Automated translation of VDM-SL to JML-annotated Java

Technical Report

Peter W. V. Tran-Jørgensen

Aarhus University - Department of Engineering

Table of Contents

1 Introduction

A model specified using the Vienna Development Method (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 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 Java Modeling Language (JML) annotations. This translation is achieved using Overture's *JML translator*, which translates Vienna Development Method Specification Language (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 [3] to make the approach fully automated. The tool implementation of the JML translation is available in Overture 2.3.8 (as of July 2016), which can be downloaded via the Overture tool's website [6]. For instructions on how to use the JML translator we refer the reader to Overture's user guide [4], 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 [2] 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 [5].

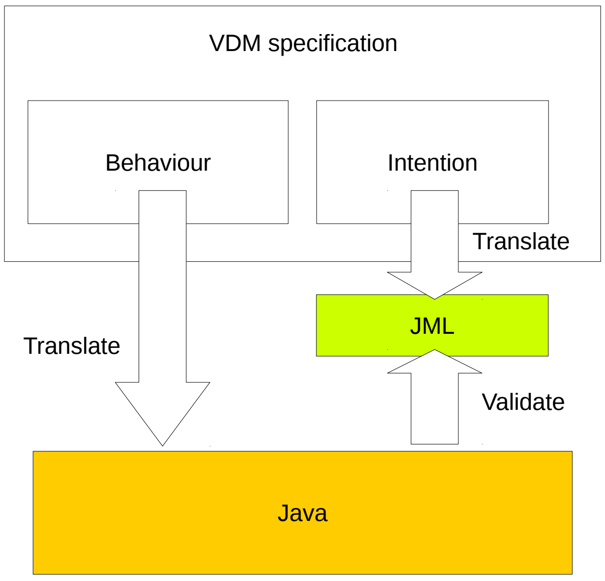

Figure 1.1: Overview of the VDM-to-JML translation.

1.2 About this report

The purpose of this report is to assist VDM users with getting started using the JML translator. This report is complementary to [7], which presents the different translation rules using an example of an ATM that is modelled in VDM-SL and implemented using the JML translator. The implementation of the ATM model is validated using the OpenJML runtime assertion checker. Due to space limitations only a reduced number of examples of limited detail are provided in [7]. To improve this, this report presents a more complete definition of the translation that uses several examples to demonstrate the translation.

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 example from [7], and the corresponding Java/JML implementation is available in appendix B.

2

The Translation

Add some of the 85 examples used to validate the translation

A

The ATM VDM-SL Example

This appendix includes the complete version of the VDM-SL ATM example used to demonstrate the JML translation presented in [7] as well as this technical report.

```
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 example

```
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

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>";
  }
}
```

```
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

Mention this chapter in “About this report”

Since the translation is not formally defined all the translation rules have been validated by running examples, or *regression tests*, through the tool. Each test consists of a VDM-SL model that is used to test some aspect of the translation (such as a single rule). Using the JML translator each test model is translated to a JML-annotated Java program that is executed using the OpenJML runtime assertion checker. The (actual) output obtained by executing the generated Java/JML is then compared to some expected result to see if the translation works as expected result.

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 [1].

C.1 Map.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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 */
}

VDMSet setCompResult_4 = SetUtil.set();
//@ assert (V2J.isSet(setCompResult_4) && (\forall int i; 0 <= i && i <
    V2J.size(setCompResult_4); Utils.is_nat(V2J.get(setCompResult_4,i))));

VDMSet set_4 = SetUtil.set(0L, 1L, 2L);
//@ 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);

  setCompResult_4.add(x);
}
```

```
    {
      final VDMSet ignorePattern_4 = Utils.copy(setCompResult_4);
      //@ assert (V2J.isSet(ignorePattern_4) && (\forall int i; 0 <= i && i <
          V2J.size(ignorePattern_4); Utils.is_nat1(V2J.get(ignorePattern_4,i)))
          );

      /* skip */
    }

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

  public String toString() {

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

```
"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

```
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
```

```
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);
  }
}
```

```
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) + "}";
  }
}
```

```
"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

```
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
```

```
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);
  }
}
```

```
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) + "}";
  }
}
```

```
"Before atomic"
"After atomic"
```

C.6 InvChecksOnFlagInOtherModule.vdmsl

```
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
```

```
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;
  }
}
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
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{}";
  }
}
```

```
"Done! Expected to exit without any errors"
```

C.7 AtomicRecUnion.vdmsl

```
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
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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

```
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
```

```
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;
  }
}
```

```
"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

```
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
```

```
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));
    IO.println("Breaking named type invariant for sequence");
    //@ assert s != null;

    Utils.mapSeqUpdate(s, 3L, 4L);
    //@ assert ((V2J.isSeq(s) && (\forall int i; 0 <= i && i < V2J.size(s);
        true)) && inv_Entry_S(s));

    return 0L;
  }

  public String toString() {

    return "Entry{}";
  }

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

  public static Boolean inv_Entry_S(final Object check_s) {

    VDMSeq s = ((VDMSeq) check_s);

    Boolean forAllExpResult_1 = true;
    VDMSet set_1 = SeqUtil.elems(Utils.copy(s));
    for (Iterator iterator_1 = set_1.iterator(); iterator_1.hasNext() &&
        forAllExpResult_1; ) {
      Object x = ((Object) iterator_1.next());
      Boolean orResult_1 = false;

      if (!(Utils.is_nat(x))) {
        orResult_1 = true;
      } else {
        orResult_1 = ((Number) x).doubleValue() > 5L;
      }

      forAllExpResult_1 = orResult_1;
    }
    return forAllExpResult_1;
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
"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

