: JML assertion is false
    //@ assert project.Entry.invChecksOn ==> ((((((x == null) || Utils.is_nat(
        x)) && inv_Entry_PossiblyOne(x)) || (Utils.is_bool(x) && inv_Entry_True
        (x))) && inv_Entry_PT(x)));
        ^
"Breaking named type invariant (assigning local variable)"
Entry.java:58: JML assertion is false
    //@ assert ((((p1 == null) || Utils.is_nat(p1)) && inv_Entry_PossiblyOne(
        p1)) || (Utils.is_bool(p1) && inv_Entry_True(p1))) && inv_Entry_PT(p1))
        ;
        ^
```

C.18 CaseExp.vdmsl

C.18.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

```

```
Run : () ==> ?
Run () ==
(
  let - = f(2) in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  nat -> nat
f (a) ==
  cases a:
        1 -> 4,
        2 -> 8,
        others -> 2
end;

end Entry
```

C.18.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f(2L);
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number f(final Number a) {

    //@ assert Utils.is_nat(a);

    Number casesExpResult_1 = null;

    Number intPattern_1 = a;
    //@ assert Utils.is_nat(intPattern_1);

    Boolean success_1 = Utils.equals(intPattern_1, 1L);
    //@ assert Utils.is_bool(success_1);

    if (!(success_1)) {
      Number intPattern_2 = a;
      //@ assert Utils.is_nat(intPattern_2);

      success_1 = Utils.equals(intPattern_2, 2L);
      //@ assert Utils.is_bool(success_1);

      if (success_1) {
        casesExpResult_1 = 8L;
        //@ assert Utils.is_nat1(casesExpResult_1);

      } else {
        casesExpResult_1 = 2L;
        //@ assert Utils.is_nat1(casesExpResult_1);

      }

    } else {
      casesExpResult_1 = 4L;
      //@ assert Utils.is_nat1(casesExpResult_1);

    }

    Number ret_1 = casesExpResult_1;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.18.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.19 TernaryIf.vdmsl

C.19.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

```

```
Rec :: x : int
inv r == r.x > 0;

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  let - = g() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

g :  () -> nat
g () ==
let x = if 1 = 1 then 1 else 2
in
  x;

f :  () -> nat
f () ==
  if 1 = 1 then 1 else 2;

end Entry
```

C.19.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    {
      final Number ignorePattern_2 = g();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number g() {

    Number ternaryIfExp_1 = null;

    if (Utils.equals(1L, 1L)) {
      ternaryIfExp_1 = 1L;
      //@ assert Utils.is_nat1(ternaryIfExp_1);

    } else {
      ternaryIfExp_1 = 2L;
      //@ assert Utils.is_nat1(ternaryIfExp_1);

    }

    final Number x = ternaryIfExp_1;
    //@ assert Utils.is_nat1(x);

    Number ret_1 = x;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }
  /*@ pure @*/

  public static Number f() {

    if (Utils.equals(1L, 1L)) {
      Number ret_2 = 1L;
      //@ assert Utils.is_nat(ret_2);

      return ret_2;

    } else {
      Number ret_3 = 2L;
      //@ assert Utils.is_nat(ret_3);

      return ret_3;
    }
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.19.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Rec implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_Rec(x);

  public Rec(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.Rec)) {
      return false;
    }

    project.Entrytypes.Rec other = ((project.Entrytypes.Rec) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.Rec copy() {

    return new project.Entrytypes.Rec(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`Rec" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_4 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_4));

    return ret_4;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Rec(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.19.4 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.20 LetBeStStm.vdmsl

C.20.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - in set {1,2,3} in skip;
  let x in set {1,2,3} be st x > 1 in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

end Entry
```

C.20.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      Number ignorePattern_1 = null;

      Boolean success_1 = false;
      //@ assert Utils.is_bool(success_1);

      VDMSet set_1 = SetUtil.set(1L, 2L, 3L);
      //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size
          (set_1); Utils.is_nat1(V2J.get(set_1,i))));

      for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() &&
          !(success_1); ) {
        ignorePattern_1 = ((Number) iterator_1.next());
        success_1 = true;
        //@ assert Utils.is_bool(success_1);

      }
      if (!(success_1)) {
        throw new RuntimeException("Let␣Be␣St␣found␣no␣applicable␣bindings")
            ;
      }

      /* skip */
    }

    {
      Number x = null;

      Boolean success_2 = false;
      //@ assert Utils.is_bool(success_2);

      VDMSet set_2 = SetUtil.set(1L, 2L, 3L);
      //@ assert (V2J.isSet(set_2) && (\forall int i; 0 <= i && i < V2J.size
          (set_2); Utils.is_nat1(V2J.get(set_2,i))));

      for (Iterator iterator_2 = set_2.iterator(); iterator_2.hasNext() &&
          !(success_2); ) {
        x = ((Number) iterator_2.next());
        success_2 = x.longValue() > 1L;
        //@ assert Utils.is_bool(success_2);

      }
      if (!(success_2)) {
        throw new RuntimeException("Let␣Be␣St␣found␣no␣applicable␣bindings")
            ;
      }

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.20.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.21 LetBeStExp.vdmsl

C.21.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  let - = g() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> nat
f () ==
  let - in set {1,2,3} in 0;

g : () -> nat
g () ==
  let x in set {1,2,3} be st x > 1 in 0;

end Entry
```

C.21.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    {
      final Number ignorePattern_2 = g();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number f() {

    Number letBeStExp_1 = null;
    Number ignorePattern_3 = null;

    Boolean success_1 = false;
    //@ assert Utils.is_bool(success_1);

    VDMSet set_1 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_nat1(V2J.get(set_1,i))));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() && !(
        success_1); ) {
      ignorePattern_3 = ((Number) iterator_1.next());
      success_1 = true;
      //@ assert Utils.is_bool(success_1);

    }
    if (!(success_1)) {
      throw new RuntimeException("Let␣Be␣St␣found␣no␣applicable␣bindings");
    }

    letBeStExp_1 = 0L;
    //@ assert Utils.is_nat(letBeStExp_1);

    Number ret_1 = letBeStExp_1;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }
  /*@ pure @*/

  public static Number g() {

    Number letBeStExp_2 = null;
    Number x = null;

    Boolean success_2 = false;
    //@ assert Utils.is_bool(success_2);

    VDMSet set_2 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_2) && (\forall int i; 0 <= i && i < V2J.size(
        set_2); Utils.is_nat1(V2J.get(set_2,i))));

    for (Iterator iterator_2 = set_2.iterator(); iterator_2.hasNext() && !(
        success_2); ) {
      x = ((Number) iterator_2.next());
      success_2 = x.longValue() > 1L;
      //@ assert Utils.is_bool(success_2);

    }
    if (!(success_2)) {
      throw new RuntimeException("Let␣Be␣St␣found␣no␣applicable␣bindings");
    }

    letBeStExp_2 = 0L;
    //@ assert Utils.is_nat(letBeStExp_2);

    Number ret_2 = letBeStExp_2;
    //@ assert Utils.is_nat(ret_2);

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.21.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.22 RealParamNil.vdmsl

C.22.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(dcl r : [real] := 1.23;
  IO`println("Before␣valid␣use.");
  doSkip(r);
  r := nil;
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  doSkip(r);
  IO`println("After␣invalid␣use.");
  return 0;
);

operations

doSkip :  real ==> ()
doSkip (-) == skip;

end Entry
```

C.22.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Number r = 1.23;
    //@ assert ((r == null) || Utils.is_real(r));

    IO.println("Before␣valid␣use.");
    doSkip(r);
    r = null;
    //@ assert ((r == null) || Utils.is_real(r));

    IO.println("After␣valid␣use.");
    IO.println("Before␣invalid␣use.");
    doSkip(r);
    IO.println("After␣invalid␣use.");
    return 0L;
  }

  public static void doSkip(final Number ignorePattern_1) {

    //@ assert Utils.is_real(ignorePattern_1);

    /* skip */

  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.22.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:34: JML assertion is false
    //@ assert Utils.is_real(ignorePattern_1);
        ^
"After invalid use."
```

C.23 QuoteAssignNil.vdmsl

C.23.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣valid␣use.");
  (dcl aOpt : [<A>] := nil;skip);
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  (dcl a : <A> := Nil(); skip);
  IO`println("After␣invalid␣use.");
  return 0;
);

functions

Nil :  () -> [<A>]
Nil () == nil;
```

```

end Entry
```

C.23.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣valid␣use.");
    {
      project.quotes.AQuote aOpt = null;
      //@ assert ((aOpt == null) || Utils.is_(aOpt,project.quotes.AQuote.
          class));

      /* skip */
    }

    IO.println("After␣valid␣use.");
    IO.println("Before␣invalid␣use.");
    {
      project.quotes.AQuote a = Nil();
      //@ assert Utils.is_(a,project.quotes.AQuote.class);

      /* skip */
    }

    IO.println("After␣invalid␣use.");
    return 0L;
  }
  /*@ pure @*/

  public static project.quotes.AQuote Nil() {

    project.quotes.AQuote ret_1 = null;
    //@ assert ((ret_1 == null) || Utils.is_(ret_1,project.quotes.AQuote.
        class));

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.23.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:29: JML assertion is false
      //@ assert Utils.is_(a,project.quotes.AQuote.class);
          ^
"After invalid use."
```

C.24 NatParamNil.vdmsl

C.24.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  dcl n : [nat] := 0;
  IO`println("Before␣valid␣use.");
  n := 1;
  n := nil;
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  n := idNat(n);
  IO`println("After␣invalid␣use.");
  return 0;
);

functions

idNat :  nat -> nat
idNat (x) == x;

end Entry
```

C.24.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
```

```
11    /*@ public ghost static boolean invChecksOn = true; @*/
12
13    private Entry() {}
14
15    public static Object Run() {
16
17      Number n = 0L;
18      //@ assert ((n == null) || Utils.is_nat(n));
19
20      IO.println("Before␣valid␣use.");
21      n = 1L;
22      //@ assert ((n == null) || Utils.is_nat(n));
23
24      n = null;
25      //@ assert ((n == null) || Utils.is_nat(n));
26
27      IO.println("After␣valid␣use.");
28      IO.println("Before␣invalid␣use.");
29      n = idNat(n);
30      //@ assert ((n == null) || Utils.is_nat(n));
31
32      IO.println("After␣invalid␣use.");
33      return 0L;
34    }
35    /*@ pure @*/
36
37    public static Number idNat(final Number x) {
38
39      //@ assert Utils.is_nat(x);
40
41      Number ret_1 = x;
42      //@ assert Utils.is_nat(ret_1);
43
44      return ret_1;
45    }
46
47    public String toString() {
48
49      return "Entry{}";
50    }
51  }
```

C.24.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:39: JML assertion is false
    //@ assert Utils.is_nat(x);
        ^
Entry.java:42: JML assertion is false
    //@ assert Utils.is_nat(ret_1);
        ^
"After invalid use."
```

C.25 CharReturnNil.vdmsl

C.25.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣valid␣use.");
  let - : char = charA() in skip;
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  let - : char = charNil() in skip;
  IO`println("After␣invalid␣use.");
  return 0;
);

functions

charA :  () -> [char]
charA () == 'a';

charNil :  () -> [char]
charNil () == nil;

end Entry
```

C.25.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣valid␣use.");
    {
      final Character ignorePattern_1 = charA();
      //@ assert Utils.is_char(ignorePattern_1);

      /* skip */
23      }
24  
25      IO.println("After␣valid␣use.");
26      IO.println("Before␣invalid␣use.");
27      {
28        final Character ignorePattern_2 = charNil();
29        //@ assert Utils.is_char(ignorePattern_2);
30  
31        /* skip */
32      }
33  
34      IO.println("After␣invalid␣use.");
35      return 0L;
36    }
37    /*@ pure @*/
38  
39    public static Character charA() {
40  
41      Character ret_1 = 'a';
42      //@ assert ((ret_1 == null) || Utils.is_char(ret_1));
43  
44      return ret_1;
45    }
46    /*@ pure @*/
47  
48    public static Character charNil() {
49  
50      Character ret_2 = null;
51      //@ assert ((ret_2 == null) || Utils.is_char(ret_2));
52  
53      return ret_2;
54    }
55  
56    public String toString() {
57  
58      return "Entry{}";
59    }
60  }
```

C.25.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:29: JML assertion is false
      //@ assert Utils.is_char(ignorePattern_2);
          ^
"After invalid use."
```

C.26 Nat1InitWithZero.vdmsl

C.26.1 The VDM-SL model

```
module Entry

```

```
exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  dcl n : nat1 := 1;
  IO`println("Before␣valid␣use.");
  n := 1;
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  (dcl n1 : nat1 := -1 + 1; skip);
  IO`println("After␣invalid␣use.");
  return 0;
);

end Entry
```

C.26.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Number n = 1L;
    //@ assert Utils.is_nat1(n);

    IO.println("Before␣valid␣use.");
    n = 1L;
    //@ assert Utils.is_nat1(n);

    IO.println("After␣valid␣use.");
    IO.println("Before␣invalid␣use.");
    {
      Number n1 = -1L + 1L;
      //@ assert Utils.is_nat1(n1);

      /* skip */
    }

    IO.println("After␣invalid␣use.");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.26.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:28: JML assertion is false
      //@ assert Utils.is_nat1(n1);
          ^
"After invalid use."
```

C.27 IntAssignNonInt.vdmsl

C.27.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  dcl i : int := -1;
  IO`println("Before␣valid␣use.");
  i := 1;
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  i := i + 0.5;
  IO`println("After␣invalid␣use.");
  return 0;
);

end Entry
```

C.27.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
```

```
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Number i = -1L;
    //@ assert Utils.is_int(i);

    IO.println("Before␣valid␣use.");
    i = 1L;
    //@ assert Utils.is_int(i);

    IO.println("After␣valid␣use.");
    IO.println("Before␣invalid␣use.");
    i = i.longValue() + 0.5;
    //@ assert Utils.is_int(i);

    IO.println("After␣invalid␣use.");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.27.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:27: JML assertion is false
    //@ assert Utils.is_int(i);
        ^
"After invalid use."
```

C.28 BoolReturnNil.vdmsl

C.28.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  dcl b : bool;
  IO`println("Before␣valid␣use.");
  b := true;
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  b := boolNil();
  IO`println("After␣invalid␣use.");
  return 0;
);

functions

boolTrue :  () -> bool
boolTrue () == true;

boolNil : () -> [bool]
boolNil () == nil;

end Entry
```

C.28.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Boolean b = false;
    //@ assert Utils.is_bool(b);

    IO.println("Before␣valid␣use.");
    b = true;
    //@ assert Utils.is_bool(b);

    IO.println("After␣valid␣use.");
    IO.println("Before␣invalid␣use.");
    b = boolNil();
    //@ assert Utils.is_bool(b);

    IO.println("After␣invalid␣use.");
    return 0L;
  }
  /*@ pure @*/

  public static Boolean boolTrue() {

    Boolean ret_1 = true;
    //@ assert Utils.is_bool(ret_1);

    return ret_1;
  }
  /*@ pure @*/

  public static Boolean boolNil() {

    Boolean ret_2 = null;
    //@ assert ((ret_2 == null) || Utils.is_bool(ret_2));

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.28.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:27: JML assertion is false
    //@ assert Utils.is_bool(b);
        ^
"After invalid use."
```

C.29 RatAssignBool.vdmsl

C.29.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  dcl i : rat := 123.456;
  IO`println("Before␣valid␣use.");
  i := i * i;
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  i := ratOpt();
  IO`println("After␣invalid␣use.");
  return 0;
);
```

```

functions

ratOpt :  () -> [rat]
ratOpt () == nil;

end Entry
```

C.29.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Number i = 123.456;
    //@ assert Utils.is_rat(i);

    IO.println("Before␣valid␣use.");
    i = i.doubleValue() * i.doubleValue();
    //@ assert Utils.is_rat(i);

    IO.println("After␣valid␣use.");
    IO.println("Before␣invalid␣use.");
    i = ratOpt();
    //@ assert Utils.is_rat(i);

    IO.println("After␣invalid␣use.");
    return 0L;
  }
  /*@ pure @*/

  public static Number ratOpt() {

    Number ret_1 = null;
    //@ assert ((ret_1 == null) || Utils.is_rat(ret_1));

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.29.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:27: JML assertion is false
    //@ assert Utils.is_rat(i);
        ^
"After invalid use."
```

C.30 TokenAssignNil.vdmsl

C.30.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

values

n : [token] = nil;
t : [token] = mk_token("");

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣valid␣use.");
  let - : token = t in skip;
  IO`println("After␣valid␣use.");
  IO`println("Before␣invalid␣use.");
  let - : token = n in skip;
  IO`println("After␣invalid␣use.");
  return 0;
);

end Entry
```

C.30.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  //@ public static invariant ((n == null) || Utils.is_token(n));

  public static final Token n = null;
  //@ public static invariant ((t == null) || Utils.is_token(t));

  public static final Token t = new Token("");
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣valid␣use.");
    {
      final Token ignorePattern_1 = t;
      //@ assert Utils.is_token(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣valid␣use.");
    IO.println("Before␣invalid␣use.");
    {
      final Token ignorePattern_2 = n;
      //@ assert Utils.is_token(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣invalid␣use.");
    return 0L;
  }

  public String toString() {

    return "Entry{" + "n␣=␣" + Utils.toString(n) + ",␣t␣=␣" + Utils.toString
        (t) + "}";
  }
}
```

C.30.3 The OpenJML runtime assertion checker output

```
"Before valid use."
"After valid use."
"Before invalid use."
Entry.java:35: JML assertion is false
      //@ assert Utils.is_token(ignorePattern_2);
          ^
"After invalid use."
```

C.31 RecLet.vdmsl

C.31.1 The VDM-SL model

```
module Entry

```

```
exports all
imports from IO all
definitions

types

R ::
  x : nat
  y : nat;

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> nat
f () ==
let mk_R(a,b) = mk_R(1,2)
in
  a + b;

end Entry
```

C.31.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number f() {

    final project.Entrytypes.R recordPattern_1 = new project.Entrytypes.R(1L
        , 2L);
    //@ assert Utils.is_(recordPattern_1,project.Entrytypes.R.class);

    Boolean success_1 = true;
    //@ assert Utils.is_bool(success_1);

    Number a = null;

    Number b = null;

    a = recordPattern_1.get_x();
    //@ assert Utils.is_nat(a);

    b = recordPattern_1.get_y();
    //@ assert Utils.is_nat(b);

    if (!(success_1)) {
      throw new RuntimeException("Record␣pattern␣match␣failed");
    }

    Number ret_1 = a.longValue() + b.longValue();
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.31.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R implements Record {
  public Number x;
  public Number y;

  public R(final Number _x, final Number _y) {

    //@ assert Utils.is_nat(_x);

    //@ assert Utils.is_nat(_y);

    x = _x;
    //@ assert Utils.is_nat(x);

    y = _y;
    //@ assert Utils.is_nat(y);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R)) {
      return false;
    }

    project.Entrytypes.R other = ((project.Entrytypes.R) obj);

    return (Utils.equals(x, other.x)) && (Utils.equals(y, other.y));
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x, y);
  }
  /*@ pure @*/

  public project.Entrytypes.R copy() {

    return new project.Entrytypes.R(x, y);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R" + Utils.formatFields(x, y);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(x));

  }
  /*@ pure @*/

  public Number get_y() {

    Number ret_3 = y;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_3));

    return ret_3;
  }

  public void set_y(final Number _y) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_y));

    y = _y;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(y));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
}
```

C.31.4 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.32 TupLet.vdmsl

C.32.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> nat
f () ==
let mk_(a,b) = mk_(1,2)
in
  a + b;

end Entry
```

C.32.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number f() {

    final Tuple tuplePattern_1 = Tuple.mk_(1L, 2L);
    //@ assert (V2J.isTup(tuplePattern_1,2) && Utils.is_nat1(V2J.field(
        tuplePattern_1,0)) && Utils.is_nat1(V2J.field(tuplePattern_1,1)));

    Boolean success_1 = tuplePattern_1.compatible(Number.class, Number.class
        );
    //@ assert Utils.is_bool(success_1);

    Number a = null;

    Number b = null;

    if (success_1) {
      a = ((Number) tuplePattern_1.get(0));
      //@ assert Utils.is_nat1(a);

      b = ((Number) tuplePattern_1.get(1));
      //@ assert Utils.is_nat1(b);

    }

    if (!(success_1)) {
      throw new RuntimeException("Tuple␣pattern␣match␣failed");
    }

    Number ret_1 = a.longValue() + b.longValue();
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.32.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.33 RecParam.vdmsl

C.33.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

R ::
 b : bool;

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> bool
f () ==
let mk_R(true) in set {mk_R(false), mk_R(true)}
in
  true;

end Entry
```

C.33.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Boolean ignorePattern_1 = f();
      //@ assert Utils.is_bool(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Boolean f() {

    Boolean letBeStExp_1 = null;
    project.Entrytypes.R recordPattern_1 = null;

    Boolean success_1 = false;
    //@ assert Utils.is_bool(success_1);

    VDMSet set_1 = SetUtil.set(new project.Entrytypes.R(false), new project.
        Entrytypes.R(true));
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_(V2J.get(set_1,i),project.Entrytypes.R.class)));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() && !(
        success_1); ) {
      recordPattern_1 = ((project.Entrytypes.R) iterator_1.next());
      //@ assert Utils.is_(recordPattern_1,project.Entrytypes.R.class);

      success_1 = true;
      //@ assert Utils.is_bool(success_1);

      Boolean boolPattern_1 = recordPattern_1.get_b();
      //@ assert Utils.is_bool(boolPattern_1);

      success_1 = Utils.equals(boolPattern_1, true);
      //@ assert Utils.is_bool(success_1);

      if (!(success_1)) {
        continue;
      }

      success_1 = true;
      //@ assert Utils.is_bool(success_1);

    }
    if (!(success_1)) {
      throw new RuntimeException("Let␣Be␣St␣found␣no␣applicable␣bindings");
    }

    letBeStExp_1 = true;
    //@ assert Utils.is_bool(letBeStExp_1);

    Boolean ret_1 = letBeStExp_1;
    //@ assert Utils.is_bool(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.33.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R implements Record {
  public Boolean b;

  public R(final Boolean _b) {

    //@ assert Utils.is_bool(_b);

    b = _b;
    //@ assert Utils.is_bool(b);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R)) {
      return false;
    }

    project.Entrytypes.R other = ((project.Entrytypes.R) obj);

    return Utils.equals(b, other.b);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(b);
  }
  /*@ pure @*/

  public project.Entrytypes.R copy() {

    return new project.Entrytypes.R(b);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R" + Utils.formatFields(b);
  }
  /*@ pure @*/

  public Boolean get_b() {

    Boolean ret_2 = b;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_bool(ret_2));

    return ret_2;
  }

  public void set_b(final Boolean _b) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_bool(_b));

    b = _b;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_bool(b));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
}
```

C.33.4 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.34 TupParam.vdmsl

C.34.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

```

```
Run : () ==> ?
Run () ==
(
  let - = f(mk_(4,'a')) in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  (nat * char) -> nat
f (mk_(a,-)) ==
  a;

end Entry
```

C.34.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f(Tuple.mk_(4L, 'a'));
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number f(final Tuple tuplePattern_1) {

    //@ assert (V2J.isTup(tuplePattern_1,2) && Utils.is_nat(V2J.field(
        tuplePattern_1,0)) && Utils.is_char(V2J.field(tuplePattern_1,1)));

    Boolean success_1 = tuplePattern_1.compatible(Number.class, Character.
        class);
    //@ assert Utils.is_bool(success_1);

    Number a = null;

    if (success_1) {
      a = ((Number) tuplePattern_1.get(0));
      //@ assert Utils.is_nat(a);

    }

    if (!(success_1)) {
      throw new RuntimeException("Tuple␣pattern␣match␣failed");
    }

    Number ret_1 = a;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.34.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.35 SeqNat1BoolMaskedAsNamedTypeInv.vdmsl

C.35.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

SeqNat1Bool = seq of (nat1 | bool);

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : SeqNat1Bool = [1,true,2,false,3] in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : SeqNat1Bool = [1,true,2,false,minusOne()] in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

```

```
minusOne :  () -> int
minusOne () == -1;

end Entry
```

C.35.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMSeq ignorePattern_1 = SeqUtil.seq(1L, true, 2L, false, 3L);
      //@ assert ((V2J.isSeq(ignorePattern_1) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_1); (Utils.is_bool(V2J.get(
          ignorePattern_1,i)) || Utils.is_nat1(V2J.get(ignorePattern_1,i))))
          ) && inv_Entry_SeqNat1Bool(ignorePattern_1));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMSeq ignorePattern_2 = SeqUtil.seq(1L, true, 2L, false,
          minusOne());
      //@ assert ((V2J.isSeq(ignorePattern_2) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_2); (Utils.is_bool(V2J.get(
          ignorePattern_2,i)) || Utils.is_nat1(V2J.get(ignorePattern_2,i))))
          ) && inv_Entry_SeqNat1Bool(ignorePattern_2));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static Number minusOne() {

    Number ret_1 = -1L;
    //@ assert Utils.is_int(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_SeqNat1Bool(final Object check_elem) {

    return true;
  }
}
```

C.35.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert ((V2J.isSeq(ignorePattern_2) && (\forall int i; 0 <= i && i <
           V2J.size(ignorePattern_2); (Utils.is_bool(V2J.get(ignorePattern_2,i)
          ) || Utils.is_nat1(V2J.get(ignorePattern_2,i))))) &&
          inv_Entry_SeqNat1Bool(ignorePattern_2));
          ^
"After illegal use"
```

C.36 SeqOfNatNilElem.vdmsl

C.36.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : seq of nat = [1,2,3] in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣uses");
 let - : seq of nat = seqOfNatsAndNil() in skip;
 IO`println("After␣illegal␣uses");
 return 0;
);

functions

```

```
seqOfNatsAndNil :  () -> seq of [nat]
seqOfNatsAndNil () == [1,nil,3];

end Entry
```

C.36.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMSeq ignorePattern_1 = SeqUtil.seq(1L, 2L, 3L);
      //@ assert (V2J.isSeq(ignorePattern_1) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_1); Utils.is_nat(V2J.get(ignorePattern_1,
          i))));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣uses");
    {
      final VDMSeq ignorePattern_2 = seqOfNatsAndNil();
      //@ assert (V2J.isSeq(ignorePattern_2) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(ignorePattern_2,
          i))));

      /* skip */
    }

    IO.println("After␣illegal␣uses");
    return 0L;
  }
  /*@ pure @*/

  public static VDMSeq seqOfNatsAndNil() {

    VDMSeq ret_1 = SeqUtil.seq(1L, null, 3L);
    //@ assert (V2J.isSeq(ret_1) && (\forall int i; 0 <= i && i < V2J.size(
        ret_1); ((V2J.get(ret_1,i) == null) || Utils.is_nat(V2J.get(ret_1,i)
        ))));

    return Utils.copy(ret_1);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.36.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal uses"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isSeq(ignorePattern_2) && (\forall int i; 0 <= i && i <
          V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(ignorePattern_2,i))))
          ;
          ^
"After illegal uses"
```

