Verifying Java Programs

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at

Verifying Java Programs

- ESC/Java2: extended static checking, not verification.
  - Even if no error is reported, a program may violate its specification.
    - Incomplete calculus for verifying while loops.
    - Incomplete calculus in automatic decision procedure (Simplify).
- We will now focus on the real verification of Java programs.
  - Complete verification calculus.
    - No finite unfolding of loops, but reasoning based on invariants.
    - Loop/class invariants must be typically provided by user.
  - Automatic generation of verification conditions.
    - From JML-annotated Java program, proof obligations are derived.
  - Human-guided proofs of these conditions (using a proof assistant).
    - Simple conditions automatically proved by automatic procedure.

We are going to present two tools for this purpose.

1. The Krakatoa/Why Tool Suite

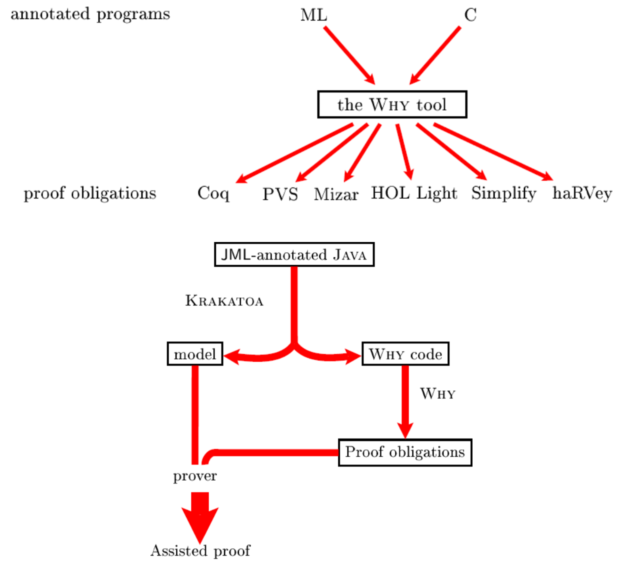
2. The KeY Tool

The Krakatoa/Why Tool Suite

- Why: generation of verification conditions.
  - Jean-Christophe Filliatre et al, LRI/INRIA, France, 2003–
    `http://why.lri.fr`
    Fillatre: "Why: a multi-language multi-prover verification condition generator", 2003.
  - Input: an annotated programs in ML (or C).
  - Output: proof obligations for Coq, PVS, Isabelle/HOL, HOL 4, HOL Light, Mizar, Simplify, CVC Lite, haRVey.
- Krakatoa: translating Java programs into Why input.
  - Claude Marche et al, LRI/INRIA, France, 2003–
    `http://krakatoa.lri.fr`
    Marche et al: "The Krakatoa Tool for Certification of Java/JavaCard Programs annotated in JML", 2003.
  - Input: an JML-annotated Java program.
  - Output: an ML program for Why and a model for a prover.
    - Support for Coq, PVS, Simplify, haRVey.

We will use Krakatoa 0.66/Why 1.60 with the PVS proof assistant.

Relationship



A Simple Verification

Marche et al: “The Krakatoa Tool Version 0.66”, 2005.

```
package tutorial;

public class Lesson1
{
  /*@ public normal_behavior
    @   requires x >= 0;
    @   ensures
    @     \result >= 0 &&
    @     \result*\result <= x &&
    @     x < (\result+1)*(\result+1);
    @*/
  public static int sqrt(int x)
```

```
  {
    int count = 0;
    int sum = 1;
    /*@ loop_invariant
      @   count >= 0 &&
      @   x >= count*count &&
      @   sum == (count+1)*(count+1);
      @ decreases x-sum;
      @*/
    while (sum <= x)
    {
      count = count+1;
      sum = sum+2*count+1;
    }
    return count;
  }
}
```

