The RISC ProgramExplorer
Third Status Report

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

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

Goals

An integrated program reasoning environment that provides insight into the semantic essence of a program.

- Is based on the concept of programs as state relations.
  - A program implements a relation on states.
  - A specification describes a relation on states.
  - The program relation must imply the specification relation.
- Addresses various semantic questions.
  - Is a specification satisfiable and not trivial?
  - What is the state relation described by a command/method?
  - What state condition is known at a particular program point?
  - Are methods only called in states that satisfy the methods' preconditions?
  - Does the method meet its specification (assuming that loop invariants hold and termination terms are appropriate)?
  - Do the invariants indeed hold?
  - Are the termination terms indeed appropriate?
- Provides a state-of-the-art graphical user interface.
  - Tight links between syntactic source code and semantic essence.
  - Helps to gain insight as much as possible.

Program Calculus

- Hoare Calculus: $\{\mathtt{x} = a\}\mathtt{x=x*x}\{\mathtt{x} = a^2\}$
  - Pair of *state conditions* "glued together" by a logical constant a.
  - Reasoning based on Hoare triples that mix program and logic.
- Dynamic Logic: $\forall a : \mathtt{x} = a \Rightarrow [\mathtt{x=x*x}]\mathtt{x} = a^2$
  - Two *state conditions* separated by a modality $[\mathtt{x=x*x}]$.
  - Reasoning based on modal formulas that mix program and logic.
- Relational Calculus: $\mathtt{x=x*x}$: $\mathtt{x}' = \mathtt{x}^2$
  - Single *state relation* $\mathtt{x}' = \mathtt{x}^2$.
    - Captures the (denotational) semantics of the command.
  - Reasoning based on classical logic.
    - The command is translated into a classical logical formula.
    - All further reasoning about the command is based on the formula.

Our approach is to use the relational calculus to give programmers *insight*.

Illustration

```
if (n < 0)
  s = -1;                  F1
else {
  var i;
   s = 0;                  F2  } Fs
   i = 1;                  F3  }
   while (i <= n) {
     s = s+i;              F4  } Fb
     i = i+1;              F5  }
   }^{F,T}
}
```

F_s, F_b, F_w, F_v, F_e, F_c

$F :\Leftrightarrow 1 <= \text{var } i <= \text{var } n{+}1 \text{ and var } s = \sum_{j=1}^{\text{var } i\text{-}1} j$

$T := \text{var } n \text{ - var } i + 1$

$$F_c \quad \Leftrightarrow \quad [\text{if old } n < 0 \text{ then var } s = \text{-}1 \text{ else var } s = \sum_{j=1}^{\text{old } n} j]^{\{s\}}$$

Translation into a formula that captures the program's semantic essence.

Program and Specification Language

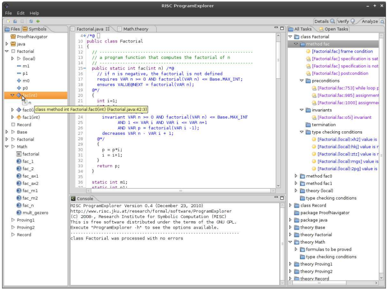
```
public static int fac(int n) /*@
  requires VAR n >= 0 AND factorial(VAR n) <= Base.MAX_INT;
  ensures VALUE@NEXT = factorial(VAR n);
@*/
{
  int i=1;
  int p=1;
  while (i <= n) /*@
    invariant VAR n >= 0 AND factorial(VAR n) <= Base.MAX_INT
          AND 1 <= VAR i AND VAR i <= VAR n+1
          AND VAR p = factorial(VAR i -1);
    decreases VAR n - VAR i + 1;
  @*/
  {
    p = p*i;
    i = i+1;
  }
  return p;
}
```

Theory Language

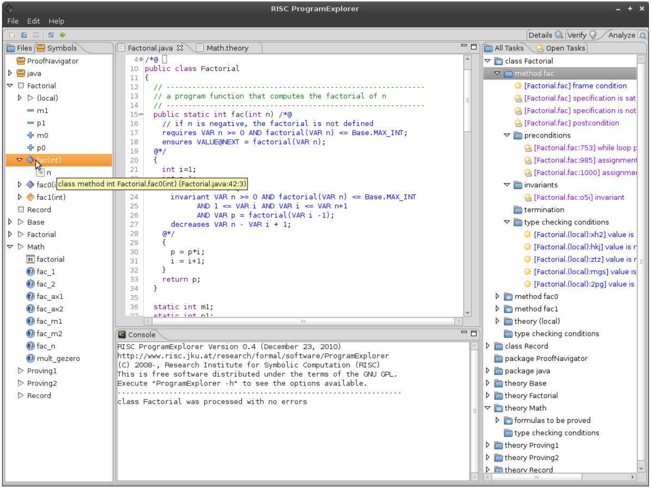
```
theory Math
{
  // an axiomatic specification of the factorial function
  factorial: NAT -> NAT;
  fac_ax1: AXIOM factorial(0) = 1;
  fac_ax2: AXIOM FORALL(n: NAT): factorial(n+1) = (n+1)*factorial(n);

  // somme auxiliary properties of factorial
  fac_1: FORMULA factorial(1) = 1;
  fac_2: FORMULA factorial(2) = 2;
  fac_n: FORMULA
    FORALL(n: NAT): n > 2 => factorial(n) > n;
  fac_m1: FORMULA
    FORALL(n: NAT, m: NAT): n >= m =>
      factorial(n) >= factorial(m);
  fac_m2: FORMULA
    FORALL(n: NAT, m: NAT): n > m AND n >= 2 =>
      factorial(n) > factorial(m);

  // a property of multiplication
  mult_gezero: AXIOM FORALL(n: NAT, m: NAT): n*m >= 0;
}
```

The Software

- Integrated environment built on top of the Eclipse SWT.
  - Provides graphical user interface and editing framework.
- Analyze view.
  - Verification tasks.
    - Type checking conditions.
    - Specification validation.
    - Statement preconditions.
    - Loop invariants.
    - Method frame preservation.
    - Method termination.
    - Method postcondition.
- Verify view.
  - Embeds the RISC ProofNavigator.
- Details view.
  - Logic of a method body.
  - Pre/post-condition reasoning.



Demonstration



Current State and Further Work

- Software in alpha3 status.
  - Almost functionality-complete.
  - Reasonably stable (tested with toy examples only).
  - Classes: ca. 120 ProgramExplorer, 100 ProofNavigator, 300 syntax.
  - Lines of code: about 130K with comments (perhaps 60-70K without).
- Website and user manual.
  - Still presenting the alpha1 status (April 2010).
- Current work:
  - Termination calculus (recursive method measures).

Functionality-complete prototype expected till May 2011.