C.37 Seq1EvenNatsMaskedAsNamedTypeInv.vdmsl

C.37.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

Seq1Even = seq1 of nat
inv xs == forall x in set elems xs & x mod 2 = 0;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : Seq1Even = [2,4,6] in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : Seq1Even = [2,4,6,9] in skip;
 let - : Seq1Even = emptySeqOfNat() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

emptySeqOfNat :  () -> seq of nat
emptySeqOfNat () == [];

end Entry
```

C.37.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMSeq ignorePattern_1 = SeqUtil.seq(2L, 4L, 6L);
      //@ assert ((V2J.isSeq1(ignorePattern_1) && (\forall int i; 0 <= i &&
          i < V2J.size(ignorePattern_1); Utils.is_nat(V2J.get(
          ignorePattern_1,i)))) && inv_Entry_Seq1Even(ignorePattern_1));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMSeq ignorePattern_2 = SeqUtil.seq(2L, 4L, 6L, 9L);
      //@ assert ((V2J.isSeq1(ignorePattern_2) && (\forall int i; 0 <= i &&
          i < V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(
          ignorePattern_2,i)))) && inv_Entry_Seq1Even(ignorePattern_2));

      /* skip */
    }

    {
      final VDMSeq ignorePattern_3 = emptySeqOfNat();
      //@ assert ((V2J.isSeq1(ignorePattern_3) && (\forall int i; 0 <= i &&
          i < V2J.size(ignorePattern_3); Utils.is_nat(V2J.get(
          ignorePattern_3,i)))) && inv_Entry_Seq1Even(ignorePattern_3));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static VDMSeq emptySeqOfNat() {

    VDMSeq ret_1 = SeqUtil.seq();
    //@ assert (V2J.isSeq(ret_1) && (\forall int i; 0 <= i && i < V2J.size(
        ret_1); Utils.is_nat(V2J.get(ret_1,i))));

    return Utils.copy(ret_1);
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Seq1Even(final Object check_xs) {

    VDMSeq xs = ((VDMSeq) check_xs);

    Boolean forAllExpResult_1 = true;
    VDMSet set_1 = SeqUtil.elems(Utils.copy(xs));
    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() &&
        forAllExpResult_1; ) {
      Number x = ((Number) iterator_1.next());
      forAllExpResult_1 = Utils.equals(Utils.mod(x.longValue(), 2L), 0L);
    }
    return forAllExpResult_1;
  }
}
```

C.37.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert ((V2J.isSeq1(ignorePattern_2) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(ignorePattern_2,i))
          )) && inv_Entry_Seq1Even(ignorePattern_2));
          ^
Entry.java:36: JML assertion is false
      //@ assert ((V2J.isSeq1(ignorePattern_3) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_3); Utils.is_nat(V2J.get(ignorePattern_3,i))
          )) && inv_Entry_Seq1Even(ignorePattern_3));
          ^
"After illegal use"
```

C.38 Seq1AssignEmptySet.vdmsl

C.38.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : seq1 of nat = [1] in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : seq1 of nat = emptySeqOfNat() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

emptySeqOfNat :  () -> seq of nat
emptySeqOfNat () == [];

end Entry
```

C.38.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMSeq ignorePattern_1 = SeqUtil.seq(1L);
      //@ assert (V2J.isSeq1(ignorePattern_1) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_1); Utils.is_nat(V2J.get(ignorePattern_1
          ,i))));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMSeq ignorePattern_2 = emptySeqOfNat();
      //@ assert (V2J.isSeq1(ignorePattern_2) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(ignorePattern_2
          ,i))));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static VDMSeq emptySeqOfNat() {

    VDMSeq ret_1 = SeqUtil.seq();
    //@ assert (V2J.isSeq(ret_1) && (\forall int i; 0 <= i && i < V2J.size(
        ret_1); Utils.is_nat(V2J.get(ret_1,i))));

    return Utils.copy(ret_1);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.38.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isSeq1(ignorePattern_2) && (\forall int i; 0 <= i && i <
           V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(ignorePattern_2,i)))
          );
          ^
"After illegal use"
```

C.39 SeqEven.vdmsl

C.39.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

SeqEven = seq of Even;

Even = nat
inv e == e mod 2 = 0;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : SeqEven = [] in skip;
```

```
  let - : SeqEven = [2,4,6,8] in skip;
  IO`println("After␣legal␣use");
  IO`println("Before␣illegal␣use");
  let - : SeqEven = [2,4,6,8,9] in skip;
  IO`println("After␣illegal␣use");
  return 0;
);

end Entry
```

C.39.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMSeq ignorePattern_1 = SeqUtil.seq();
      //@ assert ((V2J.isSeq(ignorePattern_1) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_1); (Utils.is_nat(V2J.get(
          ignorePattern_1,i)) && inv_Entry_Even(V2J.get(ignorePattern_1,i)))
          )) && inv_Entry_SeqEven(ignorePattern_1));

      /* skip */
    }

    {
      final VDMSeq ignorePattern_2 = SeqUtil.seq(2L, 4L, 6L, 8L);
      //@ assert ((V2J.isSeq(ignorePattern_2) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_2); (Utils.is_nat(V2J.get(
          ignorePattern_2,i)) && inv_Entry_Even(V2J.get(ignorePattern_2,i)))
          )) && inv_Entry_SeqEven(ignorePattern_2));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMSeq ignorePattern_3 = SeqUtil.seq(2L, 4L, 6L, 8L, 9L);
      //@ assert ((V2J.isSeq(ignorePattern_3) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_3); (Utils.is_nat(V2J.get(
          ignorePattern_3,i)) && inv_Entry_Even(V2J.get(ignorePattern_3,i)))
          )) && inv_Entry_SeqEven(ignorePattern_3));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_SeqEven(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Even(final Object check_e) {

    Number e = ((Number) check_e);

    return Utils.equals(Utils.mod(e.longValue(), 2L), 0L);
  }
}
```

C.39.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:36: JML assertion is false
      //@ assert ((V2J.isSeq(ignorePattern_3) && (\forall int i; 0 <= i && i <
           V2J.size(ignorePattern_3); (Utils.is_nat(V2J.get(ignorePattern_3,i))
           && inv_Entry_Even(V2J.get(ignorePattern_3,i))))) &&
          inv_Entry_SeqEven(ignorePattern_3));
           ^
"After illegal use"
```

C.40 RecUnion.vdmsl

C.40.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions
types

```

```
R1 :: r2 : R2
inv r1 == r1.r2.x <> -1;

R2 :: x : int
inv r2 == r2.x <> -2;

operations

Run: () ==> ?
Run () ==
(
 dcl r1 : R1 | nat := mk_R1(mk_R2(5));
 r1.r2.x := -1;
 IO`println("\\invariant_for␣is␣not␣implemented␣in␣OpenJML␣RAC␣" ^
   "so␣the␣\\invariant_for␣check␣will␣not␣detect␣the␣invariant␣violation");
 return 0;
)

end Entry
```

C.40.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public project.Entrytypes.R2 r2;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(r2);

  public R1(final project.Entrytypes.R2 _r2) {

    //@ assert Utils.is_(_r2,project.Entrytypes.R2.class);

    r2 = _r2 != null ? Utils.copy(_r2) : null;
    //@ assert Utils.is_(r2,project.Entrytypes.R2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R1)) {
      return false;
    }

    project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

    return Utils.equals(r2, other.r2);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R1 copy() {

    return new project.Entrytypes.R1(r2);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R1" + Utils.formatFields(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 get_r2() {

    project.Entrytypes.R2 ret_1 = r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_1,project.
        Entrytypes.R2.class));

    return ret_1;
  }

  public void set_r2(final project.Entrytypes.R2 _r2) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r2,project.
        Entrytypes.R2.class));

    r2 = _r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r2,project.
        Entrytypes.R2.class));

  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R1(final project.Entrytypes.R2 _r2) {

    return !(Utils.equals(_r2.x, -1L));
  }
}
```

C.40.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R2(x);

  public R2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R2)) {
      return false;
    }

    project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 copy() {

    return new project.Entrytypes.R2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R2(final Number _x) {

    return !(Utils.equals(_x, -2L));
  }
}
```

C.40.4 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Object r1 = new project.Entrytypes.R1(new project.Entrytypes.R2(5L));
    //@ assert (Utils.is_(r1,project.Entrytypes.R1.class) || Utils.is_nat(r1
        ));

    project.Entrytypes.R2 apply_1 = null;
    if (r1 instanceof project.Entrytypes.R1) {
      apply_1 = ((project.Entrytypes.R1) r1).get_r2();
      //@ assert Utils.is_(apply_1,project.Entrytypes.R2.class);

    } else {
      throw new RuntimeException("Missing␣member:␣r2");
    }

    project.Entrytypes.R2 stateDes_1 = apply_1;
    //@ assert stateDes_1 != null;

    stateDes_1.set_x(-1L);
    //@ assert (Utils.is_(r1,project.Entrytypes.R1.class) || Utils.is_nat(r1
        ));

    //@ assert r1 instanceof project.Entrytypes.R1 ==> \invariant_for(((
        project.Entrytypes.R1) r1));

    IO.println(
        "\\invariant_for␣is␣not␣implemented␣in␣OpenJML␣RAC␣"
            + "so␣the␣\\invariant_for␣check␣will␣not␣detect␣the␣invariant␣
              violation");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.40.5 The OpenJML runtime assertion checker output

```
"\invariant_for is not implemented in OpenJML RAC so the \invariant_for check
    will not detect the invariant violation"
```

C.41 Simple.vdmsl

C.41.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions
types

R1 :: r2 : R2
inv r1 == r1.r2.x <> -1;

R2 :: x : int
inv r2 == r2.x <> -2;

operations

Run: () ==> ?
Run () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(5));
 r1.r2.x := -1;
 IO`println("\\invariant_for␣is␣not␣implemented␣in␣OpenJML␣RAC␣" ^
   "so␣the␣\\invariant_for␣check␣will␣not␣detect␣the␣invariant␣violation");
 return 0;
)
end Entry
```

C.41.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public project.Entrytypes.R2 r2;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(r2);

```

```
public R1(final project.Entrytypes.R2 _r2) {

  //@ assert Utils.is_(_r2,project.Entrytypes.R2.class);

  r2 = _r2 != null ? Utils.copy(_r2) : null;
  //@ assert Utils.is_(r2,project.Entrytypes.R2.class);

}
/*@ pure @*/

public boolean equals(final Object obj) {

  if (!(obj instanceof project.Entrytypes.R1)) {
    return false;
  }

  project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

  return Utils.equals(r2, other.r2);
}
/*@ pure @*/

public int hashCode() {

  return Utils.hashCode(r2);
}
/*@ pure @*/

public project.Entrytypes.R1 copy() {

  return new project.Entrytypes.R1(r2);
}
/*@ pure @*/

public String toString() {

  return "mk_Entry`R1" + Utils.formatFields(r2);
}
/*@ pure @*/

public project.Entrytypes.R2 get_r2() {

  project.Entrytypes.R2 ret_1 = r2;
  //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_1,project.
      Entrytypes.R2.class));

  return ret_1;
}

public void set_r2(final project.Entrytypes.R2 _r2) {

  //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r2,project.
      Entrytypes.R2.class));

  r2 = _r2;
  //@ assert project.Entry.invChecksOn ==> (Utils.is_(r2,project.
      Entrytypes.R2.class));

}
/*@ pure @*/
```

```
  /*@ helper @*/

  public static Boolean inv_R1(final project.Entrytypes.R2 _r2) {

    return !(Utils.equals(_r2.x, -1L));
  }
}
```

C.41.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R2(x);

  public R2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R2)) {
      return false;
    }

    project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 copy() {

    return new project.Entrytypes.R2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R2(final Number _x) {

    return !(Utils.equals(_x, -2L));
  }
}
```

C.41.4 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    project.Entrytypes.R1 r1 = new project.Entrytypes.R1(new project.
        Entrytypes.R2(5L));
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    project.Entrytypes.R2 stateDes_1 = r1.get_r2();
    //@ assert stateDes_1 != null;

    stateDes_1.set_x(-1L);
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    //@ assert \invariant_for(r1);

    IO.println(
        "\\invariant_for is not implemented in OpenJML RAC "
            + "so the \\invariant_for check will not detect the invariant 
                violation");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.41.5 The OpenJML runtime assertion checker output

```
"\invariant_for is not implemented in OpenJML RAC so the \invariant_for check
   will not detect the invariant violation"
```

C.42 AtomicRecUnion.vdmsl

C.42.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions
types

R1 :: r2 : R2
inv r1 == r1.r2.x <> -1;

R2 :: x : int
inv r2 == r2.x <> -2;

operations

Run: () ==> ?
Run () ==
(
 dcl r1 : R1 | nat := mk_R1(mk_R2(5));

 atomic
 (
   r1.r2.x := -1;
   r1.r2.x := 1;
 );

 IO`println("\\invariant_for is not implemented in OpenJML RAC " ^
   "so the \\invariant_for check will not detect the invariant violation");
 return 0;
)
end Entry
```

C.42.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public project.Entrytypes.R2 r2;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(r2);

  public R1(final project.Entrytypes.R2 _r2) {

    //@ assert Utils.is_(_r2,project.Entrytypes.R2.class);

    r2 = _r2 != null ? Utils.copy(_r2) : null;
    //@ assert Utils.is_(r2,project.Entrytypes.R2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R1)) {
      return false;
    }

    project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

    return Utils.equals(r2, other.r2);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R1 copy() {

    return new project.Entrytypes.R1(r2);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R1" + Utils.formatFields(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 get_r2() {

    project.Entrytypes.R2 ret_1 = r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_1,project.
        Entrytypes.R2.class));

    return ret_1;
  }

  public void set_r2(final project.Entrytypes.R2 _r2) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r2,project.
        Entrytypes.R2.class));

    r2 = _r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r2,project.
        Entrytypes.R2.class));

  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R1(final project.Entrytypes.R2 _r2) {

    return !(Utils.equals(_r2.x, -1L));
  }
}
```

C.42.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R2(x);

  public R2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R2)) {
      return false;
    }

    project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 copy() {

    return new project.Entrytypes.R2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R2(final Number _x) {

    return !(Utils.equals(_x, -2L));
  }
}
```

C.42.4 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Object r1 = new project.Entrytypes.R1(new project.Entrytypes.R2(5L));
    //@ assert (Utils.is_(r1,project.Entrytypes.R1.class) || Utils.is_nat(r1
        ));

    Number atomicTmp_1 = -1L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 1L;
    //@ assert Utils.is_int(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 apply_1 = null;
      if (r1 instanceof project.Entrytypes.R1) {
        apply_1 = ((project.Entrytypes.R1) r1).get_r2();
      } else {
        throw new RuntimeException("Missing␣member:␣r2");
      }

      project.Entrytypes.R2 stateDes_1 = apply_1;
      //@ assert stateDes_1 != null;

      stateDes_1.set_x(atomicTmp_1);

      project.Entrytypes.R2 apply_2 = null;
      if (r1 instanceof project.Entrytypes.R1) {
        apply_2 = ((project.Entrytypes.R1) r1).get_r2();
      } else {
        throw new RuntimeException("Missing␣member:␣r2");
      }

      project.Entrytypes.R2 stateDes_2 = apply_2;
      //@ assert stateDes_2 != null;

      stateDes_2.set_x(atomicTmp_2);

      //@ set invChecksOn = true;

      //@ assert \invariant_for(stateDes_1);

      //@ assert (Utils.is_(r1,project.Entrytypes.R1.class) || Utils.is_nat(
          r1));

      //@ assert r1 instanceof project.Entrytypes.R1 ==> \invariant_for(((
          project.Entrytypes.R1) r1));

      //@ assert \invariant_for(stateDes_2);

    } /* End of atomic statement */

    IO.println(
        "\\invariant_for␣is␣not␣implemented␣in␣OpenJML␣RAC␣"
```

```
            + "so␣the␣\\invariant_for␣check␣will␣not␣detect␣the␣invariant␣
                violation");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.42.5 The OpenJML runtime assertion checker output

```
"\invariant_for is not implemented in OpenJML RAC so the \invariant_for check
   will not detect the invariant violation"
```

C.43 SetEvenNamedTypeInv.vdmsl

C.43.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

SetEven = set of Even;
Even = nat
inv e == e mod 2 = 0;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : SetEven = {2, 4, 6} in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let xs : SetEven = {2},
     ys : set of nat = {1},
     - : SetEven = xs union ys
 in
   skip;
 IO`println("After␣illegal␣use");
 return 0;
);

end Entry
```

C.43.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMSet ignorePattern_1 = SetUtil.set(2L, 4L, 6L);
      //@ assert ((V2J.isSet(ignorePattern_1) && (\forall int i; 0 <= i && i < V2J.size(ignorePattern_1); (Utils.is_nat(V2J.get(ignorePattern_1,i)) && inv_Entry_Even(V2J.get(ignorePattern_1,i))))) && inv_Entry_SetEven(ignorePattern_1));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMSet xs = SetUtil.set(2L);
      //@ assert ((V2J.isSet(xs) && (\forall int i; 0 <= i && i < V2J.size(xs); (Utils.is_nat(V2J.get(xs,i)) && inv_Entry_Even(V2J.get(xs,i))))) && inv_Entry_SetEven(xs));

      final VDMSet ys = SetUtil.set(1L);
      //@ assert (V2J.isSet(ys) && (\forall int i; 0 <= i && i < V2J.size(ys); Utils.is_nat(V2J.get(ys,i))));

      final VDMSet ignorePattern_2 = SetUtil.union(Utils.copy(xs), Utils.copy(ys));
      //@ assert ((V2J.isSet(ignorePattern_2) && (\forall int i; 0 <= i && i < V2J.size(ignorePattern_2); (Utils.is_nat(V2J.get(ignorePattern_2,i)) && inv_Entry_Even(V2J.get(ignorePattern_2,i))))) && inv_Entry_SetEven(ignorePattern_2));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_SetEven(final Object check_elem) {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Even(final Object check_e) {

    Number e = ((Number) check_e);

    return Utils.equals(Utils.mod(e.longValue(), 2L), 0L);
  }
}
```

C.43.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:35: JML assertion is false
      //@ assert ((V2J.isSet(ignorePattern_2) && (\forall int i; 0 <= i && i <
           V2J.size(ignorePattern_2); (Utils.is_nat(V2J.get(ignorePattern_2,i))
           && inv_Entry_Even(V2J.get(ignorePattern_2,i))))) &&
          inv_Entry_SetEven(ignorePattern_2));
           ^
"After illegal use"
```

C.44 SetOfNat.vdmsl

C.44.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : set of nat = {2,4,6} in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : set of nat = setOfNat() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);
```

```

functions

setOfNat :  () -> [set of nat]
setOfNat () == nil;

end Entry
```

C.44.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMSet ignorePattern_1 = SetUtil.set(2L, 4L, 6L);
      //@ assert (V2J.isSet(ignorePattern_1) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_1); Utils.is_nat(V2J.get(ignorePattern_1,
          i))));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMSet ignorePattern_2 = setOfNat();
      //@ assert (V2J.isSet(ignorePattern_2) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(ignorePattern_2,
          i))));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static VDMSet setOfNat() {

    VDMSet ret_1 = null;
    //@ assert ((ret_1 == null) || (V2J.isSet(ret_1) && (\forall int i; 0 <=
         i && i < V2J.size(ret_1); Utils.is_nat(V2J.get(ret_1,i)))));

    return Utils.copy(ret_1);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.44.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isSet(ignorePattern_2) && (\forall int i; 0 <= i && i <
          V2J.size(ignorePattern_2); Utils.is_nat(V2J.get(ignorePattern_2,i))))
          ;
           ^
"After illegal use"
```

C.45 SetPassNil.vdmsl

C.45.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 (dcl r : set of bool := idSet({true, false}); skip);
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 (
   dcl xs : [set of bool] := nil;
   dcl r : set of bool := idSet(xs);
   skip;
 );
 IO`println("After␣illegal␣use");
 return 0;
);

functions

idSet :  set of bool -> set of bool
idSet (xs) ==
 xs;

```

```
end Entry
```

C.45.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      VDMSet r = idSet(SetUtil.set(true, false));
      //@ assert (V2J.isSet(r) && (\forall int i; 0 <= i && i < V2J.size(r);
           Utils.is_bool(V2J.get(r,i))));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      VDMSet xs = null;
      //@ assert ((xs == null) || (V2J.isSet(xs) && (\forall int i; 0 <= i
          && i < V2J.size(xs); Utils.is_bool(V2J.get(xs,i)))));

      VDMSet r = idSet(Utils.copy(xs));
      //@ assert (V2J.isSet(r) && (\forall int i; 0 <= i && i < V2J.size(r);
           Utils.is_bool(V2J.get(r,i))));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static VDMSet idSet(final VDMSet xs) {

    //@ assert (V2J.isSet(xs) && (\forall int i; 0 <= i && i < V2J.size(xs);
         Utils.is_bool(V2J.get(xs,i))));

    VDMSet ret_1 = Utils.copy(xs);
    //@ assert (V2J.isSet(ret_1) && (\forall int i; 0 <= i && i < V2J.size(
        ret_1); Utils.is_bool(V2J.get(ret_1,i))));

    return Utils.copy(ret_1);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.45.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:44: JML assertion is false
    //@ assert (V2J.isSet(xs) && (\forall int i; 0 <= i && i < V2J.size(xs);
        Utils.is_bool(V2J.get(xs,i))));
        ^
Entry.java:47: JML assertion is false
    //@ assert (V2J.isSet(ret_1) && (\forall int i; 0 <= i && i < V2J.size(
        ret_1); Utils.is_bool(V2J.get(ret_1,i))));
        ^
Entry.java:32: JML assertion is false
      //@ assert (V2J.isSet(r) && (\forall int i; 0 <= i && i < V2J.size(r);
          Utils.is_bool(V2J.get(r,i))));
          ^
"After illegal use"
```

C.46 RecTypesUnion.vdmsl

C.46.1 The VDM-SL model

```
module Entry

imports from IO all
exports all

definitions

types
R1 :: x : int
inv r1 == r1.x > 0;

R2 :: x : int
inv r2 == r2.x > 0;

operations

Run : () ==> ?
Run () ==
(dcl r : R1 | R2 := mk_R1(1);

 IO`println("Before␣valid␣use");
 r.x := 5;
 IO`println("After␣valid␣use");

```

```
  IO`println("Before illegal use");
  r.x := -5;
  IO`println("After illegal use");

  return 0;
 )

end Entry
```

C.46.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(x);

  public R1(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R1)) {
      return false;
    }

    project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R1 copy() {

    return new project.Entrytypes.R1(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R1" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_1 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_1));

    return ret_1;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R1(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.46.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R2(x);

  public R2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R2)) {
      return false;
    }

    project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 copy() {

    return new project.Entrytypes.R2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R2(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.46.4 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Object r = new project.Entrytypes.R1(1L);
    //@ assert (Utils.is_(r,project.Entrytypes.R1.class) || Utils.is_(r,
        project.Entrytypes.R2.class));

    IO.println("Before␣valid␣use");
    if (r instanceof project.Entrytypes.R1) {
      //@ assert r != null;

      ((project.Entrytypes.R1) r).set_x(5L);

    } else if (r instanceof project.Entrytypes.R2) {
      //@ assert r != null;

      ((project.Entrytypes.R2) r).set_x(5L);

    } else {
      throw new RuntimeException("Missing␣member:␣x");
    }

    IO.println("After␣valid␣use");
    IO.println("Before␣illegal␣use");
    if (r instanceof project.Entrytypes.R1) {
      //@ assert r != null;

      ((project.Entrytypes.R1) r).set_x(-5L);

    } else if (r instanceof project.Entrytypes.R2) {
      //@ assert r != null;

      ((project.Entrytypes.R2) r).set_x(-5L);

    } else {
      throw new RuntimeException("Missing␣member:␣x");
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.46.5 The OpenJML runtime assertion checker output

```
"Before valid use"
"After valid use"
"Before illegal use"
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/RecTypesUnion/
    project/Entrytypes/R1.java:62: JML invariant is false on leaving method
    project.Entrytypes.R1.set_x(java.lang.Number)
  public void set_x(final Number _x) {
              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/RecTypesUnion/
    project/Entrytypes/R1.java:12: Associated declaration: /home/peter/git-
    repos/ovt/core/codegen/vdm2jml/target/jml/code/RecTypesUnion/project/
    Entrytypes/R1.java:62:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(x);
                        ^
"After illegal use"
```

C.47 OptionalBasicUnion.vdmsl

C.47.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 (
   dcl a : nat1 | char | bool := true;
   a := 1;
   a := 'a';
   a := true;
 );
 (
   dcl b : [nat1 | char] | bool := true;
   b := nil;
 );
 IO`println("After␣legal␣use");
```

```
  IO`println("Before␣illegal␣use");
  (
     dcl a : nat1 | char | bool := charNil();
     skip;
  );
  IO`println("After␣illegal␣use");
  return 0;
);

functions

charNil :  () -> [char]
charNil () == nil;

end Entry
```

C.47.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      Object a = true;
      //@ assert (Utils.is_bool(a) || Utils.is_char(a) || Utils.is_nat1(a));

      a = 1L;
      //@ assert (Utils.is_bool(a) || Utils.is_char(a) || Utils.is_nat1(a));

      a = 'a';
      //@ assert (Utils.is_bool(a) || Utils.is_char(a) || Utils.is_nat1(a));

      a = true;
      //@ assert (Utils.is_bool(a) || Utils.is_char(a) || Utils.is_nat1(a));

    }

    {
      Object b = true;
      //@ assert (((b == null) || Utils.is_char(b) || Utils.is_nat1(b)) ||
          Utils.is_bool(b));

      b = null;
      //@ assert (((b == null) || Utils.is_char(b) || Utils.is_nat1(b)) ||
          Utils.is_bool(b));

    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      Object a = charNil();
      //@ assert (Utils.is_bool(a) || Utils.is_char(a) || Utils.is_nat1(a));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static Character charNil() {

    Character ret_1 = null;
    //@ assert ((ret_1 == null) || Utils.is_char(ret_1));

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.47.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:46: JML assertion is false
      //@ assert (Utils.is_bool(a) || Utils.is_char(a) || Utils.is_nat1(a));
          ^
"After illegal use"
```