A Simple Verification (Contd)

```
> krakatoa
Krakatoa version 0.66 - Wed Jul 20 10:16:29 CEST 2005
krakatoa [options] class.method ...
  -dump dump typing environments
  -p main source package
  -parse-only perform only parsing
  -I input path
  -nojavalang do not import java.lang package
  -coqdir additional input path to pass to coqc using -I
  -coqopt additional option to give to coqc
  -v increments verbosity
  -k do not stop on first error
  -valid produce validation (incompatible with -bb)
  -novalid do not produce validation
  -bb use Why black boxes (incompatible with -valid)
  -globalmemorymodel use the global memory model for translation
  -localmemorymodel use the local memory model for translation (default)
  -coq produce output for the Coq proof assistant
  -simplify produce output for the Simplify prover
  -harvey produce output for the haRVey prover
  -pvs produce output for PVS
  -help  Display this list of options
```

A Simple Verification (Contd’2)

```
> ls
tutorial
> ls tutorial
Lesson1.java
> krakatoa -pvs -p tutorial Lesson1.sqrt
Krakatoa version 0.66 - Wed Jul 20 10:16:29 CEST 2005
Generating Why program Lesson1_sqrt
> ls
krakatoa.log  tutorial
> cd tutorial
> ls
Krak_model.pvs  Krak_spec.why  Lesson1_sqrt.why  spec_imports.v
Krak_model.v    Lesson1.java   makefile
```

Generating the Why input.

A Simple Verification (Contd’3)

```
> make pvs
cp /software/lib/krakatoa/local_memory_template.why local_memory.why
Running why on generated programs...
why --pvs --pvs-preamble "importing Krak_model" local_memory.why \
        Krak_spec.why \
        Lesson1_sqrt.why
echo '(typecheck "Krak_model")' > pvsbatch.el
echo '(typecheck "Lesson1_sqrt_why")' >> pvsbatch.el
pvs -q -v 3 -batch -l pvsbatch.el
...
Parsing Krak_model
Krak_model parsed in 3.61 seconds
Typechecking Krak_model
...
> ls
Krak_model.pvs  Krak_spec_why.pvs  Lesson1_sqrt_why.pvs  makefile
Krak_model.v    Lesson1.java       local_memory.why      pvsbatch.el
Krak_spec.why   Lesson1_sqrt.why   local_memory_why.pvs  spec_imports.v
```

Generating the PVS proof obligations and type checking them.

A Simple Verification (Contd’4)

```
> cat Lesson1_sqrt_why.pvs

Lesson1_sqrt_why: THEORY
BEGIN
  importing Krak_model

  %% DO NOT EDIT BELOW THIS LINE

  %% Why logic
  sorted_array: [warray[int], int, int -> bool]
  exchange: [warray[int], warray[int], int, int -> bool]
  sub_permut: [int, int, warray[int], warray[int] -> bool]
  permut: [warray[int], warray[int] -> bool]
  array_le: [warray[int], int, int, int -> bool]
  array_ge: [warray[int], int, int, int -> bool]

  ...
```

A Simple Verification (Contd’5)

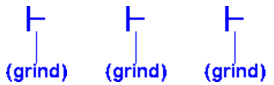
```
  ...

  % Why obligation from file "Lesson1_sqrt.why", characters 457-551
  Lesson1_sqrt_body_po_1: LEMMA
    FORALL (x: int) : x >= (0 :: int) IMPLIES
      FORALL (count: int) : count = (0 :: int) IMPLIES
        FORALL (sum: int) : sum = (1 :: int) IMPLIES
          FORALL (Variant1: int) : FORALL (count1: int) : FORALL (sum1: int) :
          Variant1 = x - sum1 IMPLIES
            count1 >= (0 :: int) AND x >= count1 * count1 AND sum1 =
            (count1 + (1 :: int)) * (count1 + (1 :: int)) IMPLIES
            sum1 <= x IMPLIES
              FORALL (count2: int) : count2 = count1 + (1 :: int) IMPLIES
                FORALL (sum2: int)
                sum2 = sum1 + (2 :: int) * count2 + (1 :: int) IMPLIES
                  count2 >= (0 :: int) AND x >= count2 * count2 AND sum2 =
                  (count2 + (1 :: int)) * (count2 + (1 :: int)) AND
                  zwf_zero(x - sum2, x - sum1)

  ...
```

