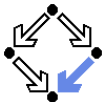


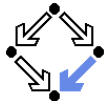
The RISC Program Explorer Third Status Report

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

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



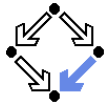
Goals



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.

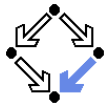
Program Calculus



- **Hoare Calculus:** $\{x = a\}x=x*x\{x = a^2\}$
 - Pair of *state conditions* “glued together” by a logical constant a .
 - Reasoning based on Hoare triples that mix program and logic.
- **Dynamic Logic:** $\forall a : x = a \Rightarrow [x=x*x]x = a^2$
 - Two *state conditions* separated by a modality $[x=x*x]$.
 - Reasoning based on modal formulas that mix program and logic.
- **Relational Calculus:** $x=x*x: x' = x^2$
 - Single *state relation* $x' = x^2$.
 - 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;
else {
  var i;
  s = 0;
  i = 1;
  while (i <= n) {
    s = s+i;
    i = i+1;
  }F,T
}
```

F_1

F_2 } F_s

F_3 }

F_4 } F_b

F_5 }

} F_w

} F_v

} F_e

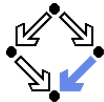
} F_c

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

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

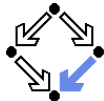
Translation into a formula that captures the program's semantic essence.

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

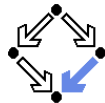


```
theory Math
{
  // an axiomatic specification of the factorial function
  factorial: NAT -> NAT;
  fac_ax1: AXIOM factorial(0) = 1;
  fac_ax2: AXIOM FORALL(n: NAT): factorial(n+1) = (n+1)*factorial(n);

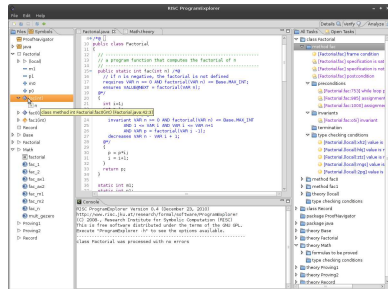
  // somme auxiliary properties of factorial
  fac_1: FORMULA factorial(1) = 1;
  fac_2: FORMULA factorial(2) = 2;
  fac_n: FORMULA
    FORALL(n: NAT): n > 2 => factorial(n) > n;
  fac_m1: FORMULA
    FORALL(n: NAT, m: NAT): n >= m =>
      factorial(n) >= factorial(m);
  fac_m2: FORMULA
    FORALL(n: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

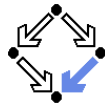
The Software



- **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



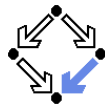
The screenshot displays the RISC Program Explorer interface. The main editor shows the source code for `Factorial.java` with formal annotations. The code includes a class `Factorial` with a static method `fac(int n)`. The annotations specify the function's purpose, preconditions, invariants, and type checking conditions. The verification report on the right lists various conditions that have been checked, such as frame conditions, specifications, preconditions, invariants, and type checking conditions. The console at the bottom shows the version information and a message indicating that the class was processed with no errors.

```
10 public class Factorial
11 {
12 // .....
13 // a program function that computes the factorial of n
14 // .....
15 public static int fac(int n) /*@
16 // if n is negative, the factorial is not defined
17 requires VAR n >= 0 AND factorial(VAR n) <= Base.MAX_INT;
18 ensures VALUE@EXIT = factorial(VAR n);
19 */
20 {
21   int i=i;
22
23
24   invariant VAR n >= 0 AND factorial(VAR n) <= Base.MAX_INT
25     AND 1 <= VAR i AND VAR i <= VAR n-1
26     AND VAR p = factorial(VAR i - 1);
27   decreases VAR n - VAR i + 1;
28
29   @*/
30   p = p*i;
31   i = i+1;
32 }
33 return p;
34 }
35
36 static int m1;
37 static int n1;
```

Verification Report:

- class Factorial
 - method fac
 - [Factorial.fac] frame condition
 - [Factorial.fac] specification is sat
 - [Factorial.fac] specification is not
 - [Factorial.fac] postcondition
 - preconditions
 - [Factorial.fac:753] while loop p
 - [Factorial.fac:985] assignment
 - [Factorial.fac:1000] assignment
 - invariants
 - [Factorial.fac:05] invariant
 - termination
 - type checking conditions
 - [Factorial.local:h2] value is
 - [Factorial.local:h4] value is r
 - [Factorial.local:ztz] value is r
 - [Factorial.local:mrg] value is r
 - [Factorial.local:2pg] value is
- method fac0
- method fac1
- theory (local)
- type checking conditions
- class Record
 - package ProofNavigator
 - package java
 - theory Base
 - theory Factorial
 - theory Math
 - formulas to be proved
 - type checking conditions
 - theory Proving1
 - theory Proving2
 - theory Record

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.