C.48 RecWithRecFieldUpdate.vdmsl

C.48.1 The VDM-SL model

```
module Entry

imports from IO all
exports all

definitions

types
A1 :: f : A2
inv a1 == a1.f.x > 0;
```

```

A2 :: x : int
inv a2 == a2.x > 0;

B1 :: f : B2
inv b1 == b1.f.x > 0;

B2 :: x : int
inv b2 == b2.x > 0;


operations

Run : () ==> ?
Run () ==
(dcl r : A1 | B1 := mk_A1(mk_A2(1));

 IO`println("Before␣valid␣use");
 r.f.x := 5;
 IO`println("After␣valid␣use");

 IO`println("Before␣illegal␣use");
 r.f.x := -5;
 IO`println("After␣illegal␣use");

 return 0;
)

end Entry
```

C.48.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class B2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_B2(x);

  public B2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.B2)) {
      return false;
    }

    project.Entrytypes.B2 other = ((project.Entrytypes.B2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.B2 copy() {

    return new project.Entrytypes.B2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`B2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_4 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_4));

    return ret_4;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_B2(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.48.3 The generated Java/JML

```java
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class A1 implements Record {
  public project.Entrytypes.A2 f;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A1(f);

  public A1(final project.Entrytypes.A2 _f) {

    //@ assert Utils.is_(_f,project.Entrytypes.A2.class);

    f = _f != null ? Utils.copy(_f) : null;
    //@ assert Utils.is_(f,project.Entrytypes.A2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.A1)) {
      return false;
    }

    project.Entrytypes.A1 other = ((project.Entrytypes.A1) obj);

    return Utils.equals(f, other.f);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(f);
  }
  /*@ pure @*/

  public project.Entrytypes.A1 copy() {

    return new project.Entrytypes.A1(f);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`A1" + Utils.formatFields(f);
  }
  /*@ pure @*/

  public project.Entrytypes.A2 get_f() {

    project.Entrytypes.A2 ret_1 = f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_1,project.
        Entrytypes.A2.class));

    return ret_1;
  }

  public void set_f(final project.Entrytypes.A2 _f) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_f,project.
        Entrytypes.A2.class));

    f = _f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(f,project.Entrytypes
        .A2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {

    return _f.x.longValue() > 0L;
  }
}
```

C.48.4 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Object r = new project.Entrytypes.A1(new project.Entrytypes.A2(1L));
    //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,
        project.Entrytypes.B1.class));

    IO.println("Before␣valid␣use");
    Object apply_1 = null;
    if (r instanceof project.Entrytypes.A1) {
      apply_1 = ((project.Entrytypes.A1) r).get_f();
      //@ assert (Utils.is_(apply_1,project.Entrytypes.A2.class) || Utils.
          is_(apply_1,project.Entrytypes.B2.class));

    } else if (r instanceof project.Entrytypes.B1) {
      apply_1 = ((project.Entrytypes.B1) r).get_f();
      //@ assert (Utils.is_(apply_1,project.Entrytypes.A2.class) || Utils.is_(apply_1,project.Entrytypes.B2.class));

    } else {
      throw new RuntimeException("Missing␣member:␣f");
    }

    Object stateDes_1 = apply_1;
    if (stateDes_1 instanceof project.Entrytypes.A2) {
      //@ assert stateDes_1 != null;

      ((project.Entrytypes.A2) stateDes_1).set_x(5L);
      //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,project.Entrytypes.B1.class));

      //@ assert r instanceof project.Entrytypes.B1 ==> ((project.Entrytypes.B1) r).valid();

      //@ assert r instanceof project.Entrytypes.A1 ==> ((project.Entrytypes.A1) r).valid();

    } else if (stateDes_1 instanceof project.Entrytypes.B2) {
      //@ assert stateDes_1 != null;

      ((project.Entrytypes.B2) stateDes_1).set_x(5L);
      //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,project.Entrytypes.B1.class));

      //@ assert r instanceof project.Entrytypes.B1 ==> ((project.Entrytypes.B1) r).valid();

      //@ assert r instanceof project.Entrytypes.A1 ==> ((project.Entrytypes.A1) r).valid();

    } else {
      throw new RuntimeException("Missing␣member:␣x");
    }

    IO.println("After␣valid␣use");
    IO.println("Before␣illegal␣use");
    Object apply_2 = null;
    if (r instanceof project.Entrytypes.A1) {
      apply_2 = ((project.Entrytypes.A1) r).get_f();
      //@ assert (Utils.is_(apply_2,project.Entrytypes.A2.class) || Utils.is_(apply_2,project.Entrytypes.B2.class));

    } else if (r instanceof project.Entrytypes.B1) {
      apply_2 = ((project.Entrytypes.B1) r).get_f();
      //@ assert (Utils.is_(apply_2,project.Entrytypes.A2.class) || Utils.is_(apply_2,project.Entrytypes.B2.class));

    } else {
      throw new RuntimeException("Missing␣member:␣f");
    }

    Object stateDes_2 = apply_2;
    if (stateDes_2 instanceof project.Entrytypes.A2) {
      //@ assert stateDes_2 != null;

      ((project.Entrytypes.A2) stateDes_2).set_x(-5L);
      //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,
         project.Entrytypes.B1.class));

      //@ assert r instanceof project.Entrytypes.B1 ==> ((project.Entrytypes
         .B1) r).valid();

      //@ assert r instanceof project.Entrytypes.A1 ==> ((project.Entrytypes
         .A1) r).valid();

    } else if (stateDes_2 instanceof project.Entrytypes.B2) {
      //@ assert stateDes_2 != null;

      ((project.Entrytypes.B2) stateDes_2).set_x(-5L);
      //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,
         project.Entrytypes.B1.class));

      //@ assert r instanceof project.Entrytypes.B1 ==> ((project.Entrytypes
         .B1) r).valid();

      //@ assert r instanceof project.Entrytypes.A1 ==> ((project.Entrytypes
         .A1) r).valid();

    } else {
      throw new RuntimeException("Missing␣member:␣x");
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.48.5 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class B1 implements Record {
  public project.Entrytypes.B2 f;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_B1(f);

  public B1(final project.Entrytypes.B2 _f) {

    //@ assert Utils.is_(_f,project.Entrytypes.B2.class);

    f = _f != null ? Utils.copy(_f) : null;
    //@ assert Utils.is_(f,project.Entrytypes.B2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.B1)) {
      return false;
    }

    project.Entrytypes.B1 other = ((project.Entrytypes.B1) obj);

    return Utils.equals(f, other.f);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(f);
  }
  /*@ pure @*/

  public project.Entrytypes.B1 copy() {

    return new project.Entrytypes.B1(f);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`B1" + Utils.formatFields(f);
  }
  /*@ pure @*/

  public project.Entrytypes.B2 get_f() {

    project.Entrytypes.B2 ret_3 = f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_3,project.
        Entrytypes.B2.class));

    return ret_3;
  }

  public void set_f(final project.Entrytypes.B2 _f) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_f,project.
        Entrytypes.B2.class));

    f = _f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(f,project.Entrytypes
        .B2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_B1(final project.Entrytypes.B2 _f) {

    return _f.x.longValue() > 0L;
  }
}
```

C.48.6 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class A2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);

  public A2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.A2)) {
      return false;
    }

    project.Entrytypes.A2 other = ((project.Entrytypes.A2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.A2 copy() {

    return new project.Entrytypes.A2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`A2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_A2(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.48.7 The OpenJML runtime assertion checker output

```
"Before valid use"
"After valid use"
"Before illegal use"
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A2.java:62: JML invariant is false
     on leaving method project.Entrytypes.A2.set_x(java.lang.Number)
  public void set_x(final Number _x) {
              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A2.java:12: Associated declaration
    : /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A2.java:62:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:79: JML caller invariant
    is false on leaving calling method (Parameter: _f, Caller: project.
    Entrytypes.A1.inv_A1(project.Entrytypes.A2), Callee: java.lang.Number.
    longValue())
  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {
                                                            ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A2.java:12: Associated declaration
    : /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:79: JML invariant is false
     on leaving method project.Entrytypes.A1.inv_A1(project.Entrytypes.A2) (
    parameter _f)
  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {
                                                            ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A2.java:12: Associated declaration
    : /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:79: JML caller invariant
    is false on leaving calling method (Parameter: _f, Caller: project.
    Entrytypes.A1.inv_A1(project.Entrytypes.A2), Callee: java.lang.Number.
    longValue())
  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {
                                                            ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A2.java:12: Associated declaration
    : /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:79: JML invariant is false
     on leaving method project.Entrytypes.A1.inv_A1(project.Entrytypes.A2) (
    parameter _f)
  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {
                                                            ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A2.java:12: Associated declaration
    : /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:72: JML invariant is false
     on leaving method project.Entrytypes.A1.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:12: Associated declaration
    : /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A1.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A1(f);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldUpdate/project/Entrytypes/A2.java:12: JML invariant is false
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                      ^
```

```
"After illegal use"
```

C.49 RecWithRecFieldAtomicViolation.vdmsl

C.49.1 The VDM-SL model

```
module Entry

imports from IO all
exports all

definitions

types
A1 :: f : A2
inv a1 == a1.f.x > 0;

A2 :: x : int
inv a2 == a2.x > 0;

B1 :: f : B2
inv b1 == b1.f.x > 0;

B2 :: x : int
inv b2 == b2.x > 0;


operations

Run : () ==> ?
Run () ==
(dcl r : A1 | B1 := mk_A1(mk_A2(1));

 IO`println("Before␣valid␣use");
 atomic
 (
   r.f.x := -5;
   r.f.x := 5;
 );
 IO`println("After␣valid␣use");

 IO`println("Before␣illegal␣use");
 atomic
 (
   r.f.x := 5;
   r.f.x := -5;
 );
 IO`println("After␣illegal␣use");

 return 0;
)

end Entry
```

C.49.2 The generated Java/JML

```java
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class B2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_B2(x);

  public B2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.B2)) {
      return false;
    }

    project.Entrytypes.B2 other = ((project.Entrytypes.B2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.B2 copy() {

    return new project.Entrytypes.B2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`B2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_4 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_4));

    return ret_4;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_B2(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.49.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class A1 implements Record {
  public project.Entrytypes.A2 f;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A1(f);

  public A1(final project.Entrytypes.A2 _f) {

    //@ assert Utils.is_(_f,project.Entrytypes.A2.class);

    f = _f != null ? Utils.copy(_f) : null;
    //@ assert Utils.is_(f,project.Entrytypes.A2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.A1)) {
      return false;
    }

    project.Entrytypes.A1 other = ((project.Entrytypes.A1) obj);

    return Utils.equals(f, other.f);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(f);
  }
  /*@ pure @*/

  public project.Entrytypes.A1 copy() {

    return new project.Entrytypes.A1(f);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`A1" + Utils.formatFields(f);
  }
  /*@ pure @*/

  public project.Entrytypes.A2 get_f() {

    project.Entrytypes.A2 ret_1 = f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_1,project.
        Entrytypes.A2.class));

    return ret_1;
  }

  public void set_f(final project.Entrytypes.A2 _f) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_f,project.
        Entrytypes.A2.class));

    f = _f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(f,project.Entrytypes
        .A2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {

    return _f.x.longValue() > 0L;
  }
}
```

C.49.4 The generated Java/JML

```java
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Object r = new project.Entrytypes.A1(new project.Entrytypes.A2(1L));
    //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,
        project.Entrytypes.B1.class));

    IO.println("Before␣valid␣use");
    Number atomicTmp_1 = -5L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 5L;
    //@ assert Utils.is_int(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      Object apply_1 = null;
      if (r instanceof project.Entrytypes.A1) {
        apply_1 = ((project.Entrytypes.A1) r).get_f();
      } else if (r instanceof project.Entrytypes.B1) {
        apply_1 = ((project.Entrytypes.B1) r).get_f();
      } else {
        throw new RuntimeException("Missing␣member:␣f");
      }

      Object stateDes_1 = apply_1;
      if (stateDes_1 instanceof project.Entrytypes.A2) {
        //@ assert stateDes_1 != null;

        ((project.Entrytypes.A2) stateDes_1).set_x(atomicTmp_1);

      } else if (stateDes_1 instanceof project.Entrytypes.B2) {
        //@ assert stateDes_1 != null;

        ((project.Entrytypes.B2) stateDes_1).set_x(atomicTmp_1);

      } else {
        throw new RuntimeException("Missing␣member:␣x");
      }

      Object apply_2 = null;
      if (r instanceof project.Entrytypes.A1) {
        apply_2 = ((project.Entrytypes.A1) r).get_f();
      } else if (r instanceof project.Entrytypes.B1) {
        apply_2 = ((project.Entrytypes.B1) r).get_f();
      } else {
        throw new RuntimeException("Missing␣member:␣f");
      }

      Object stateDes_2 = apply_2;
      if (stateDes_2 instanceof project.Entrytypes.A2) {
        //@ assert stateDes_2 != null;

        ((project.Entrytypes.A2) stateDes_2).set_x(atomicTmp_2);

      } else if (stateDes_2 instanceof project.Entrytypes.B2) {
        //@ assert stateDes_2 != null;

        ((project.Entrytypes.B2) stateDes_2).set_x(atomicTmp_2);

      } else {
        throw new RuntimeException("Missing␣member:␣x");
      }

      //@ set invChecksOn = true;

      //@ assert stateDes_1 instanceof project.Entrytypes.A2 ==> ((project.Entrytypes.A2) stateDes_1).valid();

      //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,project.Entrytypes.B1.class));

      //@ assert r instanceof project.Entrytypes.B1 ==> ((project.Entrytypes.B1) r).valid();

      //@ assert r instanceof project.Entrytypes.A1 ==> ((project.Entrytypes.A1) r).valid();

      //@ assert stateDes_1 instanceof project.Entrytypes.B2 ==> ((project.Entrytypes.B2) stateDes_1).valid();

      //@ assert stateDes_2 instanceof project.Entrytypes.A2 ==> ((project.Entrytypes.A2) stateDes_2).valid();

      //@ assert stateDes_2 instanceof project.Entrytypes.B2 ==> ((project.Entrytypes.B2) stateDes_2).valid();

    } /* End of atomic statement */

    IO.println("After␣valid␣use");
    IO.println("Before␣illegal␣use");
    Number atomicTmp_3 = 5L;
    //@ assert Utils.is_int(atomicTmp_3);

    Number atomicTmp_4 = -5L;
    //@ assert Utils.is_int(atomicTmp_4);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      Object apply_3 = null;
      if (r instanceof project.Entrytypes.A1) {
        apply_3 = ((project.Entrytypes.A1) r).get_f();
      } else if (r instanceof project.Entrytypes.B1) {
```

```
      apply_3 = ((project.Entrytypes.B1) r).get_f();
    } else {
      throw new RuntimeException("Missing␣member:␣f");
    }

    Object stateDes_3 = apply_3;
    if (stateDes_3 instanceof project.Entrytypes.A2) {
      //@ assert stateDes_3 != null;

      ((project.Entrytypes.A2) stateDes_3).set_x(atomicTmp_3);

    } else if (stateDes_3 instanceof project.Entrytypes.B2) {
      //@ assert stateDes_3 != null;

      ((project.Entrytypes.B2) stateDes_3).set_x(atomicTmp_3);

    } else {
      throw new RuntimeException("Missing␣member:␣x");
    }

    Object apply_4 = null;
    if (r instanceof project.Entrytypes.A1) {
      apply_4 = ((project.Entrytypes.A1) r).get_f();
    } else if (r instanceof project.Entrytypes.B1) {
      apply_4 = ((project.Entrytypes.B1) r).get_f();
    } else {
      throw new RuntimeException("Missing␣member:␣f");
    }

    Object stateDes_4 = apply_4;
    if (stateDes_4 instanceof project.Entrytypes.A2) {
      //@ assert stateDes_4 != null;

      ((project.Entrytypes.A2) stateDes_4).set_x(atomicTmp_4);

    } else if (stateDes_4 instanceof project.Entrytypes.B2) {
      //@ assert stateDes_4 != null;

      ((project.Entrytypes.B2) stateDes_4).set_x(atomicTmp_4);

    } else {
      throw new RuntimeException("Missing␣member:␣x");
    }

    //@ set invChecksOn = true;

    //@ assert stateDes_3 instanceof project.Entrytypes.A2 ==> ((project.
        Entrytypes.A2) stateDes_3).valid();

    //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,
        project.Entrytypes.B1.class));

    //@ assert r instanceof project.Entrytypes.B1 ==> ((project.Entrytypes
        .B1) r).valid();

    //@ assert r instanceof project.Entrytypes.A1 ==> ((project.Entrytypes
        .A1) r).valid();

    //@ assert stateDes_3 instanceof project.Entrytypes.B2 ==> ((project.
        Entrytypes.B2) stateDes_3).valid();

      //@ assert stateDes_4 instanceof project.Entrytypes.A2 ==> ((project.
         Entrytypes.A2) stateDes_4).valid();

      //@ assert stateDes_4 instanceof project.Entrytypes.B2 ==> ((project.
         Entrytypes.B2) stateDes_4).valid();

    } /* End of atomic statement */

    IO.println("After␣illegal␣use");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.49.5 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class B1 implements Record {
  public project.Entrytypes.B2 f;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_B1(f);

  public B1(final project.Entrytypes.B2 _f) {

    //@ assert Utils.is_(_f,project.Entrytypes.B2.class);

    f = _f != null ? Utils.copy(_f) : null;
    //@ assert Utils.is_(f,project.Entrytypes.B2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.B1)) {
      return false;
    }

    project.Entrytypes.B1 other = ((project.Entrytypes.B1) obj);

    return Utils.equals(f, other.f);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(f);
  }
  /*@ pure @*/

  public project.Entrytypes.B1 copy() {

    return new project.Entrytypes.B1(f);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`B1" + Utils.formatFields(f);
  }
  /*@ pure @*/

  public project.Entrytypes.B2 get_f() {

    project.Entrytypes.B2 ret_3 = f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_3,project.
        Entrytypes.B2.class));

    return ret_3;
  }

  public void set_f(final project.Entrytypes.B2 _f) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_f,project.
        Entrytypes.B2.class));

    f = _f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(f,project.Entrytypes
        .B2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_B1(final project.Entrytypes.B2 _f) {

    return _f.x.longValue() > 0L;
  }
}
```

C.49.6 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

```

```
@SuppressWarnings("all")
//@ nullable_by_default

final public class A2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);

  public A2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.A2)) {
      return false;
    }

    project.Entrytypes.A2 other = ((project.Entrytypes.A2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.A2 copy() {

    return new project.Entrytypes.A2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`A2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_A2(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.49.7 The OpenJML runtime assertion checker output

```
"Before valid use"
"After valid use"
"Before illegal use"
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:72: JML invariant
     is false on leaving method project.Entrytypes.A2.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:79: JML caller
    invariant is false on leaving calling method (Parameter: _f, Caller:
    project.Entrytypes.A1.inv_A1(project.Entrytypes.A2), Callee: java.lang.
    Number.longValue())
  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {
                                                           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:79: JML invariant
     is false on leaving method project.Entrytypes.A1.inv_A1(project.Entrytypes
    .A2) (parameter _f)
  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {
                                                           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
```

```
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:79: JML caller
    invariant is false on leaving calling method (Parameter: _f, Caller:
    project.Entrytypes.A1.inv_A1(project.Entrytypes.A2), Callee: java.lang.
    Number.longValue())
  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:79: JML invariant
     is false on leaving method project.Entrytypes.A1.inv_A1(project.Entrytypes
    .A2) (parameter _f)
  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:72: JML invariant
     is false on leaving method project.Entrytypes.A1.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /RecWithRecFieldAtomicViolation/project/Entrytypes/A1.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A1(f);
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:12: JML invariant
     is false
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:72: JML invariant
     is false on leaving method project.Entrytypes.A2.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /RecWithRecFieldAtomicViolation/project/Entrytypes/A2.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);
                    ^
"After illegal use"
```

C.50 CharUnionEven.vdmsl

C.50.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

Even = nat
inv n == n mod 2 = 0;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : char | Even = charA() in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : char | Even = charNil() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

charA :  () -> char
charA () == 'a';

charNil :  () -> [char]
charNil () == nil;

end Entry
```

C.50.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final Object ignorePattern_1 = charA();
      //@ assert ((Utils.is_nat(ignorePattern_1) && inv_Entry_Even(
          ignorePattern_1)) || Utils.is_char(ignorePattern_1));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final Object ignorePattern_2 = charNil();
      //@ assert ((Utils.is_nat(ignorePattern_2) && inv_Entry_Even(
          ignorePattern_2)) || Utils.is_char(ignorePattern_2));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static Character charA() {

    Character ret_1 = 'a';
    //@ assert Utils.is_char(ret_1);

    return ret_1;
  }
  /*@ pure @*/

  public static Character charNil() {

    Character ret_2 = null;
    //@ assert ((ret_2 == null) || Utils.is_char(ret_2));

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_Even(final Object check_n) {

    Number n = ((Number) check_n);

    return Utils.equals(Utils.mod(n.longValue(), 2L), 0L);
  }
}
```

C.50.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert ((Utils.is_nat(ignorePattern_2) && inv_Entry_Even(
          ignorePattern_2)) || Utils.is_char(ignorePattern_2));
          ^
"After illegal use"
```

C.51 RecWithRecFieldAtomicNoViolation.vdmsl

C.51.1 The VDM-SL model

```
module Entry

imports from IO all
exports all

definitions

types
A1 :: f : A2
inv a1 == a1.f.x > 0;

A2 :: x : int
inv a2 == a2.x > 0;

B1 :: f : B2
inv b1 == b1.f.x > 0;

B2 :: x : int
inv b2 == b2.x > 0;


operations

Run : () ==> ?
Run () ==
(dcl r : A1 | B1 := mk_A1(mk_A2(1));
 atomic
 (
   r.f.x := 5;
 );
 IO`println("Done!␣Expected␣no␣violations");
 return 0;
)

end Entry
```

C.51.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class B2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_B2(x);

  public B2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.B2)) {
      return false;
    }

    project.Entrytypes.B2 other = ((project.Entrytypes.B2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.B2 copy() {

    return new project.Entrytypes.B2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`B2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_4 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_4));

    return ret_4;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_B2(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.51.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class A1 implements Record {
  public project.Entrytypes.A2 f;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A1(f);

  public A1(final project.Entrytypes.A2 _f) {

    //@ assert Utils.is_(_f,project.Entrytypes.A2.class);

    f = _f != null ? Utils.copy(_f) : null;
    //@ assert Utils.is_(f,project.Entrytypes.A2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.A1)) {
      return false;
    }

    project.Entrytypes.A1 other = ((project.Entrytypes.A1) obj);

    return Utils.equals(f, other.f);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(f);
  }
  /*@ pure @*/

  public project.Entrytypes.A1 copy() {

    return new project.Entrytypes.A1(f);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`A1" + Utils.formatFields(f);
  }
  /*@ pure @*/

  public project.Entrytypes.A2 get_f() {

    project.Entrytypes.A2 ret_1 = f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_1,project.
        Entrytypes.A2.class));

    return ret_1;
  }

  public void set_f(final project.Entrytypes.A2 _f) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_f,project.
        Entrytypes.A2.class));

    f = _f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(f,project.Entrytypes
        .A2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_A1(final project.Entrytypes.A2 _f) {

    return _f.x.longValue() > 0L;
  }
}
```

C.51.4 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Object r = new project.Entrytypes.A1(new project.Entrytypes.A2(1L));
    //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,
        project.Entrytypes.B1.class));

    Number atomicTmp_1 = 5L;
    //@ assert Utils.is_int(atomicTmp_1);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      Object apply_1 = null;
      if (r instanceof project.Entrytypes.A1) {
        apply_1 = ((project.Entrytypes.A1) r).get_f();
      } else if (r instanceof project.Entrytypes.B1) {
        apply_1 = ((project.Entrytypes.B1) r).get_f();
      } else {
        throw new RuntimeException("Missing␣member:␣f");
      }

      Object stateDes_1 = apply_1;
      if (stateDes_1 instanceof project.Entrytypes.A2) {
        //@ assert stateDes_1 != null;

        ((project.Entrytypes.A2) stateDes_1).set_x(atomicTmp_1);

      } else if (stateDes_1 instanceof project.Entrytypes.B2) {
        //@ assert stateDes_1 != null;

        ((project.Entrytypes.B2) stateDes_1).set_x(atomicTmp_1);

      } else {
        throw new RuntimeException("Missing␣member:␣x");
      }

      //@ set invChecksOn = true;

      //@ assert stateDes_1 instanceof project.Entrytypes.A2 ==> ((project.
          Entrytypes.A2) stateDes_1).valid();

      //@ assert (Utils.is_(r,project.Entrytypes.A1.class) || Utils.is_(r,
          project.Entrytypes.B1.class));

      //@ assert r instanceof project.Entrytypes.B1 ==> ((project.Entrytypes
          .B1) r).valid();

      //@ assert r instanceof project.Entrytypes.A1 ==> ((project.Entrytypes
          .A1) r).valid();

      //@ assert stateDes_1 instanceof project.Entrytypes.B2 ==> ((project.
          Entrytypes.B2) stateDes_1).valid();

    } /* End of atomic statement */

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.51.5 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class B1 implements Record {
  public project.Entrytypes.B2 f;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_B1(f);

  public B1(final project.Entrytypes.B2 _f) {

    //@ assert Utils.is_(_f,project.Entrytypes.B2.class);

    f = _f != null ? Utils.copy(_f) : null;
    //@ assert Utils.is_(f,project.Entrytypes.B2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.B1)) {
      return false;
    }

    project.Entrytypes.B1 other = ((project.Entrytypes.B1) obj);

    return Utils.equals(f, other.f);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(f);
  }
  /*@ pure @*/

  public project.Entrytypes.B1 copy() {

    return new project.Entrytypes.B1(f);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`B1" + Utils.formatFields(f);
  }
  /*@ pure @*/

  public project.Entrytypes.B2 get_f() {

    project.Entrytypes.B2 ret_3 = f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_3,project.
        Entrytypes.B2.class));

    return ret_3;
  }

  public void set_f(final project.Entrytypes.B2 _f) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_f,project.
        Entrytypes.B2.class));

    f = _f;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(f,project.Entrytypes
        .B2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_B1(final project.Entrytypes.B2 _f) {

    return _f.x.longValue() > 0L;
  }
}
```

C.51.6 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

```

```
final public class A2 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A2(x);

  public A2(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.A2)) {
      return false;
    }

    project.Entrytypes.A2 other = ((project.Entrytypes.A2) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.A2 copy() {

    return new project.Entrytypes.A2(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`A2" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_2 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_A2(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.51.7 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.52 RecInRecInAtomic.vdmsl