A Simple Verification (Contd’6)

```
  ...

  % Why obligation from file "Lesson1_sqrt.why", characters 235-558
  Lesson1_sqrt_body_po_2: LEMMA
    FORALL (x: int) : x >= (0 :: int) IMPLIES
      FORALL (count: int) : count = (0 :: int) IMPLIES
        FORALL (sum: int) : sum = (1 :: int) IMPLIES
          FORALL (Variant1: int) : FORALL (count1: int) : FORALL (sum1: int) :
          Variant1 = x - sum1 IMPLIES
            count1 >= (0 :: int) AND x >= count1 * count1 AND sum1 =
            (count1 + (1 :: int)) * (count1 + (1 :: int)) IMPLIES
            sum1 > x IMPLIES
              (FORALL (result: int): (result = count1 IMPLIES
                 result >= (0 :: int) AND result * result <= x AND
                 x < (result + (1 :: int)) * (result + (1 :: int))))

  ...
```

A Simple Verification (Contd'7)

```
  ...

  % Why obligation from file "Lesson1_sqrt.why", characters 288-416
  Lesson1_sqrt_body_po_3: LEMMA
    FORALL (x: int) : x >= (0 :: int) IMPLIES
      FORALL (count: int) : count = (0 :: int) IMPLIES
        FORALL (sum: int) : sum = (1 :: int) IMPLIES
          count >= (0 :: int) AND x >= count * count AND sum =
          (count + (1 :: int)) * (count + (1 :: int))

END Lesson1_sqrt_why

>  pvs Lesson1_sqrt_why.pvs
```



Proving the obligations with PVS (in general, human guidance required).

Verifying Linearch Search

```
package linsearch;
public class Main
{
/*@ public normal_behavior
  @ requires a != null;
  @ assignable \nothing;
  @ ensures
  @ (\result == -1 &&
  @   (\forall int j;
  @     0 <= j && j < a.length;
  @       a[j] != x)) ||
  @ (0<=\result && \result<a.length
  @   && a[\result] == x &&
  @   (\forall int j;
  @     0 <= j && j < \result;
  @     a[j] != x));
  @*/
  public static
  int search(int[] a, int x)
  {
```

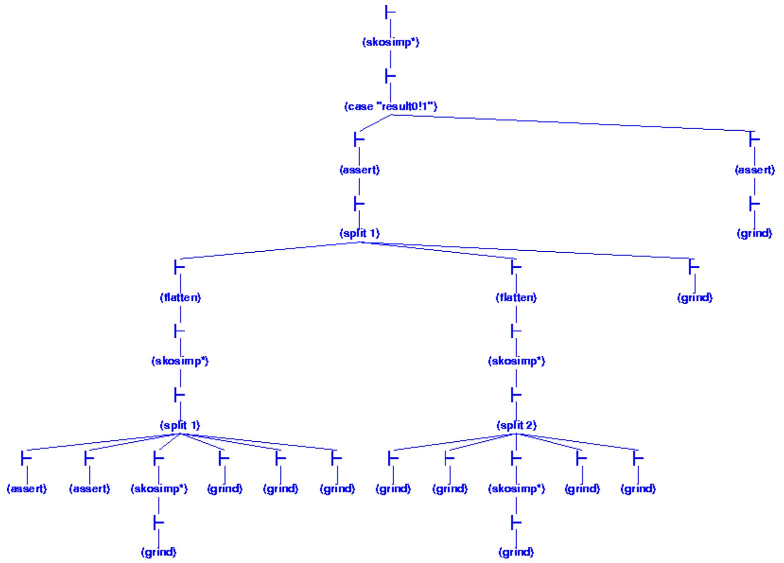
```
int n = a.length;
int i = 0;
int r = -1;
/*@ loop_invariant
  @ a != null && n == a.length &&
  @    0 <= i && i <= n &&
  @ (\forall int j; 0 <= j && j < i-1;
  @      a[j] != x) &&
  @ (i > 0 && r == -1 ==> a[i-1] != x) &&
  @ (r == -1 ||
  @    (r == i-1 && 0 < i && a[r] == x));
  @ decreases n-i;
  @*/
while (i < n && r == -1)
{
  if (a[i] == x) r = i;
  i = i+1;
}
return r;
  }
}
```