```
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
```

```
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);
  }
}
```

```
"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

```
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
```

```
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.16 NamedTypeMadeOptional.vdmsl

```
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
```

```
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);
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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;
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.19 TernaryIf.vdmsl

```
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
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
"Done! Expected no violations"
```

C.20 LetBeStStm.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.21 LetBeStExp.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.22 RealParamNil.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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 = 0L;
    //@ assert ((n == null) || Utils.is_nat(n));
    IO.println("Before valid use.");
    n = 1L;
    //@ assert ((n == null) || Utils.is_nat(n));

    n = null;
    //@ assert ((n == null) || Utils.is_nat(n));

    IO.println("After valid use.");
    IO.println("Before invalid use.");
    n = idNat(n);
    //@ assert ((n == null) || Utils.is_nat(n));

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

  public static Number idNat(final Number x) {

    //@ assert Utils.is_nat(x);

    Number ret_1 = x;
    //@ assert Utils.is_nat(ret_1);

    return ret_1;
  }

  public String toString() {

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

```
"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

```
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
```

```
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 */
    }

    IO.println("After valid use.");
    IO.println("Before invalid use.");
    {
      final Character ignorePattern_2 = charNil();
      //@ assert Utils.is_char(ignorePattern_2);

      /* skip */
    }

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

  public static Character charA() {

    Character ret_1 = 'a';
    //@ assert ((ret_1 == null) || 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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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
        ) + "}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
"Done! Expected no violations"
```

C.32 TupLet.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.33 RecParam.vdmsl

```
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
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
"Done! Expected no violations"
```

C.34 TupParam.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.35 SeqNat1BoolMaskedAsNamedTypeInv.vdmsl

```
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
```

```
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;
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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);
  }
}
```

```
"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

```
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
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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{}";
  }
}
```

```
"\invariant_for is not implemented in OpenJML RAC so the \invariant_for check
    will not detect the invariant violation"
```

C.41 Simple.vdmsl

```
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
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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{}";
  }
}
```

```
"\invariant_for is not implemented in OpenJML RAC so the \invariant_for check
    will not detect the invariant violation"
```

C.42 AtomicRecUnion.vdmsl

```
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
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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{}";
  }
}
```

```
"\invariant_for is not implemented in OpenJML RAC so the \invariant_for check
    will not detect the invariant violation"
```

C.43 SetEvenNamedTypeInv.vdmsl

```
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
```

```
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);
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
"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

```
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
```

```
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);
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
"Done! Expected no violations"
```

C.52 RecInRecInAtomic.vdmsl

```
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
```

```
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.r3.x, 1L));
  }
}
```

```
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));
  }
}
```

```
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();

    } /* 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{}";
  }
}
```

```
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));
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
"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

```
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
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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));
  }
}
```

```
"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

```
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
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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));
  }
}
```

```
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));
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
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;
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.60 Real.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.61 String.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.62 Quote.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.63 Int.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.64 Nil.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.65 Char.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no violations"
```

C.66 StateInitViolatesInv.vdmsl

```
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
```

```
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;
  }
}
```

```
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) + "}";
  }
}
```

```
St.java:12: JML invariant is false
  //@ public instance invariant project.Entry.invChecksOn ==> inv_St(x);
                      ^
```

C.67 RecTypeDefInv.vdmsl

```
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
```

```
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{}";
  }
}
```

```
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;
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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) + "}";
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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) + "}";
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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) + "}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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 = TupNil();
      //@ 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 Tuple TupNil() {
    Tuple ret_1 = null;
    //@ 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))));

    return Utils.copy(ret_1);
  }
  /*@ pure @*/

  public static Tuple TupVal() {

    Tuple ret_2 = Tuple.mk_(1L, false);
    //@ 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))));

    return Utils.copy(ret_2);
  }

  public String toString() {

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

```
"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

```
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
```

```
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;
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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;
  }
}
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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{}";
  }
}
```

```
"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

```
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
```

```
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 aOpt = null;
    //@ assert ((aOpt == null) || Utils.is_(aOpt,project.quotes.AQuote.class))
        ;

    final project.quotes.AQuote a = project.quotes.AQuote.getInstance();
    //@ assert Utils.is_(a,project.quotes.AQuote.class);

    {
      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;
    }
  }

  public static void op(final project.quotes.AQuote ignorePattern_1) {

    //@ assert Utils.is_(ignorePattern_1,project.quotes.AQuote.class);

    /* skip */

  }

  public String toString() {

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

```
"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

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no errors"
```

C.84 ForAll.vdmsl

```
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
```

```
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{}";
  }
}
```

```
"Done! Expected no errors"
```

C.85 Exists1.vdmsl

```
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
```

```
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] 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)

[2] Cok, D.R.: OpenJML: JML for Java 7 by Extending OpenJDK. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NASA Formal Methods, Lecture Notes in Computer Science, vol. 6617, pp. 472–479. Springer Berlin Heidelberg (2011)

[3] Jørgensen, P.W.V., Couto, L.D., Larsen, M.: A Code Generation Platform for VDM. In: The Overture 2014 workshop (June 2014)

[4] 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)

[5] The OpenJML website. http://http://www.openjml.org (2016)

[6] The Overture tool's website. http://overturetool.org (2016)

[7] Tran-Jørgensen, P.W.V., Larsen, P.G., Leavens, G.T.: Automated translation of VDM to JML annotated Java (Jan 2016 Accepted to appear in the International Journal on Software Tools for Technology Transfer (STTT))