C.52.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

types

R1 :: r2 : R2
inv r1 == r1.r2.r3.x <> 1;
R2 :: r3 :  R3
inv r2 == r2.r3.x <> 2;
R3 :: x : int
inv r3 == r3.x <> 3;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣useOk");
 let - = useOk() in skip;
 IO`println("After␣useOk");
 IO`println("Before␣useNotOk");
 let - = useNotOk() in skip;
 IO`println("After␣useNotOk");
 return 0;
);

useOk : () ==> nat
useOk () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(5)));

 atomic
 (
  r1.r2.r3.x := 1;
  r1.r2.r3.x := 5;
 );

 return 0;
);

useNotOk : () ==> nat
useNotOk () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(5)));

 atomic
 (
  r1.r2.r3.x := 1;
 );

 return 0;
);

end Entry
```

C.52.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public project.Entrytypes.R2 r2;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(r2);

  public R1(final project.Entrytypes.R2 _r2) {

    //@ assert Utils.is_(_r2,project.Entrytypes.R2.class);

    r2 = _r2 != null ? Utils.copy(_r2) : null;
    //@ assert Utils.is_(r2,project.Entrytypes.R2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R1)) {
      return false;
    }
```

```java

    project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

    return Utils.equals(r2, other.r2);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R1 copy() {

    return new project.Entrytypes.R1(r2);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R1" + Utils.formatFields(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 get_r2() {

    project.Entrytypes.R2 ret_3 = r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_3,project.
        Entrytypes.R2.class));

    return ret_3;
  }

  public void set_r2(final project.Entrytypes.R2 _r2) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r2,project.
        Entrytypes.R2.class));

    r2 = _r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r2,project.
        Entrytypes.R2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R1(final project.Entrytypes.R2 _r2) {

    return !(Utils.equals(_r2.r3.x, 1L));
  }
}
```

C.52.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public project.Entrytypes.R3 r3;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R2(r3);

  public R2(final project.Entrytypes.R3 _r3) {

    //@ assert Utils.is_(_r3,project.Entrytypes.R3.class);

    r3 = _r3 != null ? Utils.copy(_r3) : null;
    //@ assert Utils.is_(r3,project.Entrytypes.R3.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R2)) {
      return false;
    }

    project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

    return Utils.equals(r3, other.r3);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r3);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 copy() {

    return new project.Entrytypes.R2(r3);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R2" + Utils.formatFields(r3);
  }
  /*@ pure @*/

  public project.Entrytypes.R3 get_r3() {

    project.Entrytypes.R3 ret_4 = r3;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_4,project.
        Entrytypes.R3.class));

    return ret_4;
  }

  public void set_r3(final project.Entrytypes.R3 _r3) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r3,project.
        Entrytypes.R3.class));

    r3 = _r3;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r3,project.
        Entrytypes.R3.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R2(final project.Entrytypes.R3 _r3) {

    return !(Utils.equals(_r3.x, 2L));
  }
}
```

C.52.4 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣useOk");
    {
      final Number ignorePattern_1 = useOk();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣useOk");
    IO.println("Before␣useNotOk");
    {
      final Number ignorePattern_2 = useNotOk();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣useNotOk");
    return 0L;
  }

  public static Number useOk() {

    project.Entrytypes.R1 r1 =
        new project.Entrytypes.R1(new project.Entrytypes.R2(new project.
            Entrytypes.R3(5L)));
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    Number atomicTmp_1 = 1L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 5L;
    //@ assert Utils.is_int(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 stateDes_1 = r1.get_r2();

      project.Entrytypes.R3 stateDes_2 = stateDes_1.get_r3();

      //@ assert stateDes_2 != null;

      stateDes_2.set_x(atomicTmp_1);

      project.Entrytypes.R2 stateDes_3 = r1.get_r2();

      project.Entrytypes.R3 stateDes_4 = stateDes_3.get_r3();

      //@ assert stateDes_4 != null;

      stateDes_4.set_x(atomicTmp_2);

      //@ set invChecksOn = true;

      //@ assert stateDes_2.valid();

      //@ assert Utils.is_(stateDes_1,project.Entrytypes.R2.class);

      //@ assert stateDes_1.valid();

      //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

      //@ assert r1.valid();

      //@ assert stateDes_4.valid();

      //@ assert Utils.is_(stateDes_3,project.Entrytypes.R2.class);

      //@ assert stateDes_3.valid();
```

```java

    } /* End of atomic statement */

    Number ret_1 = 0L;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public static Number useNotOk() {

    project.Entrytypes.R1 r1 =
        new project.Entrytypes.R1(new project.Entrytypes.R2(new project.
            Entrytypes.R3(5L)));
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    Number atomicTmp_3 = 1L;
    //@ assert Utils.is_int(atomicTmp_3);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 stateDes_5 = r1.get_r2();

      project.Entrytypes.R3 stateDes_6 = stateDes_5.get_r3();

      //@ assert stateDes_6 != null;

      stateDes_6.set_x(atomicTmp_3);

      //@ set invChecksOn = true;

      //@ assert stateDes_6.valid();

      //@ assert Utils.is_(stateDes_5,project.Entrytypes.R2.class);

      //@ assert stateDes_5.valid();

      //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

      //@ assert r1.valid();

    } /* End of atomic statement */

    Number ret_2 = 0L;
    //@ assert Utils.is_nat(ret_2);

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.52.5 The generated Java/JML

```java
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R3 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(x);

  public R3(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R3)) {
      return false;
    }

    project.Entrytypes.R3 other = ((project.Entrytypes.R3) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R3 copy() {

    return new project.Entrytypes.R3(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R3" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_5 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_5));

    return ret_5;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R3(final Number _x) {

    return !(Utils.equals(_x, 3L));
  }
}
```

C.52.6 The OpenJML runtime assertion checker output

```
"Before useOk"
"After useOk"
"Before useNotOk"
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInAtomic/project/Entrytypes/R1.java:72: JML invariant is false on
    leaving method project.Entrytypes.R1.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInAtomic/project/Entrytypes/R1.java:12: Associated declaration: /
    home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInAtomic/project/Entrytypes/R1.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(r2);
                       ^
"After useNotOk"
```

C.53 NamedTypeInvUnionTypeRec.vdmsl

C.53.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

```

```
types

R1 :: r2 : R2
inv r1 == r1.r2.t3.r4.x <> 1;

R2 :: t3 :  T3
inv r2 == r2.t3.r4.x <> 2;

T3 = R3 | X
inv t3 == (is_(t3,R3) => t3.r4.x <> 10) and (is_(t3, X) => t3.b);

R3 :: r4 : R4
inv r3 == r3.r4.x <> 3;

R4 :: x : int
inv r4 == r4.x <> 4;

X :: b : bool;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣useOk");
 let - = useOk() in skip;
 IO`println("After␣useOk");
 IO`println("Before␣useNotOk");
 let - = useNotOk() in skip;
 IO`println("After␣useNotOk");
 return 0;
);

useOk : () ==> nat
useOk () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(mk_R4(5))));

 atomic
 (
  r1.r2.t3.r4.x := 10;
  r1.r2.t3.r4.x := 3;
  r1.r2.t3.r4.x := 5;
 );

 return 0;
);

useNotOk : () ==> nat
useNotOk () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(mk_R4(5))));

 atomic
 (
  r1.r2.t3.r4.x := 3;
 );

 return 0;
);
```

```

end Entry
```

C.53.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public project.Entrytypes.R2 r2;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(r2);

  public R1(final project.Entrytypes.R2 _r2) {

    //@ assert Utils.is_(_r2,project.Entrytypes.R2.class);

    r2 = _r2 != null ? Utils.copy(_r2) : null;
    //@ assert Utils.is_(r2,project.Entrytypes.R2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R1)) {
      return false;
    }

    project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

    return Utils.equals(r2, other.r2);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R1 copy() {

    return new project.Entrytypes.R1(r2);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R1" + Utils.formatFields(r2);
  }
  /*@ pure @*/

```

```
public project.Entrytypes.R2 get_r2() {

  project.Entrytypes.R2 ret_3 = r2;
  //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_3,project.
      Entrytypes.R2.class));

  return ret_3;
}

public void set_r2(final project.Entrytypes.R2 _r2) {

  //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r2,project.
      Entrytypes.R2.class));

  r2 = _r2;
  //@ assert project.Entry.invChecksOn ==> (Utils.is_(r2,project.
      Entrytypes.R2.class));

}
/*@ pure @*/

public Boolean valid() {

  return true;
}
/*@ pure @*/
/*@ helper @*/

public static Boolean inv_R1(final project.Entrytypes.R2 _r2) {

  Object obj_2 = Utils.copy(_r2.t3);
  project.Entrytypes.R4 apply_6 = null;
  if (obj_2 instanceof project.Entrytypes.R3) {
    apply_6 = Utils.copy(((project.Entrytypes.R3) obj_2).r4);
  } else {
    throw new RuntimeException("Missing␣member:␣r4");
  }

  return !(Utils.equals(apply_6.x, 1L));
}

/*@ pure @*/
/*@ helper @*/

public static Boolean inv_Entry_T3(final Object check_t3) {

  Object t3 = ((Object) check_t3);

  Boolean andResult_1 = false;

  Boolean orResult_1 = false;

  if (!(Utils.is_(t3, project.Entrytypes.R3.class))) {
    orResult_1 = true;
  } else {
    project.Entrytypes.R4 apply_9 = null;
    if (t3 instanceof project.Entrytypes.R3) {
      apply_9 = ((project.Entrytypes.R3) t3).get_r4();
    } else {
      throw new RuntimeException("Missing␣member:␣r4");
      }

      orResult_1 = !(Utils.equals(apply_9.get_x(), 10L));
    }

    if (orResult_1) {
      Boolean orResult_2 = false;

      if (!(Utils.is_(t3, project.Entrytypes.X.class))) {
        orResult_2 = true;
      } else {
        Boolean apply_10 = null;
        if (t3 instanceof project.Entrytypes.X) {
          apply_10 = ((project.Entrytypes.X) t3).get_b();
        } else {
          throw new RuntimeException("Missing␣member:␣b");
        }

        orResult_2 = apply_10;
      }

      if (orResult_2) {
        andResult_1 = true;
      }
    }

    return andResult_1;
  }
}
```

C.53.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class X implements Record {
  public Boolean b;

  public X(final Boolean _b) {

    //@ assert Utils.is_bool(_b);

    b = _b;
    //@ assert Utils.is_bool(b);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.X)) {
      return false;
    }

    project.Entrytypes.X other = ((project.Entrytypes.X) obj);

    return Utils.equals(b, other.b);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(b);
  }
  /*@ pure @*/

  public project.Entrytypes.X copy() {

    return new project.Entrytypes.X(b);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`X" + Utils.formatFields(b);
  }
  /*@ pure @*/

  public Boolean get_b() {

    Boolean ret_7 = b;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_bool(ret_7));

    return ret_7;
  }

  public void set_b(final Boolean _b) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_bool(_b));

    b = _b;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_bool(b));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    Object t3 = ((Object) check_t3);

    Boolean andResult_1 = false;

    Boolean orResult_1 = false;

    if (!(Utils.is_(t3, project.Entrytypes.R3.class))) {
      orResult_1 = true;
    } else {
      project.Entrytypes.R4 apply_9 = null;
      if (t3 instanceof project.Entrytypes.R3) {
        apply_9 = ((project.Entrytypes.R3) t3).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      orResult_1 = !(Utils.equals(apply_9.get_x(), 10L));
    }

    if (orResult_1) {
      Boolean orResult_2 = false;

      if (!(Utils.is_(t3, project.Entrytypes.X.class))) {
        orResult_2 = true;
      } else {
        Boolean apply_10 = null;
        if (t3 instanceof project.Entrytypes.X) {
          apply_10 = ((project.Entrytypes.X) t3).get_b();
        } else {
          throw new RuntimeException("Missing␣member:␣b");
        }

        orResult_2 = apply_10;
      }

      if (orResult_2) {
        andResult_1 = true;
      }
    }

    return andResult_1;
  }
}
```

C.53.4 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public Object t3;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R2(t3);

  public R2(final Object _t3) {

    //@ assert ((Utils.is_(_t3,project.Entrytypes.R3.class) || Utils.is_(_t3
        ,project.Entrytypes.X.class)) && inv_Entry_T3(_t3));

    t3 = _t3 != null ? Utils.copy(_t3) : null;
    //@ assert ((Utils.is_(t3,project.Entrytypes.R3.class) || Utils.is_(t3,
        project.Entrytypes.X.class)) && inv_Entry_T3(t3));

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R2)) {
      return false;
    }

    project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

    return Utils.equals(t3, other.t3);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(t3);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 copy() {

    return new project.Entrytypes.R2(t3);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R2" + Utils.formatFields(t3);
  }
  /*@ pure @*/

  public Object get_t3() {

    Object ret_4 = t3;
    //@ assert project.Entry.invChecksOn ==> (((Utils.is_(ret_4,project.
        Entrytypes.R3.class) || Utils.is_(ret_4,project.Entrytypes.X.class))
         && inv_Entry_T3(ret_4)));

    return ret_4;
  }

  public void set_t3(final Object _t3) {

    //@ assert project.Entry.invChecksOn ==> (((Utils.is_(_t3,project.
        Entrytypes.R3.class) || Utils.is_(_t3,project.Entrytypes.X.class))
        && inv_Entry_T3(_t3)));

    t3 = _t3;
    //@ assert project.Entry.invChecksOn ==> (((Utils.is_(t3,project.
        Entrytypes.R3.class) || Utils.is_(t3,project.Entrytypes.X.class)) &&
         inv_Entry_T3(t3)));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R2(final Object _t3) {

    Object obj_4 = _t3;
    project.Entrytypes.R4 apply_8 = null;
    if (obj_4 instanceof project.Entrytypes.R3) {
      apply_8 = Utils.copy(((project.Entrytypes.R3) obj_4).r4);
    } else {
      throw new RuntimeException("Missing␣member:␣r4");
    }

    return !(Utils.equals(apply_8.x, 2L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    Object t3 = ((Object) check_t3);

    Boolean andResult_1 = false;

    Boolean orResult_1 = false;

    if (!(Utils.is_(t3, project.Entrytypes.R3.class))) {
      orResult_1 = true;
    } else {
      project.Entrytypes.R4 apply_9 = null;
      if (t3 instanceof project.Entrytypes.R3) {
        apply_9 = ((project.Entrytypes.R3) t3).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      orResult_1 = !(Utils.equals(apply_9.get_x(), 10L));
    }

    if (orResult_1) {
      Boolean orResult_2 = false;

      if (!(Utils.is_(t3, project.Entrytypes.X.class))) {
        orResult_2 = true;
      } else {
        Boolean apply_10 = null;
        if (t3 instanceof project.Entrytypes.X) {
          apply_10 = ((project.Entrytypes.X) t3).get_b();
        } else {
          throw new RuntimeException("Missing␣member:␣b");
        }

        orResult_2 = apply_10;
      }

      if (orResult_2) {
        andResult_1 = true;
      }
    }

    return andResult_1;
  }
}
```

C.53.5 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R4 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R4(x);

  public R4(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R4)) {
      return false;
    }

    project.Entrytypes.R4 other = ((project.Entrytypes.R4) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R4 copy() {

    return new project.Entrytypes.R4(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry‘R4" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_6 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_6));

    return ret_6;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R4(final Number _x) {

    return !(Utils.equals(_x, 4L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    Object t3 = ((Object) check_t3);

    Boolean andResult_1 = false;

    Boolean orResult_1 = false;

    if (!(Utils.is_(t3, project.Entrytypes.R3.class))) {
      orResult_1 = true;
    } else {
      project.Entrytypes.R4 apply_9 = null;
      if (t3 instanceof project.Entrytypes.R3) {
        apply_9 = ((project.Entrytypes.R3) t3).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      orResult_1 = !(Utils.equals(apply_9.get_x(), 10L));
    }

    if (orResult_1) {
      Boolean orResult_2 = false;

      if (!(Utils.is_(t3, project.Entrytypes.X.class))) {
        orResult_2 = true;
      } else {
        Boolean apply_10 = null;
        if (t3 instanceof project.Entrytypes.X) {
          apply_10 = ((project.Entrytypes.X) t3).get_b();
        } else {
          throw new RuntimeException("Missing␣member:␣b");
        }

        orResult_2 = apply_10;
      }

      if (orResult_2) {
        andResult_1 = true;
      }
    }

    return andResult_1;
  }
}
```

C.53.6 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣useOk");
    {
      final Number ignorePattern_1 = useOk();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣useOk");
    IO.println("Before␣useNotOk");
    {
      final Number ignorePattern_2 = useNotOk();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣useNotOk");
    return 0L;
  }

  public static Number useOk() {

    project.Entrytypes.R1 r1 =
        new project.Entrytypes.R1(
            new project.Entrytypes.R2(new project.Entrytypes.R3(new project.
                Entrytypes.R4(5L))));
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    Number atomicTmp_1 = 10L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 3L;
    //@ assert Utils.is_int(atomicTmp_2);

    Number atomicTmp_3 = 5L;
    //@ assert Utils.is_int(atomicTmp_3);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 stateDes_1 = r1.get_r2();

      Object stateDes_2 = stateDes_1.get_t3();

      project.Entrytypes.R4 apply_1 = null;
      if (stateDes_2 instanceof project.Entrytypes.R3) {
        apply_1 = ((project.Entrytypes.R3) stateDes_2).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      project.Entrytypes.R4 stateDes_3 = apply_1;
      //@ assert stateDes_3 != null;

      stateDes_3.set_x(atomicTmp_1);

      project.Entrytypes.R2 stateDes_4 = r1.get_r2();

      Object stateDes_5 = stateDes_4.get_t3();

      project.Entrytypes.R4 apply_2 = null;
      if (stateDes_5 instanceof project.Entrytypes.R3) {
        apply_2 = ((project.Entrytypes.R3) stateDes_5).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      project.Entrytypes.R4 stateDes_6 = apply_2;
      //@ assert stateDes_6 != null;

      stateDes_6.set_x(atomicTmp_2);

      project.Entrytypes.R2 stateDes_7 = r1.get_r2();

      Object stateDes_8 = stateDes_7.get_t3();

      project.Entrytypes.R4 apply_3 = null;
      if (stateDes_8 instanceof project.Entrytypes.R3) {
        apply_3 = ((project.Entrytypes.R3) stateDes_8).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      project.Entrytypes.R4 stateDes_9 = apply_3;
      //@ assert stateDes_9 != null;

      stateDes_9.set_x(atomicTmp_3);

      //@ set invChecksOn = true;

      //@ assert stateDes_3.valid();

      //@ assert ((Utils.is_(stateDes_2,project.Entrytypes.R3.class) ||
          Utils.is_(stateDes_2,project.Entrytypes.X.class)) && inv_Entry_T3(
          stateDes_2));

      //@ assert stateDes_2 instanceof project.Entrytypes.X ==> ((project.
          Entrytypes.X) stateDes_2).valid();

      //@ assert stateDes_2 instanceof project.Entrytypes.R3 ==> ((project.
          Entrytypes.R3) stateDes_2).valid();

      //@ assert Utils.is_(stateDes_1,project.Entrytypes.R2.class);

      //@ assert stateDes_1.valid();

      //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

      //@ assert r1.valid();

      //@ assert stateDes_6.valid();

      //@ assert ((Utils.is_(stateDes_5,project.Entrytypes.R3.class) ||
          Utils.is_(stateDes_5,project.Entrytypes.X.class)) && inv_Entry_T3(
          stateDes_5));

      //@ assert stateDes_5 instanceof project.Entrytypes.X ==> ((project.
          Entrytypes.X) stateDes_5).valid();

      //@ assert stateDes_5 instanceof project.Entrytypes.R3 ==> ((project.
          Entrytypes.R3) stateDes_5).valid();

      //@ assert Utils.is_(stateDes_4,project.Entrytypes.R2.class);

      //@ assert stateDes_4.valid();

      //@ assert stateDes_9.valid();

      //@ assert ((Utils.is_(stateDes_8,project.Entrytypes.R3.class) ||
          Utils.is_(stateDes_8,project.Entrytypes.X.class)) && inv_Entry_T3(
          stateDes_8));

      //@ assert stateDes_8 instanceof project.Entrytypes.X ==> ((project.Entrytypes.X) stateDes_8).valid();

      //@ assert stateDes_8 instanceof project.Entrytypes.R3 ==> ((project.Entrytypes.R3) stateDes_8).valid();

      //@ assert Utils.is_(stateDes_7,project.Entrytypes.R2.class);

      //@ assert stateDes_7.valid();

    } /* End of atomic statement */

    Number ret_1 = 0L;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public static Number useNotOk() {

    project.Entrytypes.R1 r1 =
        new project.Entrytypes.R1(
            new project.Entrytypes.R2(new project.Entrytypes.R3(new project.Entrytypes.R4(5L))));
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    Number atomicTmp_4 = 3L;
    //@ assert Utils.is_int(atomicTmp_4);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 stateDes_10 = r1.get_r2();

      Object stateDes_11 = stateDes_10.get_t3();

      project.Entrytypes.R4 apply_4 = null;
      if (stateDes_11 instanceof project.Entrytypes.R3) {
        apply_4 = ((project.Entrytypes.R3) stateDes_11).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      project.Entrytypes.R4 stateDes_12 = apply_4;
      //@ assert stateDes_12 != null;

      stateDes_12.set_x(atomicTmp_4);

      //@ set invChecksOn = true;

      //@ assert stateDes_12.valid();

      //@ assert ((Utils.is_(stateDes_11,project.Entrytypes.R3.class) || Utils.is_(stateDes_11,project.Entrytypes.X.class)) && inv_Entry_T3(stateDes_11));

      //@ assert stateDes_11 instanceof project.Entrytypes.X ==> ((project.Entrytypes.X) stateDes_11).valid();

      //@ assert stateDes_11 instanceof project.Entrytypes.R3 ==> ((project.
         Entrytypes.R3) stateDes_11).valid();

      //@ assert Utils.is_(stateDes_10,project.Entrytypes.R2.class);

      //@ assert stateDes_10.valid();

      //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

      //@ assert r1.valid();

    } /* End of atomic statement */

    Number ret_2 = 0L;
    //@ assert Utils.is_nat(ret_2);

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    Object t3 = ((Object) check_t3);

    Boolean andResult_1 = false;

    Boolean orResult_1 = false;

    if (!(Utils.is_(t3, project.Entrytypes.R3.class))) {
      orResult_1 = true;
    } else {
      project.Entrytypes.R4 apply_9 = null;
      if (t3 instanceof project.Entrytypes.R3) {
        apply_9 = ((project.Entrytypes.R3) t3).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      orResult_1 = !(Utils.equals(apply_9.get_x(), 10L));
    }

    if (orResult_1) {
      Boolean orResult_2 = false;

      if (!(Utils.is_(t3, project.Entrytypes.X.class))) {
        orResult_2 = true;
      } else {
        Boolean apply_10 = null;
        if (t3 instanceof project.Entrytypes.X) {
          apply_10 = ((project.Entrytypes.X) t3).get_b();
        } else {
          throw new RuntimeException("Missing␣member:␣b");
        }

        orResult_2 = apply_10;
      }

      if (orResult_2) {
        andResult_1 = true;
      }
    }

    return andResult_1;
  }
}
```

C.53.7 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R3 implements Record {
  public project.Entrytypes.R4 r4;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);

  public R3(final project.Entrytypes.R4 _r4) {

    //@ assert Utils.is_(_r4,project.Entrytypes.R4.class);

    r4 = _r4 != null ? Utils.copy(_r4) : null;
    //@ assert Utils.is_(r4,project.Entrytypes.R4.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R3)) {
      return false;
    }

    project.Entrytypes.R3 other = ((project.Entrytypes.R3) obj);

    return Utils.equals(r4, other.r4);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r4);
  }
  /*@ pure @*/

  public project.Entrytypes.R3 copy() {

    return new project.Entrytypes.R3(r4);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R3" + Utils.formatFields(r4);
  }
  /*@ pure @*/

  public project.Entrytypes.R4 get_r4() {

    project.Entrytypes.R4 ret_5 = r4;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_5,project.
        Entrytypes.R4.class));

    return ret_5;
  }

  public void set_r4(final project.Entrytypes.R4 _r4) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r4,project.
        Entrytypes.R4.class));

    r4 = _r4;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r4,project.
        Entrytypes.R4.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R3(final project.Entrytypes.R4 _r4) {

    return !(Utils.equals(_r4.x, 3L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    Object t3 = ((Object) check_t3);

    Boolean andResult_1 = false;

    Boolean orResult_1 = false;

    if (!(Utils.is_(t3, project.Entrytypes.R3.class))) {
      orResult_1 = true;
    } else {
      project.Entrytypes.R4 apply_9 = null;
      if (t3 instanceof project.Entrytypes.R3) {
        apply_9 = ((project.Entrytypes.R3) t3).get_r4();
      } else {
        throw new RuntimeException("Missing␣member:␣r4");
      }

      orResult_1 = !(Utils.equals(apply_9.get_x(), 10L));
    }

    if (orResult_1) {
      Boolean orResult_2 = false;

      if (!(Utils.is_(t3, project.Entrytypes.X.class))) {
        orResult_2 = true;
      } else {
        Boolean apply_10 = null;
        if (t3 instanceof project.Entrytypes.X) {
          apply_10 = ((project.Entrytypes.X) t3).get_b();
        } else {
          throw new RuntimeException("Missing␣member:␣b");
        }

        orResult_2 = apply_10;
      }

      if (orResult_2) {
        andResult_1 = true;
      }
    }

    return andResult_1;
  }
}
```

C.53.8 The OpenJML runtime assertion checker output

```
"Before useOk"
"After useOk"
"Before useNotOk"
Entry.java:233: JML invariant is false on entering method project.Entrytypes.
    R3.get_r4() from project.Entry.inv_Entry_T3(java.lang.Object)
         apply_9 = ((project.Entrytypes.R3) t3).get_r4();
                                                      ^
Entry.java:233:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                        ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:54: JML invariant is
    false on leaving method project.Entrytypes.R3.get_r4()
  public project.Entrytypes.R4 get_r4() {
                               ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:54:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                        ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:72: JML invariant is
    false on leaving method project.Entrytypes.R3.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:44: JML caller
    invariant is false on leaving calling method (Caller: project.Entrytypes.R3
    .copy(), Callee: project.Entrytypes.R3.R3(project.Entrytypes.R4))
    return new project.Entrytypes.R3(r4);
           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:44:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:14: JML invariant is
    false on leaving method project.Entrytypes.R3.R3(project.Entrytypes.R4)
  public R3(final project.Entrytypes.R4 _r4) {
         ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:14:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:42: JML invariant is
    false on leaving method project.Entrytypes.R3.copy()
  public project.Entrytypes.R3 copy() {
                               ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:42:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:42: JML invariant is
    false on leaving method project.Entrytypes.R3.copy() (for result type)
  public project.Entrytypes.R3 copy() {
                               ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:42:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:44: JML caller
    invariant is false on leaving calling method (Caller: project.Entrytypes.R3
    .copy(), Callee: project.Entrytypes.R3.R3(project.Entrytypes.R4))
    return new project.Entrytypes.R3(r4);
           ^
```

```
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:44:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:14: JML invariant is
    false on leaving method project.Entrytypes.R3.R3(project.Entrytypes.R4)
  public R3(final project.Entrytypes.R4 _r4) {
         ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:14:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:42: JML invariant is
    false on leaving method project.Entrytypes.R3.copy()
  public project.Entrytypes.R3 copy() {
                               ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:42:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:42: JML invariant is
    false on leaving method project.Entrytypes.R3.copy() (for result type)
  public project.Entrytypes.R3 copy() {
                               ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:12: Associated
    declaration: /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code
    /NamedTypeInvUnionTypeRec/project/Entrytypes/R3.java:42:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
"After useNotOk"
```