Verifying Linearch Search (Contd)

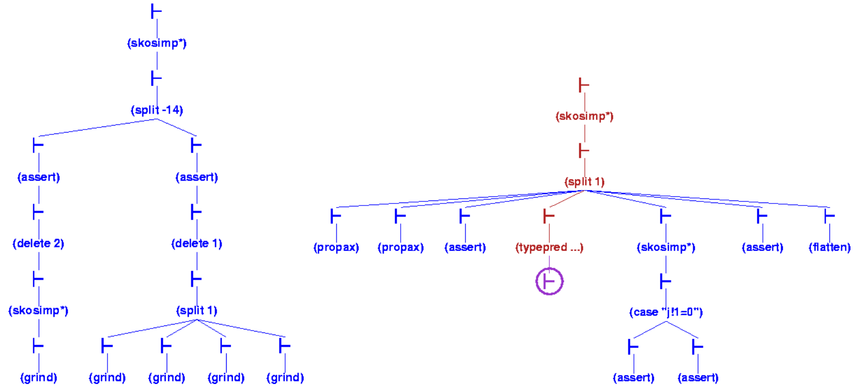
```
Main_search_why: THEORY
BEGIN
  importing Krak_model
  ...
  % Why obligation from file "Main_search.why", characters 380-405
  Main_search_body_po_1: LEMMA
    FORALL (a: value) :
    ...

  % Why obligation from file "Main_search.why", characters 405-405
  Main_search_body_po_2: LEMMA
    FORALL (a: value) :
    ...

  % Why obligation from file "Main_search.why", characters 436-975
  Main_search_body_po_3: LEMMA
    FORALL (a: value) :
    ...
END Main_search_why
```

(Condition generation for PVS fails with Why versions later than 1.6x)

Verifying Linearch Search (Contd'2)



Verifying Linearch Search (Contd’3)



Slight incompleteness in generated PVS model (weak type information).

1. The Krakatoa/Why Tool Suite

2. The KeY Tool

The KeY Tool

- KeY: verification of JavaCard programs.
  - Subset of Java for smartcard applications and embedded systems.
  - Peter Schmidt et al, Universities of Karlsruhe and Koblenz (Germany), Chalmers University (Sweden), 1998–

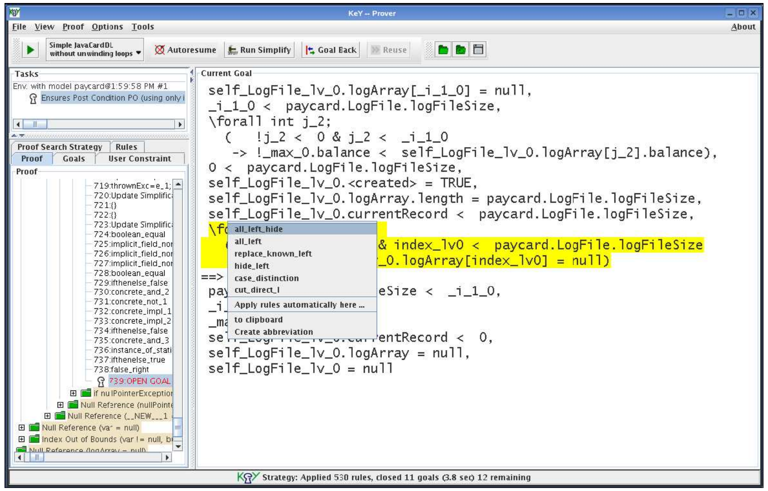
    `http://www.key-project.org`

    Ahrendt et al: “The KeY Tool”, 2005.
- Specification Languages: OCL or JML.
  - Original: OCL (Object Constraint Language), part of UML standard.
  - Later added: JML (Java Modeling Language).
- Logical Framework: Dynamic Logic (DL).
  - Successor/generalization of Hoare Logic.
  - Integrated prover with interfaces to external decision procedures.
    - Simplify, ICS.