C.54 RecWithMapOfRec.vdmsl

C.54.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

types

A :: m : map nat to B
inv a == forall i in set dom a.m & a.m(i).x = 2;
```

```
B :: x : nat;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣useOk");
 let - = useOk() in skip;
 IO`println("After␣useOk");
 IO`println("Before␣useNotOk");
 let - = useNotOk() in skip;
 IO`println("After␣useNotOk");
 return 0;
);

useOk : () ==> nat
useOk () ==
(
 dcl a : A := mk_A({|->});
 a.m := a.m munion {1 |-> mk_B(2)};
 return 0;
);

useNotOk : () ==> nat
useNotOk () ==
(
 dcl a : A := mk_A({|->});
 a.m := a.m munion {1 |-> mk_B(1)};
 return 0;
);

end Entry
```

C.54.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣useOk");
    {
      final Number ignorePattern_1 = useOk();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣useOk");
    IO.println("Before␣useNotOk");
    {
      final Number ignorePattern_2 = useNotOk();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣useNotOk");
    return 0L;
  }

  public static Number useOk() {

    project.Entrytypes.A a = new project.Entrytypes.A(MapUtil.map());
    //@ assert Utils.is_(a,project.Entrytypes.A.class);

    //@ assert a != null;

    a.set_m(
        MapUtil.munion(
            Utils.copy(a.get_m()), MapUtil.map(new Maplet(1L, new project.
                Entrytypes.B(2L)))));

    Number ret_1 = 0L;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public static Number useNotOk() {

    project.Entrytypes.A a = new project.Entrytypes.A(MapUtil.map());
    //@ assert Utils.is_(a,project.Entrytypes.A.class);

    //@ assert a != null;

    a.set_m(
        MapUtil.munion(
            Utils.copy(a.get_m()), MapUtil.map(new Maplet(1L, new project.
                Entrytypes.B(1L)))));

    Number ret_2 = 0L;
    //@ assert Utils.is_nat(ret_2);

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.54.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class A implements Record {
  public VDMMap m;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A(m);

  public A(final VDMMap _m) {

    //@ assert (V2J.isMap(_m) && (\forall int i_1; 0 <= i_1 && i_1 < V2J.
        size(_m); Utils.is_nat(V2J.getDom(_m,i_1)) && Utils.is_(V2J.getRng(
        _m,i_1),project.Entrytypes.B.class)));

    m = _m != null ? Utils.copy(_m) : null;
    //@ assert (V2J.isMap(m) && (\forall int i_1; 0 <= i_1 && i_1 < V2J.size
        (m); Utils.is_nat(V2J.getDom(m,i_1)) && Utils.is_(V2J.getRng(m,i_1),
        project.Entrytypes.B.class)));

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.A)) {
      return false;
    }

    project.Entrytypes.A other = ((project.Entrytypes.A) obj);

    return Utils.equals(m, other.m);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(m);
  }
  /*@ pure @*/

  public project.Entrytypes.A copy() {

    return new project.Entrytypes.A(m);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`A" + Utils.formatFields(m);
  }
  /*@ pure @*/

  public VDMMap get_m() {

    VDMMap ret_3 = m;
    //@ assert project.Entry.invChecksOn ==> ((V2J.isMap(ret_3) && (\forall
        int i_1; 0 <= i_1 && i_1 < V2J.size(ret_3); Utils.is_nat(V2J.getDom(
        ret_3,i_1)) && Utils.is_(V2J.getRng(ret_3,i_1),project.Entrytypes.B.
        class))));

    return ret_3;
  }

  public void set_m(final VDMMap _m) {

    //@ assert project.Entry.invChecksOn ==> ((V2J.isMap(_m) && (\forall int
         i_1; 0 <= i_1 && i_1 < V2J.size(_m); Utils.is_nat(V2J.getDom(_m,i_1
        )) && Utils.is_(V2J.getRng(_m,i_1),project.Entrytypes.B.class))));

    m = _m;
    //@ assert project.Entry.invChecksOn ==> ((V2J.isMap(m) && (\forall int
        i_1; 0 <= i_1 && i_1 < V2J.size(m); Utils.is_nat(V2J.getDom(m,i_1))
        && Utils.is_(V2J.getRng(m,i_1),project.Entrytypes.B.class))));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_A(final VDMMap _m) {

    Boolean forAllExpResult_2 = true;
    VDMSet set_2 = MapUtil.dom(_m);
    for (Iterator iterator_2 = set_2.iterator(); iterator_2.hasNext() &&
        forAllExpResult_2; ) {
      Number i = ((Number) iterator_2.next());
      forAllExpResult_2 = Utils.equals(((project.Entrytypes.B) Utils.get(_m,
          i)).x, 2L);
    }
    return forAllExpResult_2;
  }
}
```

C.54.4 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class B implements Record {
  public Number x;

  public B(final Number _x) {

    //@ assert Utils.is_nat(_x);

    x = _x;
    //@ assert Utils.is_nat(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.B)) {
      return false;
    }

    project.Entrytypes.B other = ((project.Entrytypes.B) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.B copy() {

    return new project.Entrytypes.B(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`B" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_4 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_4));

    return ret_4;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
}
```

C.54.5 The OpenJML runtime assertion checker output

```
"Before useOk"
"After useOk"
"Before useNotOk"
A.java:62: JML invariant is false on leaving method project.Entrytypes.A.set_m
    (org.overture.codegen.runtime.VDMMap)
  public void set_m(final VDMMap _m) {
              ^
A.java:12: Associated declaration
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A(m);
                      ^
"After useNotOk"
```

C.55 RecInRecInvViolation.vdmsl

C.55.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

T1 :: t2 : T2
inv t1 == t1.t2.t3.t4.x > 1;

T2 :: t3 : T3
inv t2 == t2.t3.t4.x > 2 and t2.t3.t4.x <> 60;

T3 :: t4 : T4
inv t3 == t3.t4.x > 3;

T4 :: x : nat
inv t4 == t4.x > 4;

operations

useOk : () ==> nat
useOk () ==
(
  dcl t1 : T1 := mk_T1(mk_T2(mk_T3(mk_T4(5))));
  t1.t2.t3.t4.x := 6;
  t1.t2.t3.t4.x := 7;
  return 0;
);

useNotOk : () ==> nat
useNotOk () ==
(
  dcl t1 : T1 := mk_T1(mk_T2(mk_T3(mk_T4(5))));
  t1.t2.t3.t4.x := 60;
  t1.t2.t3.t4.x := 5;
  return 0;
);

Run : () ==> ?
Run () ==
(
 IO`println("Before␣useOk");
 let - = useOk() in skip;
 IO`println("After␣useOk");
 IO`println("Before␣useNotOk");
 let - = useNotOk() in skip;
 IO`println("After␣useNotOk");
 return 0;
);

end Entry
```

C.55.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Number useOk() {

    project.Entrytypes.T1 t1 =
        new project.Entrytypes.T1(
            new project.Entrytypes.T2(new project.Entrytypes.T3(new project.
                Entrytypes.T4(5L))));
    //@ assert Utils.is_(t1,project.Entrytypes.T1.class);

    project.Entrytypes.T2 stateDes_1 = t1.get_t2();

    project.Entrytypes.T3 stateDes_2 = stateDes_1.get_t3();

    project.Entrytypes.T4 stateDes_3 = stateDes_2.get_t4();

    //@ assert stateDes_3 != null;

    stateDes_3.set_x(6L);
    //@ assert Utils.is_(stateDes_2,project.Entrytypes.T3.class);

    //@ assert stateDes_2.valid();

    //@ assert Utils.is_(stateDes_1,project.Entrytypes.T2.class);

    //@ assert stateDes_1.valid();

    //@ assert Utils.is_(t1,project.Entrytypes.T1.class);

    //@ assert t1.valid();

    project.Entrytypes.T2 stateDes_4 = t1.get_t2();

    project.Entrytypes.T3 stateDes_5 = stateDes_4.get_t3();

    project.Entrytypes.T4 stateDes_6 = stateDes_5.get_t4();

    //@ assert stateDes_6 != null;

    stateDes_6.set_x(7L);
    //@ assert Utils.is_(stateDes_5,project.Entrytypes.T3.class);

    //@ assert stateDes_5.valid();

    //@ assert Utils.is_(stateDes_4,project.Entrytypes.T2.class);

    //@ assert stateDes_4.valid();

    //@ assert Utils.is_(t1,project.Entrytypes.T1.class);

    //@ assert t1.valid();

    Number ret_1 = 0L;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public static Number useNotOk() {

    project.Entrytypes.T1 t1 =
        new project.Entrytypes.T1(
            new project.Entrytypes.T2(new project.Entrytypes.T3(new project.
                Entrytypes.T4(5L))));
    //@ assert Utils.is_(t1,project.Entrytypes.T1.class);

    project.Entrytypes.T2 stateDes_7 = t1.get_t2();

    project.Entrytypes.T3 stateDes_8 = stateDes_7.get_t3();

    project.Entrytypes.T4 stateDes_9 = stateDes_8.get_t4();

    //@ assert stateDes_9 != null;

    stateDes_9.set_x(60L);
    //@ assert Utils.is_(stateDes_8,project.Entrytypes.T3.class);

    //@ assert stateDes_8.valid();

    //@ assert Utils.is_(stateDes_7,project.Entrytypes.T2.class);

    //@ assert stateDes_7.valid();

    //@ assert Utils.is_(t1,project.Entrytypes.T1.class);

    //@ assert t1.valid();

    project.Entrytypes.T2 stateDes_10 = t1.get_t2();

    project.Entrytypes.T3 stateDes_11 = stateDes_10.get_t3();

    project.Entrytypes.T4 stateDes_12 = stateDes_11.get_t4();

    //@ assert stateDes_12 != null;

    stateDes_12.set_x(5L);
    //@ assert Utils.is_(stateDes_11,project.Entrytypes.T3.class);

    //@ assert stateDes_11.valid();

    //@ assert Utils.is_(stateDes_10,project.Entrytypes.T2.class);

    //@ assert stateDes_10.valid();

    //@ assert Utils.is_(t1,project.Entrytypes.T1.class);

    //@ assert t1.valid();

    Number ret_2 = 0L;
    //@ assert Utils.is_nat(ret_2);

    return ret_2;
  }

  public static Object Run() {

    IO.println("Before␣useOk");
    {
      final Number ignorePattern_1 = useOk();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣useOk");
    IO.println("Before␣useNotOk");
    {
      final Number ignorePattern_2 = useNotOk();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣useNotOk");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.55.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class T3 implements Record {
  public project.Entrytypes.T4 t4;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T3(t4);

  public T3(final project.Entrytypes.T4 _t4) {

    //@ assert Utils.is_(_t4,project.Entrytypes.T4.class);

    t4 = _t4 != null ? Utils.copy(_t4) : null;
    //@ assert Utils.is_(t4,project.Entrytypes.T4.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.T3)) {
      return false;
    }

    project.Entrytypes.T3 other = ((project.Entrytypes.T3) obj);

    return Utils.equals(t4, other.t4);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(t4);
  }
  /*@ pure @*/

  public project.Entrytypes.T3 copy() {

    return new project.Entrytypes.T3(t4);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`T3" + Utils.formatFields(t4);
  }
  /*@ pure @*/

  public project.Entrytypes.T4 get_t4() {

    project.Entrytypes.T4 ret_5 = t4;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_5,project.
        Entrytypes.T4.class));

    return ret_5;
  }

  public void set_t4(final project.Entrytypes.T4 _t4) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_t4,project.
        Entrytypes.T4.class));

    t4 = _t4;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(t4,project.
        Entrytypes.T4.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_T3(final project.Entrytypes.T4 _t4) {

    return _t4.x.longValue() > 3L;
  }
}
```

C.55.4 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class T4 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T4(x);

  public T4(final Number _x) {

    //@ assert Utils.is_nat(_x);

    x = _x;
    //@ assert Utils.is_nat(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.T4)) {
      return false;
    }

    project.Entrytypes.T4 other = ((project.Entrytypes.T4) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.T4 copy() {

    return new project.Entrytypes.T4(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`T4" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_6 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_6));

    return ret_6;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_T4(final Number _x) {

    return _x.longValue() > 4L;
  }
}
```

C.55.5 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class T1 implements Record {
  public project.Entrytypes.T2 t2;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T1(t2);

  public T1(final project.Entrytypes.T2 _t2) {

    //@ assert Utils.is_(_t2,project.Entrytypes.T2.class);

    t2 = _t2 != null ? Utils.copy(_t2) : null;
    //@ assert Utils.is_(t2,project.Entrytypes.T2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.T1)) {
      return false;
    }

    project.Entrytypes.T1 other = ((project.Entrytypes.T1) obj);

    return Utils.equals(t2, other.t2);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(t2);
  }
  /*@ pure @*/

  public project.Entrytypes.T1 copy() {

    return new project.Entrytypes.T1(t2);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`T1" + Utils.formatFields(t2);
  }
  /*@ pure @*/

  public project.Entrytypes.T2 get_t2() {

    project.Entrytypes.T2 ret_3 = t2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_3,project.
        Entrytypes.T2.class));

    return ret_3;
  }

  public void set_t2(final project.Entrytypes.T2 _t2) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_t2,project.
        Entrytypes.T2.class));

    t2 = _t2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(t2,project.
        Entrytypes.T2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {

    return _t2.t3.t4.x.longValue() > 1L;
  }
}
```

C.55.6 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class T2 implements Record {
  public project.Entrytypes.T3 t3;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);

  public T2(final project.Entrytypes.T3 _t3) {

    //@ assert Utils.is_(_t3,project.Entrytypes.T3.class);

    t3 = _t3 != null ? Utils.copy(_t3) : null;
    //@ assert Utils.is_(t3,project.Entrytypes.T3.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.T2)) {
      return false;
    }

    project.Entrytypes.T2 other = ((project.Entrytypes.T2) obj);

    return Utils.equals(t3, other.t3);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(t3);
  }
  /*@ pure @*/

  public project.Entrytypes.T2 copy() {

    return new project.Entrytypes.T2(t3);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`T2" + Utils.formatFields(t3);
  }
  /*@ pure @*/

  public project.Entrytypes.T3 get_t3() {

    project.Entrytypes.T3 ret_4 = t3;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_4,project.
        Entrytypes.T3.class));

    return ret_4;
  }

  public void set_t3(final project.Entrytypes.T3 _t3) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_t3,project.
        Entrytypes.T3.class));

    t3 = _t3;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(t3,project.
        Entrytypes.T3.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_T2(final project.Entrytypes.T3 _t3) {

    Boolean andResult_2 = false;

    if (_t3.t4.x.longValue() > 2L) {
      if (!(Utils.equals(_t3.t4.x, 60L))) {
        andResult_2 = true;
      }
    }

    return andResult_2;
  }
}
```

C.55.7 The OpenJML runtime assertion checker output

```
"Before useOk"
"After useOk"
"Before useNotOk"
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:72: JML invariant is false
    on leaving method project.Entrytypes.T2.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t2, Caller: project.
    Entrytypes.T1.inv_T1(project.Entrytypes.T2), Callee: java.lang.Number.
    longValue())
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                          ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML invariant is false
    on leaving method project.Entrytypes.T1.inv_T1(project.Entrytypes.T2) (
    parameter _t2)
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                          ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t2, Caller: project.
    Entrytypes.T1.inv_T1(project.Entrytypes.T2), Callee: java.lang.Number.
    longValue())
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                          ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML invariant is false
    on leaving method project.Entrytypes.T1.inv_T1(project.Entrytypes.T2) (
    parameter _t2)
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: JML invariant is false
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t2, Caller: project.
    Entrytypes.T1.inv_T1(project.Entrytypes.T2), Callee: java.lang.Number.
    longValue())
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML invariant is false
    on leaving method project.Entrytypes.T1.inv_T1(project.Entrytypes.T2) (
    parameter _t2)
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t2, Caller: project.
    Entrytypes.T1.inv_T1(project.Entrytypes.T2), Callee: java.lang.Number.
    longValue())
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML invariant is false
    on leaving method project.Entrytypes.T1.inv_T1(project.Entrytypes.T2) (
    parameter _t2)
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t2, Caller: project.
    Entrytypes.T1.inv_T1(project.Entrytypes.T2), Callee: java.lang.Number.
    longValue())
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML invariant is false
    on leaving method project.Entrytypes.T1.inv_T1(project.Entrytypes.T2) (
    parameter _t2)
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:54: JML invariant is false
    on leaving method project.Entrytypes.T1.get_t2() (for result type)
  public project.Entrytypes.T2 get_t2() {
                               ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:54:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: JML invariant is false
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t2, Caller: project.
    Entrytypes.T1.inv_T1(project.Entrytypes.T2), Callee: java.lang.Number.
    longValue())
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                             ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                        ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79: JML invariant is false
    on leaving method project.Entrytypes.T1.inv_T1(project.Entrytypes.T2) (
    parameter _t2)
  public static Boolean inv_T1(final project.Entrytypes.T2 _t2) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T1.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                        ^
Entry.java:100: JML invariant is false on entering method project.Entrytypes.
    T2.get_t3() from project.Entry.useNotOk()
    project.Entrytypes.T3 stateDes_11 = stateDes_10.get_t3();
                                                           ^
Entry.java:100:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                        ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:54: JML invariant is false
    on leaving method project.Entrytypes.T2.get_t3()
  public project.Entrytypes.T3 get_t3() {
                               ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    RecInRecInvViolation/project/Entrytypes/T2.java:54:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_T2(t3);
                        ^
"After useNotOk"
```

C.56 MaskedRecNamedTypeInv.vdmsl

C.56.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

types

R1 :: r2 : R2
inv r1 == r1.r2.t3.r4.x <> 1;

R2 :: t3 :  T3
inv r2 == r2.t3.r4.x <> 2;

T3 = R3
inv t3 == t3.r4.x <> 10;
```

```

R3 :: r4 : R4
inv r3 == r3.r4.x <> 3;

R4 :: x : int
inv r4 == r4.x <> 4;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣useOk");
 let - = useOk() in skip;
 IO`println("After␣useOk");
 IO`println("Before␣useNotOk");
 let - = useNotOk() in skip;
 IO`println("After␣useNotOk");
 return 0;
);

useOk : () ==> nat
useOk () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(mk_R4(5))));

 atomic
 (
  r1.r2.t3.r4.x := 10;
  r1.r2.t3.r4.x := 5;
 );

 return 0;
);

useNotOk : () ==> nat
useNotOk () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(mk_R4(5))));

 atomic
 (
  r1.r2.t3.r4.x := 10;
 );

 return 0;
);

end Entry
```

C.56.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

```

```
@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public project.Entrytypes.R2 r2;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(r2);

  public R1(final project.Entrytypes.R2 _r2) {

    //@ assert Utils.is_(_r2,project.Entrytypes.R2.class);

    r2 = _r2 != null ? Utils.copy(_r2) : null;
    //@ assert Utils.is_(r2,project.Entrytypes.R2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R1)) {
      return false;
    }

    project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

    return Utils.equals(r2, other.r2);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R1 copy() {

    return new project.Entrytypes.R1(r2);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R1" + Utils.formatFields(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 get_r2() {

    project.Entrytypes.R2 ret_3 = r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_3,project.
        Entrytypes.R2.class));

    return ret_3;
  }

  public void set_r2(final project.Entrytypes.R2 _r2) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r2,project.
        Entrytypes.R2.class));

    r2 = _r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r2,project.
        Entrytypes.R2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R1(final project.Entrytypes.R2 _r2) {

    return !(Utils.equals(_r2.t3.r4.x, 1L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.56.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public project.Entrytypes.R3 t3;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R2(t3);

  public R2(final project.Entrytypes.R3 _t3) {

    //@ assert (Utils.is_(_t3,project.Entrytypes.R3.class) && inv_Entry_T3(
        _t3));

    t3 = _t3 != null ? Utils.copy(_t3) : null;
    //@ assert (Utils.is_(t3,project.Entrytypes.R3.class) && inv_Entry_T3(t3
        ));

  }
  /*@ pure @*/

```

```
public boolean equals(final Object obj) {

  if (!(obj instanceof project.Entrytypes.R2)) {
    return false;
  }

  project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

  return Utils.equals(t3, other.t3);
}
/*@ pure @*/

public int hashCode() {

  return Utils.hashCode(t3);
}
/*@ pure @*/

public project.Entrytypes.R2 copy() {

  return new project.Entrytypes.R2(t3);
}
/*@ pure @*/

public String toString() {

  return "mk_Entry`R2" + Utils.formatFields(t3);
}
/*@ pure @*/

public project.Entrytypes.R3 get_t3() {

  project.Entrytypes.R3 ret_4 = t3;
  //@ assert project.Entry.invChecksOn ==> ((Utils.is_(ret_4,project.
      Entrytypes.R3.class) && inv_Entry_T3(ret_4)));

  return ret_4;
}

public void set_t3(final project.Entrytypes.R3 _t3) {

  //@ assert project.Entry.invChecksOn ==> ((Utils.is_(_t3,project.
      Entrytypes.R3.class) && inv_Entry_T3(_t3)));

  t3 = _t3;
  //@ assert project.Entry.invChecksOn ==> ((Utils.is_(t3,project.
      Entrytypes.R3.class) && inv_Entry_T3(t3)));

}
/*@ pure @*/

public Boolean valid() {

  return true;
}
/*@ pure @*/
/*@ helper @*/

public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {

  return !(Utils.equals(_t3.r4.x, 2L));
}

/*@ pure @*/
/*@ helper @*/

public static Boolean inv_Entry_T3(final Object check_t3) {

  project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

  return !(Utils.equals(t3.get_r4().get_x(), 10L));
}
}
```

C.56.4 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R4 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R4(x);

  public R4(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R4)) {
      return false;
    }

    project.Entrytypes.R4 other = ((project.Entrytypes.R4) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R4 copy() {

    return new project.Entrytypes.R4(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R4" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_6 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_6));

    return ret_6;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R4(final Number _x) {

    return !(Utils.equals(_x, 4L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.56.5 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣useOk");
    {
      final Number ignorePattern_1 = useOk();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣useOk");
    IO.println("Before␣useNotOk");
    {
      final Number ignorePattern_2 = useNotOk();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣useNotOk");
    return 0L;
  }

  public static Number useOk() {

    project.Entrytypes.R1 r1 =
        new project.Entrytypes.R1(
            new project.Entrytypes.R2(new project.Entrytypes.R3(new project.
                Entrytypes.R4(5L))));
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    Number atomicTmp_1 = 10L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 5L;
    //@ assert Utils.is_int(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 stateDes_1 = r1.get_r2();

      project.Entrytypes.R3 stateDes_2 = stateDes_1.get_t3();

      project.Entrytypes.R4 stateDes_3 = stateDes_2.get_r4();

      //@ assert stateDes_3 != null;

      stateDes_3.set_x(atomicTmp_1);

    project.Entrytypes.R2 stateDes_4 = r1.get_r2();

    project.Entrytypes.R3 stateDes_5 = stateDes_4.get_t3();

    project.Entrytypes.R4 stateDes_6 = stateDes_5.get_r4();

    //@ assert stateDes_6 != null;

    stateDes_6.set_x(atomicTmp_2);

    //@ set invChecksOn = true;

    //@ assert stateDes_3.valid();

    //@ assert (Utils.is_(stateDes_2,project.Entrytypes.R3.class) &&
        inv_Entry_T3(stateDes_2));

    //@ assert stateDes_2.valid();

    //@ assert Utils.is_(stateDes_1,project.Entrytypes.R2.class);

    //@ assert stateDes_1.valid();

    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    //@ assert r1.valid();

    //@ assert stateDes_6.valid();

    //@ assert (Utils.is_(stateDes_5,project.Entrytypes.R3.class) &&
        inv_Entry_T3(stateDes_5));

    //@ assert stateDes_5.valid();

    //@ assert Utils.is_(stateDes_4,project.Entrytypes.R2.class);

    //@ assert stateDes_4.valid();

  } /* End of atomic statement */

  Number ret_1 = 0L;
  //@ assert Utils.is_nat(ret_1);

  return ret_1;
}

public static Number useNotOk() {

  project.Entrytypes.R1 r1 =
      new project.Entrytypes.R1(
          new project.Entrytypes.R2(new project.Entrytypes.R3(new project.
              Entrytypes.R4(5L))));
  //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

  Number atomicTmp_3 = 10L;
  //@ assert Utils.is_int(atomicTmp_3);

  {
      /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 stateDes_7 = r1.get_r2();

      project.Entrytypes.R3 stateDes_8 = stateDes_7.get_t3();

      project.Entrytypes.R4 stateDes_9 = stateDes_8.get_r4();

      //@ assert stateDes_9 != null;

      stateDes_9.set_x(atomicTmp_3);

      //@ set invChecksOn = true;

      //@ assert stateDes_9.valid();

      //@ assert (Utils.is_(stateDes_8,project.Entrytypes.R3.class) &&
          inv_Entry_T3(stateDes_8));

      //@ assert stateDes_8.valid();

      //@ assert Utils.is_(stateDes_7,project.Entrytypes.R2.class);

      //@ assert stateDes_7.valid();

      //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

      //@ assert r1.valid();

    } /* End of atomic statement */

    Number ret_2 = 0L;
    //@ assert Utils.is_nat(ret_2);

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.56.6 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R3 implements Record {
  public project.Entrytypes.R4 r4;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);

  public R3(final project.Entrytypes.R4 _r4) {

    //@ assert Utils.is_(_r4,project.Entrytypes.R4.class);

    r4 = _r4 != null ? Utils.copy(_r4) : null;
    //@ assert Utils.is_(r4,project.Entrytypes.R4.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R3)) {
      return false;
    }

    project.Entrytypes.R3 other = ((project.Entrytypes.R3) obj);

    return Utils.equals(r4, other.r4);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r4);
  }
  /*@ pure @*/

  public project.Entrytypes.R3 copy() {

    return new project.Entrytypes.R3(r4);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R3" + Utils.formatFields(r4);
  }
  /*@ pure @*/

  public project.Entrytypes.R4 get_r4() {

    project.Entrytypes.R4 ret_5 = r4;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_5,project.
        Entrytypes.R4.class));

    return ret_5;
  }

  public void set_r4(final project.Entrytypes.R4 _r4) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r4,project.
        Entrytypes.R4.class));

    r4 = _r4;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r4,project.
        Entrytypes.R4.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R3(final project.Entrytypes.R4 _r4) {

    return !(Utils.equals(_r4.x, 3L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.56.7 The OpenJML runtime assertion checker output

```
"Before useOk"
"After useOk"
"Before useNotOk"
Entry.java:137: JML assertion is false
      //@ assert (Utils.is_(stateDes_8,project.Entrytypes.R3.class) &&
          inv_Entry_T3(stateDes_8));
          ^
"After useNotOk"
```