We will only deal with the tool’s JML interface “JMLKeY”.

The JMLKeY Prover

```
/zvol/formal/bin/startProver &
```



Env. with model paycard@1:59:58 PM #1

Ensures Post Condition PO (using only i

```
719:thrownExc=e_1;
720:Update Simplific
721:{}
722:{}
723:Update Simplific
724:boolean_equal
725:implicit_field_nor
726:implicit_field_nor
727:implicit_field_nor
728:boolean_equal
729:ifthenelse_false
730:concrete_and_2
731:concrete_not_1
732:concrete_impl_1
733:concrete_impl_2
734:ifthenelse_false
735:concrete_and_3
736:instance_of_stati
737:ifthenelse_true
738:false_right
739:OPEN GOAL
if nullPointerExceptior
Null Reference (nullPointe
Null Reference (__NEW___1
Null Reference (var = null)
Index Out of Bounds (var != null, b
Null Reference (logArray = null)
```

```
self_LogFile_lv_0.logArray[_i_1_0] = null,
_i_1_0 <  paycard.LogFile.logFileSize,
\forall int j_2;
  (   !j_2 <  0 & j_2 <  _i_1_0
   -> !_max_0.balance <  self_LogFile_lv_0.logArray[j_2].balance),
0 <  paycard.LogFile.logFileSize,
self_LogFile_lv_0.<created> = TRUE,
self_LogFile_lv_0.logArray.length = paycard.LogFile.logFileSize,
self_LogFile_lv_0.currentRecord <  paycard.LogFile.logFileSize,
\fo                                  & index_lv0 <  paycard.LogFile.logFileSize
                                     _0.logArray[index_lv0] = null)
==>
pay                                  eSize <  _i_1_0,
_i
_ma
se                                   entRecord <  0,
self_LogFile_lv_0.logArray = null,
self_LogFile_lv_0 = null
```

Strategy: Applied 530 rules, closed 11 goals (3.8 sec) 12 remaining

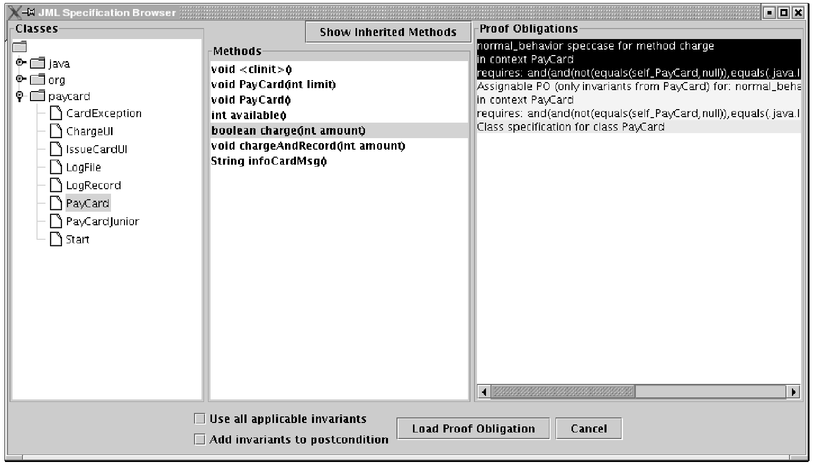
A Simple Example

Engel et al: “KeY Quicktour for JML”, 2005.

```
package paycard;

public class PayCard {

/*@ public instance invariant
  @    log != null
  @ && balance >=0
  @ && limit >0
  @ && unsuccessfulOperations >=0;
  @*/

/*@ spec_public @*/ int limit=1000;
/*@ spec_public @*/
  int unsuccessfulOperations;
/*@ spec_public @*/ int id;
/*@ spec_public @*/ int balance=0;
/*@ spec_public @*/
  protected LogFile log;

...
```

```
/*@
  @ public normal_behavior
  @ requires amount>0 ;
  @ assignable
  @   unsuccessfulOperations, balance;
  @ ensures balance >= \old(balance);
  @*/
public boolean charge(int amount) {
  if (this.balance+amount>=this.limit) {
    this.unsuccessfulOperations++;
      return false;
  } else {
    this.balance=this.balance+amount;
      return true;
  }
}

 ...
}
```

A Simple Example (Contd)

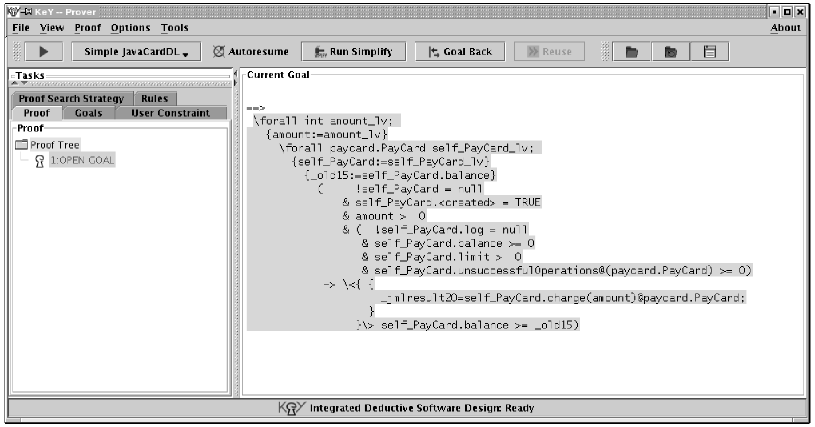


```
void <clinit>()
void PayCard(int limit)
void PayCard()
int available()
boolean charge(int amount)
void chargeAndRecord(int amount)
String infoCardMsg()
```

```
normal_behavior speccase for method charge
in context PayCard
requires: and(and(not(equals(self_PayCard,null)),equals(.java.l
Assignable PO (only invariants from PayCard) for: normal_beha
in context PayCard
requires: and(and(not(equals(self_PayCard,null)),equals(.java.l
Class specification for class PayCard
```

Generate and load the proof obligations.

A Simple Example (Contd’2)



```
==>
 \forall int amount_lv;
   {amount:=amount_lv}
     \forall paycard.PayCard self_PayCard_lv;
       {self_PayCard:=self_PayCard_lv}
         {_old15:=self_PayCard.balance}
           (     !self_PayCard = null
               & self_PayCard.<created> = TRUE
               & amount >  0
               & (  !self_PayCard.log = null
                  & self_PayCard.balance >= 0
                  & self_PayCard.limit >  0
                  & self_PayCard.unsuccessfulOperations@(paycard.PayCard) >= 0)
            -> \<{ {
                     _jmlresult20=self_PayCard.charge(amount)@paycard.PayCard;
                   }
                 }\> self_PayCard.balance >= _old15)
```

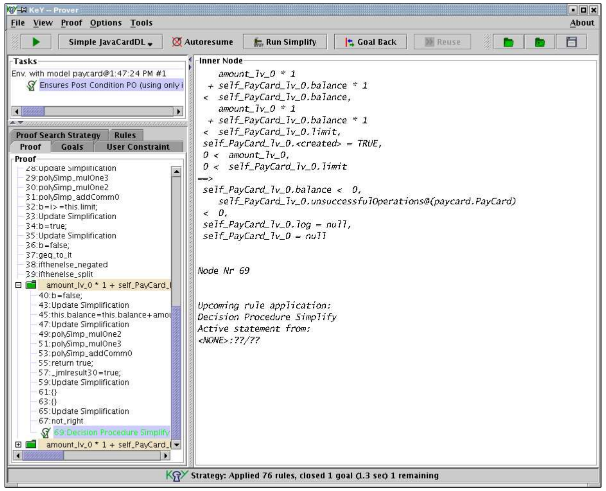
Select the automatic proof strategy “Simple JavaCardDL”.