C.57 MaskedRecInvViolated.vdmsl

C.57.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

```

```
operations

types

R1 :: r2 : R2
inv r1 == r1.r2.t3.r4.x <> 1;

R2 :: t3 :  T3
inv r2 == r2.t3.r4.x <> 2;

T3 = R3
inv t3 == t3.r4.x <> 10;

R3 :: r4 : R4
inv r3 == r3.r4.x <> 3;

R4 :: x : int
inv r4 == r4.x <> 4;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣useOk");
 let - = useOk() in skip;
 IO`println("After␣useOk");
 IO`println("Before␣useNotOk");
 let - = useNotOk() in skip;
 IO`println("After␣useNotOk");
 return 0;
);

useOk : () ==> nat
useOk () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(mk_R4(5))));

 atomic
 (
  r1.r2.t3.r4.x := 10;
  r1.r2.t3.r4.x := 3;
  r1.r2.t3.r4.x := 5;
 );

 return 0;
);

useNotOk : () ==> nat
useNotOk () ==
(
 dcl r1 : R1 := mk_R1(mk_R2(mk_R3(mk_R4(5))));

 atomic
 (
  r1.r2.t3.r4.x := 3;
 );

 return 0;
);
```

```

end Entry
```

C.57.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R1 implements Record {
  public project.Entrytypes.R2 r2;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R1(r2);

  public R1(final project.Entrytypes.R2 _r2) {

    //@ assert Utils.is_(_r2,project.Entrytypes.R2.class);

    r2 = _r2 != null ? Utils.copy(_r2) : null;
    //@ assert Utils.is_(r2,project.Entrytypes.R2.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R1)) {
      return false;
    }

    project.Entrytypes.R1 other = ((project.Entrytypes.R1) obj);

    return Utils.equals(r2, other.r2);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R1 copy() {

    return new project.Entrytypes.R1(r2);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R1" + Utils.formatFields(r2);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 get_r2() {

    project.Entrytypes.R2 ret_3 = r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_3,project.
        Entrytypes.R2.class));

    return ret_3;
  }

  public void set_r2(final project.Entrytypes.R2 _r2) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r2,project.
        Entrytypes.R2.class));

    r2 = _r2;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r2,project.
        Entrytypes.R2.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R1(final project.Entrytypes.R2 _r2) {

    return !(Utils.equals(_r2.t3.r4.x, 1L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.57.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R2 implements Record {
  public project.Entrytypes.R3 t3;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R2(t3);

  public R2(final project.Entrytypes.R3 _t3) {

    //@ assert (Utils.is_(_t3,project.Entrytypes.R3.class) && inv_Entry_T3(_t3));

    t3 = _t3 != null ? Utils.copy(_t3) : null;
    //@ assert (Utils.is_(t3,project.Entrytypes.R3.class) && inv_Entry_T3(t3));

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R2)) {
      return false;
    }

    project.Entrytypes.R2 other = ((project.Entrytypes.R2) obj);

    return Utils.equals(t3, other.t3);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(t3);
  }
  /*@ pure @*/

  public project.Entrytypes.R2 copy() {

    return new project.Entrytypes.R2(t3);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R2" + Utils.formatFields(t3);
  }
  /*@ pure @*/

  public project.Entrytypes.R3 get_t3() {

    project.Entrytypes.R3 ret_4 = t3;
    //@ assert project.Entry.invChecksOn ==> ((Utils.is_(ret_4,project.Entrytypes.R3.class) && inv_Entry_T3(ret_4)));

    return ret_4;
  }

  public void set_t3(final project.Entrytypes.R3 _t3) {

    //@ assert project.Entry.invChecksOn ==> ((Utils.is_(_t3,project.Entrytypes.R3.class) && inv_Entry_T3(_t3)));

    t3 = _t3;
    //@ assert project.Entry.invChecksOn ==> ((Utils.is_(t3,project.Entrytypes.R3.class) && inv_Entry_T3(t3)));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {

    return !(Utils.equals(_t3.r4.x, 2L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.57.4 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R4 implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R4(x);

  public R4(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R4)) {
      return false;
    }

    project.Entrytypes.R4 other = ((project.Entrytypes.R4) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.R4 copy() {

    return new project.Entrytypes.R4(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R4" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_6 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_6));

    return ret_6;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R4(final Number _x) {

    return !(Utils.equals(_x, 4L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.57.5 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣useOk");
    {
      final Number ignorePattern_1 = useOk();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣useOk");
    IO.println("Before␣useNotOk");
    {
      final Number ignorePattern_2 = useNotOk();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣useNotOk");
    return 0L;
  }

  public static Number useOk() {

    project.Entrytypes.R1 r1 =
        new project.Entrytypes.R1(
            new project.Entrytypes.R2(new project.Entrytypes.R3(new project.
                Entrytypes.R4(5L))));
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    Number atomicTmp_1 = 10L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 3L;
    //@ assert Utils.is_int(atomicTmp_2);

    Number atomicTmp_3 = 5L;
    //@ assert Utils.is_int(atomicTmp_3);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 stateDes_1 = r1.get_r2();

      project.Entrytypes.R3 stateDes_2 = stateDes_1.get_t3();

      project.Entrytypes.R4 stateDes_3 = stateDes_2.get_r4();

      //@ assert stateDes_3 != null;

      stateDes_3.set_x(atomicTmp_1);

      project.Entrytypes.R2 stateDes_4 = r1.get_r2();

      project.Entrytypes.R3 stateDes_5 = stateDes_4.get_t3();

      project.Entrytypes.R4 stateDes_6 = stateDes_5.get_r4();

      //@ assert stateDes_6 != null;

      stateDes_6.set_x(atomicTmp_2);

      project.Entrytypes.R2 stateDes_7 = r1.get_r2();

      project.Entrytypes.R3 stateDes_8 = stateDes_7.get_t3();

      project.Entrytypes.R4 stateDes_9 = stateDes_8.get_r4();

      //@ assert stateDes_9 != null;

      stateDes_9.set_x(atomicTmp_3);

      //@ set invChecksOn = true;

      //@ assert stateDes_3.valid();

      //@ assert (Utils.is_(stateDes_2,project.Entrytypes.R3.class) &&
          inv_Entry_T3(stateDes_2));

      //@ assert stateDes_2.valid();

      //@ assert Utils.is_(stateDes_1,project.Entrytypes.R2.class);

      //@ assert stateDes_1.valid();

      //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

      //@ assert r1.valid();

      //@ assert stateDes_6.valid();

      //@ assert (Utils.is_(stateDes_5,project.Entrytypes.R3.class) &&
          inv_Entry_T3(stateDes_5));

      //@ assert stateDes_5.valid();

      //@ assert Utils.is_(stateDes_4,project.Entrytypes.R2.class);

      //@ assert stateDes_4.valid();

      //@ assert stateDes_9.valid();

      //@ assert (Utils.is_(stateDes_8,project.Entrytypes.R3.class) &&
          inv_Entry_T3(stateDes_8));

      //@ assert stateDes_8.valid();

      //@ assert Utils.is_(stateDes_7,project.Entrytypes.R2.class);

      //@ assert stateDes_7.valid();

    } /* End of atomic statement */

    Number ret_1 = 0L;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public static Number useNotOk() {

    project.Entrytypes.R1 r1 =
        new project.Entrytypes.R1(
            new project.Entrytypes.R2(new project.Entrytypes.R3(new project.
                Entrytypes.R4(5L))));
    //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

    Number atomicTmp_4 = 3L;
    //@ assert Utils.is_int(atomicTmp_4);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      project.Entrytypes.R2 stateDes_10 = r1.get_r2();

      project.Entrytypes.R3 stateDes_11 = stateDes_10.get_t3();

      project.Entrytypes.R4 stateDes_12 = stateDes_11.get_r4();

      //@ assert stateDes_12 != null;

      stateDes_12.set_x(atomicTmp_4);

      //@ set invChecksOn = true;

      //@ assert stateDes_12.valid();

      //@ assert (Utils.is_(stateDes_11,project.Entrytypes.R3.class) &&
          inv_Entry_T3(stateDes_11));

      //@ assert stateDes_11.valid();

      //@ assert Utils.is_(stateDes_10,project.Entrytypes.R2.class);

      //@ assert stateDes_10.valid();

      //@ assert Utils.is_(r1,project.Entrytypes.R1.class);

      //@ assert r1.valid();

    } /* End of atomic statement */

    Number ret_2 = 0L;
    //@ assert Utils.is_nat(ret_2);

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.57.6 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class R3 implements Record {
  public project.Entrytypes.R4 r4;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);

  public R3(final project.Entrytypes.R4 _r4) {

    //@ assert Utils.is_(_r4,project.Entrytypes.R4.class);

    r4 = _r4 != null ? Utils.copy(_r4) : null;
    //@ assert Utils.is_(r4,project.Entrytypes.R4.class);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.R3)) {
      return false;
    }

    project.Entrytypes.R3 other = ((project.Entrytypes.R3) obj);

    return Utils.equals(r4, other.r4);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(r4);
  }
  /*@ pure @*/

  public project.Entrytypes.R3 copy() {

    return new project.Entrytypes.R3(r4);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`R3" + Utils.formatFields(r4);
  }
  /*@ pure @*/

  public project.Entrytypes.R4 get_r4() {

    project.Entrytypes.R4 ret_5 = r4;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(ret_5,project.
        Entrytypes.R4.class));

    return ret_5;
  }

  public void set_r4(final project.Entrytypes.R4 _r4) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_(_r4,project.
        Entrytypes.R4.class));

    r4 = _r4;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_(r4,project.
        Entrytypes.R4.class));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_R3(final project.Entrytypes.R4 _r4) {

    return !(Utils.equals(_r4.x, 3L));
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_T3(final Object check_t3) {

    project.Entrytypes.R3 t3 = ((project.Entrytypes.R3) check_t3);

    return !(Utils.equals(t3.get_r4().get_x(), 10L));
  }
}
```

C.57.7 The OpenJML runtime assertion checker output

```
"Before useOk"
"After useOk"
"Before useNotOk"
Entry.java:192: JML invariant is false on entering method project.Entrytypes.
    R3.get_r4() from project.Entry.inv_Entry_T3(java.lang.Object)
    return !(Utils.equals(t3.get_r4().get_x(), 10L));
                                    ^
Entry.java:192:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:54: JML invariant is false
    on leaving method project.Entrytypes.R3.get_r4()
  public project.Entrytypes.R4 get_r4() {
                               ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:54:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:72: JML invariant is false
    on leaving method project.Entrytypes.R3.valid()
  public Boolean valid() {
                 ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:72:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
```

```
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: JML invariant is false
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
```

```
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                          ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                          ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                          ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                          ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                      ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                    ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                              ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                    ^
```

```
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                           ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
```

```
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                            ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                            ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML caller invariant is
     false on leaving calling method (Parameter: _t3, Caller: project.
    Entrytypes.R2.inv_R2(project.Entrytypes.R3), Callee: org.overture.codegen.
    runtime.Utils.equals(java.lang.Object,java.lang.Object))
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                            ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79: JML invariant is false
    on leaving method project.Entrytypes.R2.inv_R2(project.Entrytypes.R3) (
    parameter _t3)
  public static Boolean inv_R2(final project.Entrytypes.R3 _t3) {
                                                            ^
/home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R3.java:12: Associated declaration:
     /home/peter/git-repos/ovt/core/codegen/vdm2jml/target/jml/code/
    MaskedRecInvViolated/project/Entrytypes/R2.java:79:
  //@ public instance invariant project.Entry.invChecksOn ==> inv_R3(r4);
                     ^
"After useNotOk"
```

C.58 ModifyRecInMap.vdmsl

C.58.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

types

A :: m : map nat to B
inv a == forall i in set dom a.m & a.m(i).x = 2;
B :: x : nat;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣useOk");
 let - = useOk() in skip;
 IO`println("After␣useOk");
 IO`println("Before␣useNotOk");
 let - = useNotOk() in skip;
 IO`println("After␣useNotOk");
 return 0;
);

useOk : () ==> nat
useOk () ==
(
 dcl a : A := mk_A({1 |-> mk_B(2)});
 a.m(1).x := 2;
 return 0;
);

useNotOk : () ==> nat
useNotOk () ==
(
 dcl a : A := mk_A({1 |-> mk_B(2)});
 a.m(1).x := 1;
 return 0;
);

end Entry
```

C.58.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;
```

```

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣useOk");
    {
      final Number ignorePattern_1 = useOk();
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣useOk");
    IO.println("Before␣useNotOk");
    {
      final Number ignorePattern_2 = useNotOk();
      //@ assert Utils.is_nat(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣useNotOk");
    return 0L;
  }

  public static Number useOk() {

    project.Entrytypes.A a =
        new project.Entrytypes.A(MapUtil.map(new Maplet(1L, new project.
            Entrytypes.B(2L))));
    //@ assert Utils.is_(a,project.Entrytypes.A.class);

    VDMMap stateDes_1 = a.get_m();

    project.Entrytypes.B stateDes_2 = ((project.Entrytypes.B) Utils.get(
        stateDes_1, 1L));

    //@ assert stateDes_2 != null;

    stateDes_2.set_x(2L);
    //@ assert (V2J.isMap(stateDes_1) && (\forall int i_1; 0 <= i_1 && i_1 <
         V2J.size(stateDes_1); Utils.is_nat(V2J.getDom(stateDes_1,i_1)) &&
        Utils.is_(V2J.getRng(stateDes_1,i_1),project.Entrytypes.B.class)));

    //@ assert Utils.is_(a,project.Entrytypes.A.class);

    //@ assert a.valid();

    Number ret_1 = 0L;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public static Number useNotOk() {

    project.Entrytypes.A a =
        new project.Entrytypes.A(MapUtil.map(new Maplet(1L, new project.
            Entrytypes.B(2L))));
    //@ assert Utils.is_(a,project.Entrytypes.A.class);

    VDMMap stateDes_3 = a.get_m();

    project.Entrytypes.B stateDes_4 = ((project.Entrytypes.B) Utils.get(
        stateDes_3, 1L));

    //@ assert stateDes_4 != null;

    stateDes_4.set_x(1L);
    //@ assert (V2J.isMap(stateDes_3) && (\forall int i_1; 0 <= i_1 && i_1 <
         V2J.size(stateDes_3); Utils.is_nat(V2J.getDom(stateDes_3,i_1)) &&
        Utils.is_(V2J.getRng(stateDes_3,i_1),project.Entrytypes.B.class)));

    //@ assert Utils.is_(a,project.Entrytypes.A.class);

    //@ assert a.valid();

    Number ret_2 = 0L;
    //@ assert Utils.is_nat(ret_2);

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.58.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class A implements Record {
  public VDMMap m;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A(m);

  public A(final VDMMap _m) {

    //@ assert (V2J.isMap(_m) && (\forall int i_1; 0 <= i_1 && i_1 < V2J.
        size(_m); Utils.is_nat(V2J.getDom(_m,i_1)) && Utils.is_(V2J.getRng(
        _m,i_1),project.Entrytypes.B.class)));

    m = _m != null ? Utils.copy(_m) : null;
    //@ assert (V2J.isMap(m) && (\forall int i_1; 0 <= i_1 && i_1 < V2J.size
        (m); Utils.is_nat(V2J.getDom(m,i_1)) && Utils.is_(V2J.getRng(m,i_1),
        project.Entrytypes.B.class)));

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.A)) {
      return false;
    }

    project.Entrytypes.A other = ((project.Entrytypes.A) obj);

    return Utils.equals(m, other.m);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(m);
  }
  /*@ pure @*/

  public project.Entrytypes.A copy() {

    return new project.Entrytypes.A(m);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`A" + Utils.formatFields(m);
  }
  /*@ pure @*/

  public VDMMap get_m() {

    VDMMap ret_3 = m;
    //@ assert project.Entry.invChecksOn ==> ((V2J.isMap(ret_3) && (\forall
        int i_1; 0 <= i_1 && i_1 < V2J.size(ret_3); Utils.is_nat(V2J.getDom(
        ret_3,i_1)) && Utils.is_(V2J.getRng(ret_3,i_1),project.Entrytypes.B.
        class))));

    return ret_3;
  }

  public void set_m(final VDMMap _m) {

    //@ assert project.Entry.invChecksOn ==> ((V2J.isMap(_m) && (\forall int
         i_1; 0 <= i_1 && i_1 < V2J.size(_m); Utils.is_nat(V2J.getDom(_m,i_1
        )) && Utils.is_(V2J.getRng(_m,i_1),project.Entrytypes.B.class))));

    m = _m;
    //@ assert project.Entry.invChecksOn ==> ((V2J.isMap(m) && (\forall int
        i_1; 0 <= i_1 && i_1 < V2J.size(m); Utils.is_nat(V2J.getDom(m,i_1))
        && Utils.is_(V2J.getRng(m,i_1),project.Entrytypes.B.class))));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_A(final VDMMap _m) {

    Boolean forAllExpResult_2 = true;
    VDMSet set_2 = MapUtil.dom(_m);
    for (Iterator iterator_2 = set_2.iterator(); iterator_2.hasNext() &&
        forAllExpResult_2; ) {
      Number i = ((Number) iterator_2.next());
      forAllExpResult_2 = Utils.equals(((project.Entrytypes.B) Utils.get(_m,
           i)).x, 2L);
    }
    return forAllExpResult_2;
  }
}
```

C.58.4 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class B implements Record {
  public Number x;

  public B(final Number _x) {

    //@ assert Utils.is_nat(_x);

    x = _x;
    //@ assert Utils.is_nat(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.B)) {
      return false;
    }

    project.Entrytypes.B other = ((project.Entrytypes.B) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.B copy() {

    return new project.Entrytypes.B(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`B" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_4 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_4));

    return ret_4;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
}
```

C.58.5 The OpenJML runtime assertion checker output

```
"Before useOk"
"After useOk"
"Before useNotOk"
A.java:72: JML invariant is false on leaving method project.Entrytypes.A.valid
    ()
  public Boolean valid() {
                 ^
A.java:12: Associated declaration
  //@ public instance invariant project.Entry.invChecksOn ==> inv_A(m);
                      ^
"After useNotOk"
```

C.59 Bool.vdmsl

C.59.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> bool
f () ==
let true = true
in
  true;

end Entry
```

C.59.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Boolean ignorePattern_1 = f();
      //@ assert Utils.is_bool(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Boolean f() {

    final Boolean boolPattern_1 = true;
    //@ assert Utils.is_bool(boolPattern_1);

    Boolean success_1 = Utils.equals(boolPattern_1, true);
    //@ assert Utils.is_bool(success_1);

    if (!(success_1)) {
      throw new RuntimeException("Bool␣pattern␣match␣failed");
    }

    Boolean ret_1 = true;
    //@ assert Utils.is_bool(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.59.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.60 Real.vdmsl

C.60.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> real
f () ==
let 1.5 = 1.5
in
  1.5;

end Entry
```

C.60.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f();
      //@ assert Utils.is_real(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number f() {

    final Number realPattern_1 = 1.5;
    //@ assert Utils.is_rat(realPattern_1);

    Boolean success_1 = Utils.equals(realPattern_1, 1.5);
    //@ assert Utils.is_bool(success_1);

    if (!(success_1)) {
      throw new RuntimeException("Real␣pattern␣match␣failed");
    }

    Number ret_1 = 1.5;
    //@ assert Utils.is_real(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.60.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.61 String.vdmsl

C.61.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> seq of char
f () ==
let "a" = "a"
in
  "a";

end Entry
```

C.61.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final String ignorePattern_1 = f();
      //@ assert Utils.is_(ignorePattern_1,String.class);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static String f() {

    final String stringPattern_1 = "a";
    //@ assert Utils.is_(stringPattern_1,String.class);

    Boolean success_1 = Utils.equals(stringPattern_1, "a");
    //@ assert Utils.is_bool(success_1);

    if (!(success_1)) {
      throw new RuntimeException("String␣pattern␣match␣failed");
    }

    String ret_1 = "a";
    //@ assert Utils.is_(ret_1,String.class);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.61.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.62 Quote.vdmsl

C.62.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> <A>
f () ==
let <A> = <A>
in
  <A>;

end Entry
```

C.62.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final project.quotes.AQuote ignorePattern_1 = f();
      //@ assert Utils.is_(ignorePattern_1,project.quotes.AQuote.class);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static project.quotes.AQuote f() {

    final project.quotes.AQuote quotePattern_1 = project.quotes.AQuote.
        getInstance();
    //@ assert Utils.is_(quotePattern_1,project.quotes.AQuote.class);

    Boolean success_1 = Utils.equals(quotePattern_1, project.quotes.AQuote.
        getInstance());
    //@ assert Utils.is_bool(success_1);

    if (!(success_1)) {
      throw new RuntimeException("Quote␣pattern␣match␣failed");
    }

    project.quotes.AQuote ret_1 = project.quotes.AQuote.getInstance();
    //@ assert Utils.is_(ret_1,project.quotes.AQuote.class);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.62.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.63 Int.vdmsl

C.63.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - = f(1) in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  nat -> nat
f (1) ==
 2;

end Entry
```

C.63.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;
```

```

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f(1L);
      //@ assert Utils.is_nat(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number f(final Number intPattern_1) {

    //@ assert Utils.is_nat(intPattern_1);

    Boolean success_1 = Utils.equals(intPattern_1, 1L);
    //@ assert Utils.is_bool(success_1);

    if (!(success_1)) {
      throw new RuntimeException("Integer␣pattern␣match␣failed");
    }

    Number ret_1 = 2L;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.63.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.64 Nil.vdmsl

C.64.1 The VDM-SL model

```
module Entry
```

```

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> [nat]
f () ==
let nil in set {nil}
in
 nil;

end Entry
```

C.64.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Number ignorePattern_1 = f();
      //@ assert ((ignorePattern_1 == null) || Utils.is_nat(ignorePattern_1)
          );

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Number f() {

    Object letBeStExp_1 = null;
    Object nullPattern_1 = null;

    Boolean success_1 = false;
    //@ assert Utils.is_bool(success_1);

    VDMSet set_1 = SetUtil.set(null);
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); true));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() && !(
        success_1); ) {
      nullPattern_1 = ((Object) iterator_1.next());
      success_1 = Utils.equals(nullPattern_1, null);
      //@ assert Utils.is_bool(success_1);

      if (!(success_1)) {
        continue;
      }

      success_1 = true;
      //@ assert Utils.is_bool(success_1);

    }
    if (!(success_1)) {
      throw new RuntimeException("Let␣Be␣St␣found␣no␣applicable␣bindings");
    }

    letBeStExp_1 = null;
    Number ret_1 = ((Number) letBeStExp_1);
    //@ assert ((ret_1 == null) || Utils.is_nat(ret_1));

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.64.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.65 Char.vdmsl

C.65.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

```

```
operations

Run : () ==> ?
Run () ==
(
  let - = f() in skip;
  IO`println("Done!␣Expected␣no␣violations");
  return 0;
);

functions

f :  () -> char
f () ==
let 'a' in set {'a'}
in
 'a';

end Entry
```

C.65.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    {
      final Character ignorePattern_1 = f();
      //@ assert Utils.is_char(ignorePattern_1);

      /* skip */
    }

    IO.println("Done!␣Expected␣no␣violations");
    return 0L;
  }
  /*@ pure @*/

  public static Character f() {

    Character letBeStExp_1 = null;
    Character charPattern_1 = null;

    Boolean success_1 = false;
    //@ assert Utils.is_bool(success_1);

    VDMSet set_1 = SetUtil.set('a');
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_char(V2J.get(set_1,i))));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() && !(
        success_1); ) {
      charPattern_1 = ((Character) iterator_1.next());
      //@ assert Utils.is_char(charPattern_1);

      success_1 = Utils.equals(charPattern_1, 'a');
      //@ assert Utils.is_bool(success_1);

      if (!(success_1)) {
        continue;
      }

      success_1 = true;
      //@ assert Utils.is_bool(success_1);

    }
    if (!(success_1)) {
      throw new RuntimeException("Let␣Be␣St␣found␣no␣applicable␣bindings");
    }

    letBeStExp_1 = 'a';
    //@ assert Utils.is_char(letBeStExp_1);

    Character ret_1 = letBeStExp_1;
    //@ assert Utils.is_char(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.65.3 The OpenJML runtime assertion checker output

```
"Done! Expected no violations"
```

C.66 StateInitViolatesInv.vdmsl

C.66.1 The VDM-SL model

```
module Entry

exports all
definitions

state St of
  x : int
init s == s = mk_St(-5)
```

```
inv s == s.x > 0
end

operations

Run : () ==> ?
Run () ==
  return 1;

end Entry
```

C.66.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class St implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);

  public St(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.St)) {
      return false;
    }

    project.Entrytypes.St other = ((project.Entrytypes.St) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.St copy() {

    return new project.Entrytypes.St(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`St" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_1 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_1));

    return ret_1;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_St(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.66.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ spec_public @*/

  private static project.Entrytypes.St St = new project.Entrytypes.St(-5L);
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    return 1L;
  }

  public String toString() {

    return "Entry{" + "St␣:=␣" + Utils.toString(St) + "}";
  }
}
```

C.66.4 The OpenJML runtime assertion checker output

```
St.java:12: JML invariant is false
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);
                        ^
```

C.67 RecTypeDefInv.vdmsl

C.67.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

types

Rec ::
  x : int
  y : int
inv mk_Rec(x,y) == x > 0 and y > 0;

operations

Run : () ==> ?
Run () ==
(
  recInvOk();
  IO`println("Before␣breaking␣record␣invariant");
  recInvBreak();
  IO`println("After␣breaking␣record␣invariant");
  return 0;
);

recInvOk : () ==> ()
recInvOk () ==
let - = mk_Rec(1,2)
in
  skip;

recInvBreak : () ==> ()
recInvBreak () ==
let - = mk_Rec(1,-2)
```

```
in
  skip;

end Entry
```

C.67.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    recInvOk();
    IO.println("Before␣breaking␣record␣invariant");
    recInvBreak();
    IO.println("After␣breaking␣record␣invariant");
    return 0L;
  }

  public static void recInvOk() {

    final project.Entrytypes.Rec ignorePattern_1 = new project.Entrytypes.
        Rec(1L, 2L);
    //@ assert Utils.is_(ignorePattern_1,project.Entrytypes.Rec.class);

    /* skip */

  }

  public static void recInvBreak() {

    final project.Entrytypes.Rec ignorePattern_2 = new project.Entrytypes.
        Rec(1L, -2L);
    //@ assert Utils.is_(ignorePattern_2,project.Entrytypes.Rec.class);

    /* skip */

  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.67.3 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Rec implements Record {
  public Number x;
  public Number y;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_Rec(x,y);

  public Rec(final Number _x, final Number _y) {

    //@ assert Utils.is_int(_x);

    //@ assert Utils.is_int(_y);

    x = _x;
    //@ assert Utils.is_int(x);

    y = _y;
    //@ assert Utils.is_int(y);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.Rec)) {
      return false;
    }

    project.Entrytypes.Rec other = ((project.Entrytypes.Rec) obj);

    return (Utils.equals(x, other.x)) && (Utils.equals(y, other.y));
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x, y);
  }
  /*@ pure @*/

  public project.Entrytypes.Rec copy() {

    return new project.Entrytypes.Rec(x, y);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`Rec" + Utils.formatFields(x, y);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_1 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_1));

    return ret_1;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Number get_y() {

    Number ret_2 = y;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_2));

    return ret_2;
  }

  public void set_y(final Number _y) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_y));

    y = _y;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(y));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Rec(final Number _x, final Number _y) {

    Boolean success_2 = true;
    Number x = null;
    Number y = null;
    x = _x;
    y = _y;

    if (!(success_2)) {
      throw new RuntimeException("Record␣pattern␣match␣failed");
    }

    Boolean andResult_2 = false;

    if (x.longValue() > 0L) {
      if (y.longValue() > 0L) {
        andResult_2 = true;
      }
    }

    return andResult_2;
  }
}
```

C.67.4 The OpenJML runtime assertion checker output

```
"Before breaking record invariant"
Rec.java:15: JML invariant is false on leaving method project.Entrytypes.Rec.
    Rec(java.lang.Number,java.lang.Number)
  public Rec(final Number _x, final Number _y) {
         ^
Rec.java:13: Associated declaration
  //@ public instance invariant project.Entry.invChecksOn ==> inv_Rec(x,y);
                       ^
"After breaking record invariant"
```

C.68 StateInv.vdmsl

C.68.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

state St of
  x : int
init s == s = mk_St(5)
inv s == s.x > 0
end

operations

Run : () ==> ?
Run () ==
(
  opAtomic();
  IO`println("Before␣breaking␣state␣invariant");
  op();
  IO`println("After␣breaking␣state␣invariant");
  return x;
);

opAtomic : () ==> ()
opAtomic () ==
atomic
(
  x := -1;
  x := 1;
);
```

```

op : () ==> ()
op () ==
(
  x := -10;
  x := 10;
);

end Entry
```

C.68.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class St implements Record {
  public Number x;
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);

  public St(final Number _x) {

    //@ assert Utils.is_int(_x);

    x = _x;
    //@ assert Utils.is_int(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.St)) {
      return false;
    }

    project.Entrytypes.St other = ((project.Entrytypes.St) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.St copy() {

    return new project.Entrytypes.St(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`St" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_1 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_1));

    return ret_1;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_St(final Number _x) {

    return _x.longValue() > 0L;
  }
}
```

C.68.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ spec_public @*/

  private static project.Entrytypes.St St = new project.Entrytypes.St(5L);
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    opAtomic();
    IO.println("Before␣breaking␣state␣invariant");
    op();
    IO.println("After␣breaking␣state␣invariant");
    return St.get_x();
  }

  public static void opAtomic() {

    Number atomicTmp_1 = -1L;
    //@ assert Utils.is_int(atomicTmp_1);

    Number atomicTmp_2 = 1L;
    //@ assert Utils.is_int(atomicTmp_2);

    {
        /* Start of atomic statement */
      //@ set invChecksOn = false;

      //@ assert St != null;

      St.set_x(atomicTmp_1);

      //@ assert St != null;

      St.set_x(atomicTmp_2);

      //@ set invChecksOn = true;

      //@ assert St.valid();

    } /* End of atomic statement */
  }

  public static void op() {

    //@ assert St != null;

    St.set_x(-10L);

    //@ assert St != null;

    St.set_x(10L);
  }

  public String toString() {

    return "Entry{" + "St␣:=␣" + Utils.toString(St) + "}";
  }
}
```

C.68.4 The OpenJML runtime assertion checker output

```
"Before breaking state invariant"
St.java:62: JML invariant is false on leaving method project.Entrytypes.St.
    set_x(java.lang.Number)
  public void set_x(final Number _x) {
```

```
                ^
St.java:12: Associated declaration
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);
                                  ^
Entry.java:62: JML invariant is false on entering method project.Entrytypes.St
    .set_x(java.lang.Number) from project.Entry.op()
    St.set_x(10L);
              ^
St.java:12: Associated declaration
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);
                                  ^
"After breaking state invariant"
```

C.69 PostCond.vdmsl

C.69.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

state St of
  x : nat
init s == s = mk_St(0)
end

functions

f :  nat -> nat
f (a) ==
  if a mod 2 = 0 then a + 2 else a + 1
post RESULT mod 2 = 0;

operations

Run : () ==> ?
Run () ==
let - = opRet(1),
    - = f(3)
in
(
  opVoid();
  IO`println("Before␣breaking␣post␣condition");
  let - = opRet(4) in skip;
  IO`println("After␣breaking␣post␣condition");
  return 0;
);

opVoid : () ==> ()
opVoid () ==
  x := x + 1
post x = x~+1;

opRet : nat ==> nat
opRet (a) ==
(
  x := x + 1;
  return x;
)
post x = x~+1 and RESULT = a;

end Entry
```

C.69.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class St implements Record {
  public Number x;

  public St(final Number _x) {

    //@ assert Utils.is_nat(_x);

    x = _x;
    //@ assert Utils.is_nat(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.St)) {
      return false;
    }

    project.Entrytypes.St other = ((project.Entrytypes.St) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.St copy() {

    return new project.Entrytypes.St(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`St" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_7 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_7));

    return ret_7;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
}
```

C.69.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ spec_public @*/

  private static project.Entrytypes.St St = new project.Entrytypes.St(0L);
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    final Number ignorePattern_1 = opRet(1L);
    //@ assert Utils.is_nat(ignorePattern_1);

    final Number ignorePattern_2 = f(3L);
    //@ assert Utils.is_nat(ignorePattern_2);

    {
      opVoid();
      IO.println("Before␣breaking␣post␣condition");
      {
        final Number ignorePattern_3 = opRet(4L);
        //@ assert Utils.is_nat(ignorePattern_3);

        /* skip */
      }

      IO.println("After␣breaking␣post␣condition");
      return 0L;
    }
  }
  //@ ensures post_opVoid(\old(St.copy()),St);

  public static void opVoid() {

    //@ assert St != null;

    St.set_x(St.get_x().longValue() + 1L);
  }
  //@ ensures post_opRet(a,\result,\old(St.copy()),St);

  public static Number opRet(final Number a) {

    //@ assert Utils.is_nat(a);

    //@ assert St != null;

    St.set_x(St.get_x().longValue() + 1L);

    Number ret_1 = St.get_x();
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }
  //@ ensures post_f(a,\result);
  /*@ pure @*/

  public static Number f(final Number a) {

    //@ assert Utils.is_nat(a);

    if (Utils.equals(Utils.mod(a.longValue(), 2L), 0L)) {
      Number ret_2 = a.longValue() + 2L;
      //@ assert Utils.is_nat(ret_2);

      return ret_2;

    } else {
      Number ret_3 = a.longValue() + 1L;
      //@ assert Utils.is_nat(ret_3);

      return ret_3;
    }
  }
  /*@ pure @*/

  public static Boolean post_opVoid(
      final project.Entrytypes.St _St, final project.Entrytypes.St St) {

    //@ assert Utils.is_(_St,project.Entrytypes.St.class);

    //@ assert Utils.is_(St,project.Entrytypes.St.class);

    Boolean ret_4 = Utils.equals(St.get_x(), _St.get_x().longValue() + 1L);
    //@ assert Utils.is_bool(ret_4);

    return ret_4;
  }
  /*@ pure @*/

  public static Boolean post_opRet(
      final Number a,
      final Number RESULT,
      final project.Entrytypes.St _St,
      final project.Entrytypes.St St) {

    //@ assert Utils.is_nat(a);

    //@ assert Utils.is_nat(RESULT);

    //@ assert Utils.is_(_St,project.Entrytypes.St.class);

    //@ assert Utils.is_(St,project.Entrytypes.St.class);

    Boolean andResult_1 = false;
    //@ assert Utils.is_bool(andResult_1);

    if (Utils.equals(St.get_x(), _St.get_x().longValue() + 1L)) {
      if (Utils.equals(RESULT, a)) {
        andResult_1 = true;
        //@ assert Utils.is_bool(andResult_1);

      }
    }

    Boolean ret_5 = andResult_1;
    //@ assert Utils.is_bool(ret_5);

    return ret_5;
  }
  /*@ pure @*/

  public static Boolean post_f(final Number a, final Number RESULT) {

    //@ assert Utils.is_nat(a);

    //@ assert Utils.is_nat(RESULT);

    Boolean ret_6 = Utils.equals(Utils.mod(RESULT.longValue(), 2L), 0L);
    //@ assert Utils.is_bool(ret_6);

    return ret_6;
  }

  public String toString() {

    return "Entry{" + "St␣:=␣" + Utils.toString(St) + "}";
  }
}
```

C.69.4 The OpenJML runtime assertion checker output

```
"Before breaking post condition"
Entry.java:50: JML postcondition is false
  public static Number opRet(final Number a) {
                        ^
Entry.java:48: Associated declaration
  //@ ensures post_opRet(a,\result,\old(St.copy()),St);
      ^
"After breaking post condition"
```

C.70 PreCond.vdmsl

C.70.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

state St of
x : nat
init s == s = mk_St(5)
end

operations

Run : () ==> ?
Run () ==
let - = opRet(1)
in
(
  opVoid(2);
  IO`println("Before␣breaking␣pre␣condition");
  let - = id(-1) in skip;
  IO`println("After␣breaking␣pre␣condition");
  return 0;
);

opRet : nat ==> nat
opRet (a) ==
(
  x := a + 1;
  return x;
)
pre x > 0;

opVoid : nat ==> ()
opVoid (a) ==
  x := a + 1
pre St.x > 0;

functions

id : int -> int
id (a) == a
```

```
pre a > 0;

end Entry
```

C.70.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class St implements Record {
  public Number x;

  public St(final Number _x) {

    //@ assert Utils.is_nat(_x);

    x = _x;
    //@ assert Utils.is_nat(x);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.St)) {
      return false;
    }

    project.Entrytypes.St other = ((project.Entrytypes.St) obj);

    return Utils.equals(x, other.x);
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x);
  }
  /*@ pure @*/

  public project.Entrytypes.St copy() {

    return new project.Entrytypes.St(x);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`St" + Utils.formatFields(x);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_6 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(ret_6));

    return ret_6;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_nat(x));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
}
```

C.70.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ spec_public @*/

  private static project.Entrytypes.St St = new project.Entrytypes.St(5L);
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    final Number ignorePattern_1 = opRet(1L);
    //@ assert Utils.is_nat(ignorePattern_1);

    {
      opVoid(2L);
      IO.println("Before␣breaking␣pre␣condition");
      {
        final Number ignorePattern_2 = id(-1L);
        //@ assert Utils.is_int(ignorePattern_2);

        /* skip */
      }

      IO.println("After␣breaking␣pre␣condition");
      return 0L;
    }
  }
  //@ requires pre_opRet(a,St);

  public static Number opRet(final Number a) {

    //@ assert Utils.is_nat(a);

    //@ assert St != null;

    St.set_x(a.longValue() + 1L);

    Number ret_1 = St.get_x();
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }
  //@ requires pre_opVoid(a,St);

  public static void opVoid(final Number a) {

    //@ assert Utils.is_nat(a);

    //@ assert St != null;

    St.set_x(a.longValue() + 1L);
  }
  //@ requires pre_id(a);
  /*@ pure @*/

  public static Number id(final Number a) {

    //@ assert Utils.is_int(a);

    Number ret_2 = a;
    //@ assert Utils.is_int(ret_2);

    return ret_2;
  }
  /*@ pure @*/

  public static Boolean pre_opRet(final Number a, final project.Entrytypes.
      St St) {

    //@ assert Utils.is_nat(a);

    //@ assert Utils.is_(St,project.Entrytypes.St.class);

    Boolean ret_3 = St.get_x().longValue() > 0L;
    //@ assert Utils.is_bool(ret_3);

    return ret_3;
  }
  /*@ pure @*/

  public static Boolean pre_opVoid(final Number a, final project.Entrytypes.
      St St) {

```

```
    //@ assert Utils.is_nat(a);

    //@ assert Utils.is_(St,project.Entrytypes.St.class);

    Boolean ret_4 = St.get_x().longValue() > 0L;
    //@ assert Utils.is_bool(ret_4);

    return ret_4;
  }
  /*@ pure @*/

  public static Boolean pre_id(final Number a) {

    //@ assert Utils.is_int(a);

    Boolean ret_5 = a.longValue() > 0L;
    //@ assert Utils.is_bool(ret_5);

    return ret_5;
  }

  public String toString() {

    return "Entry{" + "St␣:=␣" + Utils.toString(St) + "}";
  }
}
```

C.70.4 The OpenJML runtime assertion checker output

```
"Before breaking pre condition"
Entry.java:27: JML precondition is false
        final Number ignorePattern_2 = id(-1L);
                                         ^
Entry.java:62: Associated declaration
  //@ requires pre_id(a);
      ^
"After breaking pre condition"
```

C.71 TupleSizeMismatch.vdmsl

C.71.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : nat * nat * char * bool = Tup4() in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : nat * nat * char * bool = Tup3() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

Tup3 :  () -> (nat * char * bool) | (nat * nat * char * bool)
Tup3 () == mk_(1,'a',true);

Tup4 :  () -> (nat * char * bool) | (nat * nat * char * bool)
Tup4 () == mk_(1,2,'b',false);

end Entry
```

C.71.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final Tuple ignorePattern_1 = ((Tuple) Tup4());
      //@ assert (V2J.isTup(ignorePattern_1,4) && Utils.is_nat(V2J.field(
          ignorePattern_1,0)) && Utils.is_nat(V2J.field(ignorePattern_1,1))
          && Utils.is_char(V2J.field(ignorePattern_1,2)) && Utils.is_bool(
          V2J.field(ignorePattern_1,3)));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final Tuple ignorePattern_2 = ((Tuple) Tup3());
      //@ assert (V2J.isTup(ignorePattern_2,4) && Utils.is_nat(V2J.field(
          ignorePattern_2,0)) && Utils.is_nat(V2J.field(ignorePattern_2,1))
          && Utils.is_char(V2J.field(ignorePattern_2,2)) && Utils.is_bool(
          V2J.field(ignorePattern_2,3)));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static Object Tup3() {

    Object ret_1 = Tuple.mk_(1L, 'a', true);
    //@ assert ((V2J.isTup(ret_1,3) && Utils.is_nat(V2J.field(ret_1,0)) &&
        Utils.is_char(V2J.field(ret_1,1)) && Utils.is_bool(V2J.field(ret_1
        ,2))) || (V2J.isTup(ret_1,4) && Utils.is_nat(V2J.field(ret_1,0)) &&
        Utils.is_nat(V2J.field(ret_1,1)) && Utils.is_char(V2J.field(ret_1,2)
        ) && Utils.is_bool(V2J.field(ret_1,3))));

    return Utils.copy(ret_1);
  }
  /*@ pure @*/

  public static Object Tup4() {

    Object ret_2 = Tuple.mk_(1L, 2L, 'b', false);
    //@ assert ((V2J.isTup(ret_2,3) && Utils.is_nat(V2J.field(ret_2,0)) &&
        Utils.is_char(V2J.field(ret_2,1)) && Utils.is_bool(V2J.field(ret_2
        ,2))) || (V2J.isTup(ret_2,4) && Utils.is_nat(V2J.field(ret_2,0)) &&
        Utils.is_nat(V2J.field(ret_2,1)) && Utils.is_char(V2J.field(ret_2,2)
        ) && Utils.is_bool(V2J.field(ret_2,3))));

    return Utils.copy(ret_2);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.71.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isTup(ignorePattern_2,4) && Utils.is_nat(V2J.field(
          ignorePattern_2,0)) && Utils.is_nat(V2J.field(ignorePattern_2,1)) &&
          Utils.is_char(V2J.field(ignorePattern_2,2)) && Utils.is_bool(V2J.
          field(ignorePattern_2,3)));
          ^
"After illegal use"
```

C.72 NatBoolTupNil.vdmsl

C.72.1 The VDM-SL model

```
module Entry

```

```
exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : nat * bool = mk_(1,true) in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : nat * bool = TupNil() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

TupNil :  () -> [nat1 * bool]
TupNil () == nil;

TupVal :  () -> [nat1 * bool]
TupVal () == mk_(1,false);

end Entry
```

C.72.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final Tuple ignorePattern_1 = Tuple.mk_(1L, true);
      //@ assert (V2J.isTup(ignorePattern_1,2) && Utils.is_nat(V2J.field(
          ignorePattern_1,0)) && Utils.is_bool(V2J.field(ignorePattern_1,1))
          );

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
27      {
28        final Tuple ignorePattern_2 = TupNil();
29        //@ assert (V2J.isTup(ignorePattern_2,2) && Utils.is_nat(V2J.field(
              ignorePattern_2,0)) && Utils.is_bool(V2J.field(ignorePattern_2,1))
              );
30
31        /* skip */
32      }
33
34      IO.println("After␣illegal␣use");
35      return 0L;
36    }
37    /*@ pure @*/
38
39    public static Tuple TupNil() {
40
41      Tuple ret_1 = null;
42      //@ assert ((ret_1 == null) || (V2J.isTup(ret_1,2) && Utils.is_nat1(V2J.
            field(ret_1,0)) && Utils.is_bool(V2J.field(ret_1,1))));
43
44      return Utils.copy(ret_1);
45    }
46    /*@ pure @*/
47
48    public static Tuple TupVal() {
49
50      Tuple ret_2 = Tuple.mk_(1L, false);
51      //@ assert ((ret_2 == null) || (V2J.isTup(ret_2,2) && Utils.is_nat1(V2J.
            field(ret_2,0)) && Utils.is_bool(V2J.field(ret_2,1))));
52
53      return Utils.copy(ret_2);
54    }
55
56    public String toString() {
57
58      return "Entry{}";
59    }
60  }
```

C.72.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isTup(ignorePattern_2,2) && Utils.is_nat(V2J.field(
          ignorePattern_2,0)) && Utils.is_bool(V2J.field(ignorePattern_2,1)));
          ^
"After illegal use"
```

C.73 NatBooolBasedNamedTypeInv.vdmsl

C.73.1 The VDM-SL model

```
module Entry
```

```

exports all
imports from IO all
definitions

types

TrueEven = nat * bool
inv te == te.#1 > 1000 and te.#2;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : TrueEven = mk_(1001,true) in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣uses");
 let - : TrueEven = mk_(1000,true) in skip;
 let - : TrueEven = mk_(1001,false) in skip;
 IO`println("After␣illegal␣uses");
 return 0;
);

end Entry
```

C.73.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final Tuple ignorePattern_1 = Tuple.mk_(1001L, true);
      //@ assert ((V2J.isTup(ignorePattern_1,2) && Utils.is_nat(V2J.field(
          ignorePattern_1,0)) && Utils.is_bool(V2J.field(ignorePattern_1,1))
          ) && inv_Entry_TrueEven(ignorePattern_1));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣uses");
    {
      final Tuple ignorePattern_2 = Tuple.mk_(1000L, true);
      //@ assert ((V2J.isTup(ignorePattern_2,2) && Utils.is_nat(V2J.field(
          ignorePattern_2,0)) && Utils.is_bool(V2J.field(ignorePattern_2,1))
          ) && inv_Entry_TrueEven(ignorePattern_2));

      /* skip */
    }

    {
      final Tuple ignorePattern_3 = Tuple.mk_(1001L, false);
      //@ assert ((V2J.isTup(ignorePattern_3,2) && Utils.is_nat(V2J.field(
          ignorePattern_3,0)) && Utils.is_bool(V2J.field(ignorePattern_3,1))
          ) && inv_Entry_TrueEven(ignorePattern_3));

      /* skip */
    }

    IO.println("After␣illegal␣uses");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }

  /*@ pure @*/
  /*@ helper @*/

  public static Boolean inv_Entry_TrueEven(final Object check_te) {

    Tuple te = ((Tuple) check_te);

    Boolean andResult_1 = false;

    if (((Number) te.get(0)).longValue() > 1000L) {
      if (((Boolean) te.get(1))) {
        andResult_1 = true;
      }
    }

    return andResult_1;
  }
}
```

C.73.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal uses"
Entry.java:29: JML assertion is false
      //@ assert ((V2J.isTup(ignorePattern_2,2) && Utils.is_nat(V2J.field(
          ignorePattern_2,0)) && Utils.is_bool(V2J.field(ignorePattern_2,1)))
          && inv_Entry_TrueEven(ignorePattern_2));
          ^
Entry.java:36: JML assertion is false
      //@ assert ((V2J.isTup(ignorePattern_3,2) && Utils.is_nat(V2J.field(
          ignorePattern_3,0)) && Utils.is_bool(V2J.field(ignorePattern_3,1)))
          && inv_Entry_TrueEven(ignorePattern_3));
          ^
"After illegal uses"
```

C.74 NatBoolNegField.vdmsl

C.74.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : nat * bool = mk_(1,true) in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : nat * bool = mk_(negInt(),true) in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

negInt :  () -> int
negInt () == -1;

end Entry
```

C.74.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final Tuple ignorePattern_1 = Tuple.mk_(1L, true);
      //@ assert (V2J.isTup(ignorePattern_1,2) && Utils.is_nat(V2J.field(
          ignorePattern_1,0)) && Utils.is_bool(V2J.field(ignorePattern_1,1))
          );

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final Tuple ignorePattern_2 = Tuple.mk_(negInt(), true);
      //@ assert (V2J.isTup(ignorePattern_2,2) && Utils.is_nat(V2J.field(
          ignorePattern_2,0)) && Utils.is_bool(V2J.field(ignorePattern_2,1))
          );

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static Number negInt() {

    Number ret_1 = -1L;
    //@ assert Utils.is_int(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.74.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isTup(ignorePattern_2,2) && Utils.is_nat(V2J.field(
          ignorePattern_2,0)) && Utils.is_bool(V2J.field(ignorePattern_2,1)));
          ^
"After illegal use"
```

C.75 ReturnNonInjMapWhereInjMapRequired.vdmsl

C.75.1 The VDM-SL model

```
module Entry

```

```
exports all
imports from IO all
definitions

types

V2 ::
 x : int
 y : int;

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : inmap nat to V2 = consInjMap() in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : inmap nat to V2 = consInjMapErr() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

consInjMap :  () -> inmap nat to V2
consInjMap () ==
  {1 |-> mk_V2(1,2), 2 |-> mk_V2(2,1)};

consInjMapErr :  () -> inmap nat to V2
consInjMapErr () ==
  {1 |-> mk_V2(1,2), 2 |-> mk_V2(2,1), 3 |-> mk_V2(1,2)};

end Entry
```

C.75.2 The generated Java/JML

```
package project.Entrytypes;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class V2 implements Record {
  public Number x;
  public Number y;

  public V2(final Number _x, final Number _y) {

    //@ assert Utils.is_int(_x);

    //@ assert Utils.is_int(_y);

    x = _x;
    //@ assert Utils.is_int(x);

    y = _y;
    //@ assert Utils.is_int(y);

  }
  /*@ pure @*/

  public boolean equals(final Object obj) {

    if (!(obj instanceof project.Entrytypes.V2)) {
      return false;
    }

    project.Entrytypes.V2 other = ((project.Entrytypes.V2) obj);

    return (Utils.equals(x, other.x)) && (Utils.equals(y, other.y));
  }
  /*@ pure @*/

  public int hashCode() {

    return Utils.hashCode(x, y);
  }
  /*@ pure @*/

  public project.Entrytypes.V2 copy() {

    return new project.Entrytypes.V2(x, y);
  }
  /*@ pure @*/

  public String toString() {

    return "mk_Entry`V2" + Utils.formatFields(x, y);
  }
  /*@ pure @*/

  public Number get_x() {

    Number ret_3 = x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_3));

    return ret_3;
  }

  public void set_x(final Number _x) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_x));

    x = _x;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(x));

  }
  /*@ pure @*/

  public Number get_y() {

    Number ret_4 = y;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(ret_4));

    return ret_4;
  }

  public void set_y(final Number _y) {

    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(_y));

    y = _y;
    //@ assert project.Entry.invChecksOn ==> (Utils.is_int(y));

  }
  /*@ pure @*/

  public Boolean valid() {

    return true;
  }
}
```

C.75.3 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMMap ignorePattern_1 = consInjMap();
      //@ assert (V2J.isInjMap(ignorePattern_1) && (\forall int i; 0 <= i &&
           i < V2J.size(ignorePattern_1); Utils.is_nat(V2J.getDom(
         ignorePattern_1,i)) && Utils.is_(V2J.getRng(ignorePattern_1,i),
         project.Entrytypes.V2.class)));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMMap ignorePattern_2 = consInjMapErr();
      //@ assert (V2J.isInjMap(ignorePattern_2) && (\forall int i; 0 <= i &&
           i < V2J.size(ignorePattern_2); Utils.is_nat(V2J.getDom(
         ignorePattern_2,i)) && Utils.is_(V2J.getRng(ignorePattern_2,i),
         project.Entrytypes.V2.class)));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static VDMMap consInjMap() {

    VDMMap ret_1 =
        MapUtil.map(
            new Maplet(1L, new project.Entrytypes.V2(1L, 2L)),
            new Maplet(2L, new project.Entrytypes.V2(2L, 1L)));
    //@ assert (V2J.isInjMap(ret_1) && (\forall int i; 0 <= i && i < V2J.
        size(ret_1); Utils.is_nat(V2J.getDom(ret_1,i)) && Utils.is_(V2J.
        getRng(ret_1,i),project.Entrytypes.V2.class)));

    return Utils.copy(ret_1);
  }
  /*@ pure @*/

  public static VDMMap consInjMapErr() {

    VDMMap ret_2 =
        MapUtil.map(
            new Maplet(1L, new project.Entrytypes.V2(1L, 2L)),
            new Maplet(2L, new project.Entrytypes.V2(2L, 1L)),
            new Maplet(3L, new project.Entrytypes.V2(1L, 2L)));
    //@ assert (V2J.isInjMap(ret_2) && (\forall int i; 0 <= i && i < V2J.
        size(ret_2); Utils.is_nat(V2J.getDom(ret_2,i)) && Utils.is_(V2J.
        getRng(ret_2,i),project.Entrytypes.V2.class)));

    return Utils.copy(ret_2);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.75.4 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:58: JML assertion is false
    //@ assert (V2J.isInjMap(ret_2) && (\forall int i; 0 <= i && i < V2J.size(
        ret_2); Utils.is_nat(V2J.getDom(ret_2,i)) && Utils.is_(V2J.getRng(ret_2
        ,i),project.Entrytypes.V2.class)));
        ^
Entry.java:29: JML assertion is false
      //@ assert (V2J.isInjMap(ignorePattern_2) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_2); Utils.is_nat(V2J.getDom(ignorePattern_2
          ,i)) && Utils.is_(V2J.getRng(ignorePattern_2,i),project.Entrytypes.V2
          .class)));
          ^
```

```
"After illegal use"
```