A Simple Example (Contd’3)

```
==>
 \forall int amount_lv;
   {amount:=amount_lv}
     \forall paycard.PayCard self_PayCard_lv;
       {self_PayCard:=self_PayCard_lv}
         {_old16:=self_PayCard.balance}
           (     !self_PayCard = null
               & self_PayCard.<created>  = TRUE
               & amount >  0
               & (  !self_PayCard.log  = null
                  & self_PayCard.balance  >= 0
                  & self_PayCard.limit >  0
                  & self_PayCard.unsuccessfulOperations@(paycard.PayCard)  >= 0)
            -> \<{ {
                     _jmlresult30=self_PayCard.charge(amount)@paycard.PayCard;
                   }
                 }\> self_PayCard.balance  >= _old16)
```

Press the “Run” button and then “Run Simplify”.

A Simple Example (Contd'4)



Env. with model paycard@1:47:24 PM #1
Ensures Post Condition PO (using only i

```
28:Update Simplification
29:polySimp_mulOne3
30:polySimp_mulOne2
31:polySimp_addComm0
32:b=i>=this.limit;
33:Update Simplification
34:b=true;
35:Update Simplification
36:b=false;
37:geq_to_lt
38:ifthenelse_negated
39:ifthenelse_split
amount_lv_0 * 1 + self_PayCard_l
  40:b=false;
  43:Update Simplification
  45:this.balance=this.balance+amou
  47:Update Simplification
  49:polySimp_mulOne2
  51:polySimp_mulOne3
  53:polySimp_addComm0
  55:return true;
  57:_jmlresult30=true;
  59:Update Simplification
  61:{}
  63:{}
  65:Update Simplification
  67:not_right
  69:Decision Procedure Simplify
amount_lv_0 * 1 + self_PayCard_l
```

```
   amount_lv_0 * 1
 + self_PayCard_lv_0.balance * 1
<  self_PayCard_lv_0.balance,
   amount_lv_0 * 1
 + self_PayCard_lv_0.balance * 1
<  self_PayCard_lv_0.limit,
self_PayCard_lv_0.<created> = TRUE,
0 <  amount_lv_0,
0 <  self_PayCard_lv_0.limit
==>
self_PayCard_lv_0.balance <  0,
   self_PayCard_lv_0.unsuccessfulOperations@(paycard.PayCard)
<  0,
self_PayCard_lv_0.log = null,
self_PayCard_lv_0 = null


Node Nr 69


Upcoming rule application:
Decision Procedure Simplify
Active statement from:
<NONE>:??/??
```

Strategy: Applied 76 rules, closed 1 goal (1.3 sec) 1 remaining

Proof runs through (almost) automatically.