C.76 AssignNonInjMapToInjMap.vdmsl

C.76.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : map nat to nat = {1 |-> 2, 3 |-> 4} in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : inmap nat to nat = {1 |-> 2, 3 |-> 2} in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

end Entry
```

C.76.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMMap ignorePattern_1 = MapUtil.map(new Maplet(1L, 2L), new
          Maplet(3L, 4L));
      //@ assert (V2J.isMap(ignorePattern_1) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_1); Utils.is_nat(V2J.getDom(
          ignorePattern_1,i)) && Utils.is_nat(V2J.getRng(ignorePattern_1,i))
          ));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMMap ignorePattern_2 = MapUtil.map(new Maplet(1L, 2L), new
          Maplet(3L, 2L));
      //@ assert (V2J.isInjMap(ignorePattern_2) && (\forall int i; 0 <= i &&
           i < V2J.size(ignorePattern_2); Utils.is_nat(V2J.getDom(
          ignorePattern_2,i)) && Utils.is_nat(V2J.getRng(ignorePattern_2,i))
          ));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.76.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isInjMap(ignorePattern_2) && (\forall int i; 0 <= i && i
           < V2J.size(ignorePattern_2); Utils.is_nat(V2J.getDom(ignorePattern_2
          ,i)) && Utils.is_nat(V2J.getRng(ignorePattern_2,i))));
           ^
"After illegal use"
```

C.77 MapBoolToNatAssignNil.vdmsl

C.77.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : map bool to nat = {false |-> 0, true |-> 1} in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : map bool to nat = mapNil() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

functions

mapNil :  () -> [map bool to nat]
mapNil () == nil;

end Entry
```

C.77.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMMap ignorePattern_1 = MapUtil.map(new Maplet(false, 0L), new
          Maplet(true, 1L));
      //@ assert (V2J.isMap(ignorePattern_1) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_1); Utils.is_bool(V2J.getDom(
          ignorePattern_1,i)) && Utils.is_nat(V2J.getRng(ignorePattern_1,i))
          ));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMMap ignorePattern_2 = mapNil();
      //@ assert (V2J.isMap(ignorePattern_2) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_2); Utils.is_bool(V2J.getDom(
          ignorePattern_2,i)) && Utils.is_nat(V2J.getRng(ignorePattern_2,i))
          ));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static VDMMap mapNil() {

    VDMMap ret_1 = null;
    //@ assert ((ret_1 == null) || (V2J.isMap(ret_1) && (\forall int i; 0 <=
         i && i < V2J.size(ret_1); Utils.is_bool(V2J.getDom(ret_1,i)) &&
        Utils.is_nat(V2J.getRng(ret_1,i)))));

    return Utils.copy(ret_1);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.77.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isMap(ignorePattern_2) && (\forall int i; 0 <= i && i <
          V2J.size(ignorePattern_2); Utils.is_bool(V2J.getDom(ignorePattern_2,i
          )) && Utils.is_nat(V2J.getRng(ignorePattern_2,i))));
          ^
"After illegal use"
```

C.78 MapBoolToNatDetectNegInt.vdmsl

C.78.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
(
 IO`println("Before␣legal␣use");
 let - : map bool to nat = {false |-> 0, true |-> 1} in skip;
 IO`println("After␣legal␣use");
 IO`println("Before␣illegal␣use");
 let - : map bool to nat = mapBoolToInt() in skip;
 IO`println("After␣illegal␣use");
 return 0;
);

```

```
functions

mapBoolToInt :  () -> map bool to int
mapBoolToInt () == {false |-> 0, true |-> -1}

end Entry
```

C.78.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣legal␣use");
    {
      final VDMMap ignorePattern_1 = MapUtil.map(new Maplet(false, 0L), new
          Maplet(true, 1L));
      //@ assert (V2J.isMap(ignorePattern_1) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_1); Utils.is_bool(V2J.getDom(
          ignorePattern_1,i)) && Utils.is_nat(V2J.getRng(ignorePattern_1,i))
          ));

      /* skip */
    }

    IO.println("After␣legal␣use");
    IO.println("Before␣illegal␣use");
    {
      final VDMMap ignorePattern_2 = mapBoolToInt();
      //@ assert (V2J.isMap(ignorePattern_2) && (\forall int i; 0 <= i && i
          < V2J.size(ignorePattern_2); Utils.is_bool(V2J.getDom(
          ignorePattern_2,i)) && Utils.is_nat(V2J.getRng(ignorePattern_2,i))
          ));

      /* skip */
    }

    IO.println("After␣illegal␣use");
    return 0L;
  }
  /*@ pure @*/

  public static VDMMap mapBoolToInt() {

    VDMMap ret_1 = MapUtil.map(new Maplet(false, 0L), new Maplet(true, -1L))
        ;
    //@ assert (V2J.isMap(ret_1) && (\forall int i; 0 <= i && i < V2J.size(
        ret_1); Utils.is_bool(V2J.getDom(ret_1,i)) && Utils.is_int(V2J.
        getRng(ret_1,i))));

    return Utils.copy(ret_1);
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.78.3 The OpenJML runtime assertion checker output

```
"Before legal use"
"After legal use"
"Before illegal use"
Entry.java:29: JML assertion is false
      //@ assert (V2J.isMap(ignorePattern_2) && (\forall int i; 0 <= i && i <
          V2J.size(ignorePattern_2); Utils.is_bool(V2J.getDom(ignorePattern_2,i
          )) && Utils.is_nat(V2J.getRng(ignorePattern_2,i))));
          ^
"After illegal use"
```

C.79 AssignBoolTypeViolation.vdmsl

C.79.1 The VDM-SL model

```
module Entry

imports from IO all
exports all
definitions

operations

Run : () ==> ?
Run () ==
(
  dcl b : bool := true;
  dcl bOpt : [bool] := nil;

  IO`println("Before␣doing␣valid␣assignments");
  bOpt := true;
  b := bOpt;
  bOpt := nil;
  IO`println("After␣doing␣valid␣assignments");

  IO`println("Before␣doing␣illegal␣assignments");
  b := bOpt;
  IO`println("After␣doing␣illegal␣assignments");

  return true;
);
```

```


end Entry
```

C.79.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    Boolean b = true;
    //@ assert Utils.is_bool(b);

    Boolean bOpt = null;
    //@ assert ((bOpt == null) || Utils.is_bool(bOpt));

    IO.println("Before␣doing␣valid␣assignments");
    bOpt = true;
    //@ assert ((bOpt == null) || Utils.is_bool(bOpt));

    b = bOpt;
    //@ assert Utils.is_bool(b);

    bOpt = null;
    //@ assert ((bOpt == null) || Utils.is_bool(bOpt));

    IO.println("After␣doing␣valid␣assignments");
    IO.println("Before␣doing␣illegal␣assignments");
    b = bOpt;
    //@ assert Utils.is_bool(b);

    IO.println("After␣doing␣illegal␣assignments");
    return true;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.79.3 The OpenJML runtime assertion checker output

```
"Before doing valid assignments"
```

```
"After doing valid assignments"
"Before doing illegal assignments"
Entry.java:36: JML assertion is false
    //@ assert Utils.is_bool(b);
        ^
"After doing illegal assignments"
```

C.80 VarDeclTypeViolation.vdmsl

C.80.1 The VDM-SL model

```
module Entry

imports from IO all
exports all
definitions

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣VALID␣initialisation");
  (dcl bOkay : nat := natOne(); skip);
  IO`println("After␣VALID␣initialisation");

  IO`println("Before␣INVALID␣initialisation");
  (dcl bError : nat := natNil(); skip);
  IO`println("After␣INVALID␣initialisation");

  return true;
);

functions

natNil :  () -> [nat]
natNil () == nil;

natOne :  () -> [nat]
natOne () == 1;

end Entry
```

C.80.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣VALID␣initialisation");
    {
      Number bOkay = natOne();
      //@ assert Utils.is_nat(bOkay);

      /* skip */
    }

    IO.println("After␣VALID␣initialisation");
    IO.println("Before␣INVALID␣initialisation");
    {
      Number bError = natNil();
      //@ assert Utils.is_nat(bError);

      /* skip */
    }

    IO.println("After␣INVALID␣initialisation");
    return true;
  }
  /*@ pure @*/

  public static Number natNil() {

    Number ret_1 = null;
    //@ assert ((ret_1 == null) || Utils.is_nat(ret_1));

    return ret_1;
  }
  /*@ pure @*/

  public static Number natOne() {

    Number ret_2 = 1L;
    //@ assert ((ret_2 == null) || Utils.is_nat(ret_2));

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.80.3 The OpenJML runtime assertion checker output

```
"Before VALID initialisation"
"After VALID initialisation"
"Before INVALID initialisation"
Entry.java:29: JML assertion is false
      //@ assert Utils.is_nat(bError);
```

```
         ^
"After INVALID initialisation"
```

C.81 FuncReturnTokenViolation.vdmsl

C.81.1 The VDM-SL model

```
module Entry

imports from IO all
exports all
definitions

operations

Run : () ==> ?
Run () ==
(
  IO`println("Before␣evaluating␣ok()");
  let - = ok() in skip;
  IO`println("After␣evaluating␣ok()");

  IO`println("Before␣evaluating␣error()");
  let - = err() in skip;
  IO`println("After␣evaluating␣error()");

  return true;
);

functions

ok : () -> token
ok () ==
let aOpt : [token] = mk_token("")
in
  aOpt;


err : () -> token
err () ==
let aOpt : [token] = nil
in
  aOpt;

end Entry
```

C.81.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
```

```
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    IO.println("Before␣evaluating␣ok()");
    {
      final Token ignorePattern_1 = ok();
      //@ assert Utils.is_token(ignorePattern_1);

      /* skip */
    }

    IO.println("After␣evaluating␣ok()");
    IO.println("Before␣evaluating␣error()");
    {
      final Token ignorePattern_2 = err();
      //@ assert Utils.is_token(ignorePattern_2);

      /* skip */
    }

    IO.println("After␣evaluating␣error()");
    return true;
  }
  /*@ pure @*/

  public static Token ok() {

    final Token aOpt = new Token("");
    //@ assert ((aOpt == null) || Utils.is_token(aOpt));

    Token ret_1 = aOpt;
    //@ assert Utils.is_token(ret_1);

    return ret_1;
  }
  /*@ pure @*/

  public static Token err() {

    final Token aOpt = null;
    //@ assert ((aOpt == null) || Utils.is_token(aOpt));

    Token ret_2 = aOpt;
    //@ assert Utils.is_token(ret_2);

    return ret_2;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.81.3 The OpenJML runtime assertion checker output

```
"Before evaluating ok()"
"After evaluating ok()"
"Before evaluating error()"
Entry.java:57: JML assertion is false
    //@ assert Utils.is_token(ret_2);
        ^
Entry.java:29: JML assertion is false
      //@ assert Utils.is_token(ignorePattern_2);
          ^
"After evaluating error()"
```

C.82 OpParamQuoteTypeViolation.vdmsl

C.82.1 The VDM-SL model

```
module Entry

imports from IO all
exports all
definitions

operations

Run : () ==> ?
Run () ==
let aOpt : [<A>] = nil,
    a : <A> = <A>
in
(
  IO`println("Before␣passing␣LEGAL␣value");
  op(a);
  IO`println("After␣passing␣LEGAL␣value");

  IO`println("Before␣passing␣ILLEGAL␣value");
  op(aOpt);
  IO`println("After␣passing␣ILLEGAL␣value");

  return true;
);

op : <A> ==> ()
op (-) == skip;

end Entry
```

C.82.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;
```

```
 6
 7  @SuppressWarnings("all")
 8  //@ nullable_by_default
 9
10  final public class Entry {
11    /*@ public ghost static boolean invChecksOn = true; @*/
12
13    private Entry() {}
14
15    public static Object Run() {
16
17      final project.quotes.AQuote aOpt = null;
18      //@ assert ((aOpt == null) || Utils.is_(aOpt,project.quotes.AQuote.class
          ));
19
20      final project.quotes.AQuote a = project.quotes.AQuote.getInstance();
21      //@ assert Utils.is_(a,project.quotes.AQuote.class);
22
23      {
24        IO.println("Before␣passing␣LEGAL␣value");
25        op(a);
26        IO.println("After␣passing␣LEGAL␣value");
27        IO.println("Before␣passing␣ILLEGAL␣value");
28        op(aOpt);
29        IO.println("After␣passing␣ILLEGAL␣value");
30        return true;
31      }
32    }
33
34    public static void op(final project.quotes.AQuote ignorePattern_1) {
35
36      //@ assert Utils.is_(ignorePattern_1,project.quotes.AQuote.class);
37
38      /* skip */
39
40    }
41
42    public String toString() {
43
44      return "Entry{}";
45    }
46  }
```

C.82.3 The OpenJML runtime assertion checker output

```
"Before passing LEGAL value"
"After passing LEGAL value"
"Before passing ILLEGAL value"
Entry.java:36: JML assertion is false
    //@ assert Utils.is_(ignorePattern_1,project.quotes.AQuote.class);
        ^
"After passing ILLEGAL value"
```

C.83 Exists.vdmsl

C.83.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
let - = f()
in
(
  IO`println("Done!␣Expected␣no␣errors");
  return 0;
);

functions

f :  () -> bool
f () ==
  exists x in set {1,2,3} & x > 0;

end Entry
```

C.83.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    final Boolean ignorePattern_1 = f();
    //@ assert Utils.is_bool(ignorePattern_1);

    {
      IO.println("Done!␣Expected␣no␣errors");
      return 0L;
    }
  }
  /*@ pure @*/

  public static Boolean f() {

    Boolean existsExpResult_1 = false;
    //@ assert Utils.is_bool(existsExpResult_1);

    VDMSet set_1 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_nat1(V2J.get(set_1,i))));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() && !(
        existsExpResult_1); ) {
      Number x = ((Number) iterator_1.next());
      //@ assert Utils.is_nat1(x);

      existsExpResult_1 = x.longValue() > 0L;
      //@ assert Utils.is_bool(existsExpResult_1);

    }
    Boolean ret_1 = existsExpResult_1;
    //@ assert Utils.is_bool(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.83.3 The OpenJML runtime assertion checker output

```
"Done! Expected no errors"
```

C.84 ForAll.vdmsl

C.84.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
let - = f()
in
(
  IO`println("Done!␣Expected␣no␣errors");
  return 0;
);

```

```
functions

f :  () -> bool
f () ==
  forall x in set {1,2,3} & x > 0;

end Entry
```

C.84.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    final Boolean ignorePattern_1 = f();
    //@ assert Utils.is_bool(ignorePattern_1);

    {
      IO.println("Done!␣Expected␣no␣errors");
      return 0L;
    }
  }
  /*@ pure @*/

  public static Boolean f() {

    Boolean forAllExpResult_1 = true;
    //@ assert Utils.is_bool(forAllExpResult_1);

    VDMSet set_1 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_nat1(V2J.get(set_1,i))));

    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() &&
        forAllExpResult_1; ) {
      Number x = ((Number) iterator_1.next());
      //@ assert Utils.is_nat1(x);

      forAllExpResult_1 = x.longValue() > 0L;
      //@ assert Utils.is_bool(forAllExpResult_1);

    }
    Boolean ret_1 = forAllExpResult_1;
    //@ assert Utils.is_bool(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

C.84.3 The OpenJML runtime assertion checker output

```
"Done! Expected no errors"
```

C.85 Exists1.vdmsl

C.85.1 The VDM-SL model

```
module Entry

exports all
imports from IO all
definitions

operations

Run : () ==> ?
Run () ==
let - = f()
in
(
  IO`println("Done!␣Expected␣no␣errors");
  return 0;
);

functions

f :  () -> bool
f () ==
  exists1 x in set {1,2,3} & x > 0;

end Entry
```

C.85.2 The generated Java/JML

```
package project;

import java.util.*;
import org.overture.codegen.runtime.*;
import org.overture.codegen.vdm2jml.runtime.*;

@SuppressWarnings("all")
//@ nullable_by_default

final public class Entry {
  /*@ public ghost static boolean invChecksOn = true; @*/

  private Entry() {}

  public static Object Run() {

    final Boolean ignorePattern_1 = f();
    //@ assert Utils.is_bool(ignorePattern_1);

    {
      IO.println("Done!␣Expected␣no␣errors");
      return 0L;
    }
  }
  /*@ pure @*/

  public static Boolean f() {

    Long exists1Counter_1 = 0L;

    VDMSet set_1 = SetUtil.set(1L, 2L, 3L);
    //@ assert (V2J.isSet(set_1) && (\forall int i; 0 <= i && i < V2J.size(
        set_1); Utils.is_nat1(V2J.get(set_1,i))));

    for (Iterator iterator_1 = set_1.iterator();
        iterator_1.hasNext() && (exists1Counter_1.longValue() < 2L);
        ) {
      Number x = ((Number) iterator_1.next());
      //@ assert Utils.is_nat1(x);

      if (x.longValue() > 0L) {
        exists1Counter_1++;
      }
    }
    Boolean ret_1 = Utils.equals(exists1Counter_1, 1L);
    //@ assert Utils.is_bool(ret_1);

    return ret_1;
  }

  public String toString() {

    return "Entry{}";
  }
}
```

Bibliography

[1] Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)

[2] The test examples used to validate the JML translator using the OpenJML runtime assertion checker. https://github.com/overturetool/overture/tree/development/core/codegen/vdm2jml/src/test/resources/dynamic_analysis/ (2016)

[3] Andrews, D., Bruun, H., Damm, F., Dawes, J., Hansen, B., Larsen, P., Parkin, G., Plat, N., Totenel, H.: A Formal Definition of VDM-SL. Tech. Rep. 1998/9, Leicester University (June 1998)

[4] The Apache Maven Project website. https://maven.apache.org (2016)

[5] Bjørner, D., Jones, C. (eds.): The Vienna Development Method: The Meta-Language, Lecture Notes in Computer Science, vol. 61. Springer-Verlag (1978)

[6] Burdy, L., Cheon, Y., Cok, D., Ernst, M.D., Kiniry, J.R., Leavens, G.T., Leino, K.R.M., Poll, E.: An overview of JML Tools and Applications. Intl. Journal of Software Tools for Technology Transfer 7, 212–232 (2005)

[7] Cok, D.R.: Reasoning with specifications containing method calls and model fields. Journal of Object Technology 4(8), 77–103 (2005)

[8] Cok, D.: OpenJML: JML for Java 7 by Extending OpenJDK. In: Bobaru, M., Havelund, K., Holzmann, G., Joshi, R. (eds.) NASA Formal Methods, Lecture Notes in Computer Science, vol. 6617, pp. 472–479. Springer Berlin Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-20398-5_35

[9] Filliâtre, J.C.: Why: a multi-language multi-prover verification tool. Research Report 1366, LRI, Université Paris Sud (March 2003), http://www.lri.fr/~filliatr/ftp/publis/why-tool.ps.gz

[10] Fitzgerald, J.S., Larsen, P.G., Verhoef, M.: Vienna Development Method. Wiley Encyclopedia of Computer Science and Engineering (2008), edited by Benjamin Wah, John Wiley & Sons, Inc.

[11] Fitzgerald, J., Larsen, P.G.: Modelling Systems – Practical Tools and Techniques in Software Development. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, Second edn. (2009), ISBN 0-521-62348-0

[12] Fitzgerald, J., Larsen, P.G., Mukherjee, P., Plat, N., Verhoef, M.: Validated Designs for Object–oriented Systems. Springer, New York (2005), http://overturetool.org/publications/books/vdoos/

[13] Hubbers, E., Oostdijk, M.: Generating JML specifications from UML state diagrams. In: Forum on Specification and Design Languages FDL'03. pp. 263–273 (2003)

[14] Jin, D., Yang, Z.: Strategies of Modeling from VDM-SL to JML. Advanced Language Processing and Web Information Technology, International Conference on 0, 320–323 (2008)

[15] Jones, C.B.: Software Development A Rigorous Approach. Prentice-Hall International, Englewood Cliffs, New Jersey (1980)

[16] Jørgensen, P.W.V., Couto, L.D., Larsen, M.: A Code Generation Platform for VDM. In: Proceedings of the 12th Overture workshop (June 2014)

[17] Klebanov, A.: Automata-Based Programming Technology Extension for Generation of JML Annotated Java Card Code. pp. 41–44. Proc. of the Spring/Summer Young Researchers' Colloquium on Software Engineering (2008)

[18] Larsen, P.G.: Ten Years of Historical Development: "Bootstrapping" VDMTools. Journal of Universal Computer Science 7(8), 692–709 (2001)

[19] Larsen, P.G., Battle, N., Ferreira, M., Fitzgerald, J., Lausdahl, K., Verhoef, M.: The Overture Initiative – Integrating Tools for VDM. SIGSOFT Softw. Eng. Notes 35(1), 1–6 (January 2010), http://doi.acm.org/10.1145/1668862.1668864

[20] Larsen, P.G., Lausdahl, K., Battle, N.: Combinatorial Testing for VDM. In: Proceedings of the 2010 8th IEEE International Conference on Software Engineering and Formal Methods. pp. 278–285. SEFM '10, IEEE Computer Society, Washington, DC, USA (September 2010), http://dx.doi.org/10.1109/SEFM.2010.32, ISBN 978-0-7695-4153-2

[21] Larsen, P.G., Lausdahl, K., Battle, N., Fitzgerald, J., Wolff, S., Sahara, S., Verhoef, M., Tran-Jørgensen, P.W.V., Oda, T., Chisholm, P.: The VDM-10 Language Manual. Tech. Rep. TR-2010-06, The Overture Open Source Initiative (April 2010)

[22] Larsen, P.G., Lausdahl, K., Tran-Jørgensen, P.W.V., Ribeiro, A., Wolff, S., Battle, N.: Overture VDM-10 Tool Support: User Guide. Tech. Rep. TR-2010-02, The Overture Initiative, www.overturetool.org (May 2010)

[23] Lausdahl, K., Larsen, P.G., Battle, N.: A Deterministic Interpreter Simulating A Distributed real time system using VDM. In: Qin, S., Qiu, Z. (eds.) Proceedings of the 13th international conference on Formal methods and software engineering. Lecture Notes in Computer Science, vol. 6991, pp. 179–194. Springer-Verlag, Berlin, Heidelberg (October 2011), ISBN 978-3-642-24558-9

[24] Leavens, G.T., Poll, E., Clifton, C., Cheon, Y., Ruby, C., Cok, D., Kiniry, J.: JML Reference Manual, revision 2344 edn. (May 2013)

[25] Lensink, L., Smetsers, S., van Eekelen, M.: Generating Verifiable Java Code from Verified PVS Specifications. In: Goodloe, A., Person, S. (eds.) NASA Formal Methods, Lecture Notes in Computer Science, vol. 7226, pp. 310–325. Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-28891-3_30

[26] McCarthy, J.: A Basis for a Mathematical Theory of Computation. In: Western Joint Computer Conference (1961)

[27] Meyer, B.: Object-oriented Software Construction. Prentice-Hall International (1988)

[28] Müller, P., Poetzsch-Heffter, A., Leavens, G.T.: Modular Invariants for Layered Object Structures. Sci. Comput. Program. 62(3), 253–286 (Oct 2006), `http://dx.doi.org/10.1016/j.scico.2006.03.001`

[29] The OpenJML website. `http://http://www.openjml.org` (2016)

[30] The Overture tool website. `http://overturetool.org/` (2015)

[31] The Overture tool Github repository. `https://github.com/overturetool/overture` (2016)

[32] Owre, S., Rushby, J.M., Shankar, N.: PVS: A Prototype Verification System. In: Kapur, D. (ed.) 11th International Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 607, pp. 748–752. Springer-Verlag, Saratoga, NY (June 1992)

[33] Rivera, V., Cataño, N., Wahls, T., Rueda, C.: Code generation for Event-B. International Journal on Software Tools for Technology Transfer pp. 1–22 (2015)

[34] Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education (2004)

[35] Tran-Jørgensen, P.W.V., Larsen, P.G., Battle, N.: Using JML-based Code Generation to Enhance Test Automation for VDM Models. In: Proceedings of the 14th Overture Workshop (November 2016)

[36] Tran-Jørgensen, P.W.V., Larsen, P.G., Leavens, G.T.: Automated translation of vdm to jml-annotated java. International Journal on Software Tools for Technology Transfer pp. 1–25 (2017), `http://dx.doi.org/10.1007/s10009-017-0448-3`

[37] Vilhena, C.: Connecting between VDM++ and JML. Master's thesis, Minho University with exchange to Engineering College of Aarhus (July 2008)

[38] Wing, J.M.: Writing Larch interface language specifications. ACM Trans. Progr. Lang. Syst. 9(1), 1–24 (Jan 1987), `http://doi.acm.org/10.1145/9758.10500`

[39] Woodcock, J., Davies, J.: Using Z – Specification, Refinement, and Proof. Prentice Hall International Series in Computer Science (1996)

[40] Yi, J., Robby, Deng, X., Roychoudhury, A.: Past Expression: Encapsulating Pre-states at Post-conditions by Means of AOP. In: Proceedings of the 12th Annual International Conference on Aspect-oriented Software Development. pp. 133–144. AOSD '13, ACM (2013), `http://doi.acm.org/10.1145/2451436.2451453`

[41] Zhen, Z.: Java Card Technology for Smart Cards. Prentice-Hall, Boston (2000)

[42] Zhou, J., Jin, D.: Research on modeling from VDM-SL to JML for systematic software development. In: Control and Decision Conference (CCDC), 2010 Chinese. pp. 2312–2317. IEEE, Xuzhou (July 2010)

Peter W. V. Tran-Jørgensen, Automated translation of VDM-SL to JML-annotated Java, 2017

Department of Engineering
Aarhus University
Inge Lehmanns Gade 10
8000 Aarhus
Denmark

Tel.: +45 8715 0000