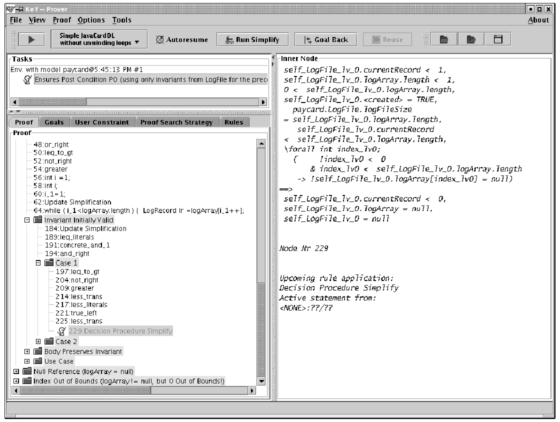
A Loop Example

```
public class LogFile {

/*@ public invariant
  @ logArray.length
  @   == logFileSize &&
  @ currentRecord < logFileSize
  @ && currentRecord >= 0 &&
  @ \nonnullelements(logArray);
  @*/
private /*@ spec_public @*/
  static int logFileSize = 3;
private /*@ spec_public @*/
  int currentRecord;
private /*@ spec_public @*/
  LogRecord[] logArray =
  new LogRecord[logFileSize];

...
```

```
/*@ public normal_behavior
  @ ensures
  @ (\forall int i; 0 <= i && i<logArray.length;
  @   logArray[i].balance <= \result.balance);
  @ diverges true; */
public /*@pure@*/
LogRecord getMaximumRecord(){
  LogRecord max = logArray[0];
  int i=1;
  /*@ loop_invariant
    @   0<=i && i <= logArray.length &&
    @   max!=null &&
    @ (\forall int j; 0 <= j && j<i;
    @   max.balance >= logArray[j].balance);
    @ assignable max, i;
    @*/
  while(i<logArray.length){
    LogRecord lr = logArray[i++];
    if (lr.getBalance() > max.getBalance())
      max = lr;
  }
  return max;
}
```

A Loop Example (Contd)

Proof strategy: "Simple JavaCardDL without unwinding loops".



Env. with model paycard@5:45:13 PM #1
Ensures Post Condition PO (using only invariants from LogFile for the prec

```
48:or_right
50:leq_to_gt
52:not_right
54:greater
56:int i =1;
58:int i;
60:i_1=1;
62:Update Simplification
64:while ( i_1<logArray.length ) {  LogRecord lr =logArray[i_1++];
Invariant Initially Valid
  184:Update Simplification
  189:leq_literals
  191:concrete_and_1
  194:and_right
  Case 1
    197:leq_to_gt
    204:not_right
    209:greater
    214:less_trans
    217:less_literals
    221:true_left
    225:less_trans
    229:Decision Procedure Simplify
  Case 2
Body Preserves Invariant
Use Case
Null Reference (logArray = null)
Index Out of Bounds (logArray != null, but 0 Out of Bounds!)
```

```
self_LogFile_lv_0.currentRecord <  1,
self_LogFile_lv_0.logArray.length <  1,
0 <  self_LogFile_lv_0.logArray.length,
self_LogFile_lv_0.<created> = TRUE,
  paycard.LogFile.logFileSize
= self_LogFile_lv_0.logArray.length,
   self_LogFile_lv_0.currentRecord
<  self_LogFile_lv_0.logArray.length,
\forall int index_lv0;
  (     !index_lv0 <  0
      & index_lv0 <  self_LogFile_lv_0.logArray.length
   -> !self_LogFile_lv_0.logArray[index_lv0] = null)
==>
self_LogFile_lv_0.currentRecord <  0,
self_LogFile_lv_0.logArray = null,
self_LogFile_lv_0 = null


Node Nr 229


Upcoming rule application:
Decision Procedure Simplify
Active statement from:
<NONE>:??/??
```

Various human interactions required (see demo).

Summary

- Various academic approaches to verifying Java(Card) programs.
  - Krakatoa/Why, KeY.
  - Loop: http://www.sos.cs.ru.nl/research/loop/main.html
  - Jack: http://www-sop.inria.fr/everest/soft/Jack/core.html
  - Jive: http://www.sct.ethz.ch/research/jive
- Do not yet scale to verification of large Java applications.
  - General language/program model is too complex.
  - Simplifying assumptions about program may be made.
  - Possibly only special properties may be verified.
- Nevertheless helpful for reasoning on Java in the small.
  - Beyond Hoare calculus on programs in toy languages.
- Enforce clearer understanding of language features.
  - Perhaps constructs with complex reasoning are not a good idea...
- Trend: modularization of reasoning.

In a not too distant future, customers might demand that some critical code is shipped with formal certificates (correctness proofs)...