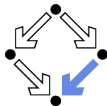


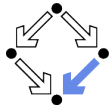
Computer-Assisted Program Reasoning Based on a Relational Semantics of Programs

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

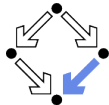
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
<http://www.risc.jku.at>



Program Verification (Classical)



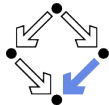
Program Verification (Classical)



P

- Program P

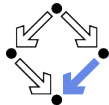
Program Verification (Classical)



$$P \quad + \quad S$$

- Program P
- Specification S

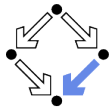
Program Verification (Classical)



$$P^A + S$$

- Program P
- Specification S
- Annotation A

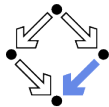
Program Verification (Classical)



$$P^A + S \begin{array}{l} \nearrow \\ \longrightarrow \\ \searrow \end{array} \begin{array}{l} VC_1 \\ \vdots \\ VC_i \\ \vdots \\ VC_n \end{array}$$

- Program P
- Specification S
- Annotation A
- Verification Conditions VC

Program Verification (Classical)

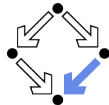


$$P^A + S \begin{array}{l} \nearrow \\ \longrightarrow \\ \searrow \end{array} \begin{array}{l} VC_1 \\ \vdots \\ VC_i \\ \vdots \\ VC_n \end{array}$$

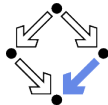
- Program P
- Specification S
- Annotation A
- Verification Conditions VC

Problem: only when proving VC , we learn whether P, A, S “match”.

Soundness of Verification

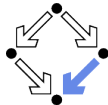


Soundness of Verification

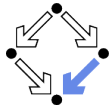


- Program P ... command

Soundness of Verification

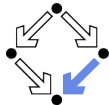


- Program P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$



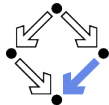
Soundness of Verification

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$
- **Specification** S ... formula



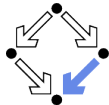
Soundness of Verification

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$
- **Specification** S ... formula
 - Specification semantics $\llbracket S \rrbracket \subseteq State \times State$



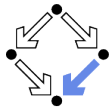
Soundness of Verification

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$
- **Specification** S ... formula
 - Specification semantics $\llbracket S \rrbracket \subseteq State \times State$
- **Verification Condition** VC ... formula



Soundness of Verification

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$
- **Specification** S ... formula
 - Specification semantics $\llbracket S \rrbracket \subseteq State \times State$
- **Verification Condition** VC ... formula
 - Condition semantics $\llbracket VC \rrbracket \in \{\text{true}, \text{false}\}$



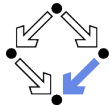
Soundness of Verification

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$
- **Specification** S ... formula
 - Specification semantics $\llbracket S \rrbracket \subseteq State \times State$
- **Verification Condition** VC ... formula
 - Condition semantics $\llbracket VC \rrbracket \in \{\text{true}, \text{false}\}$

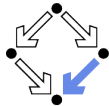
$$\llbracket VC \rrbracket \Rightarrow \forall s, s' \in State : \llbracket P \rrbracket(s, s') \Rightarrow \llbracket S \rrbracket(s, s')$$

Both the program P and the specification S denote state relations.

Program Reasoning (Alternative)



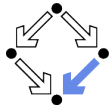
Program Reasoning (Alternative)



P^A

- Annotated program P^A

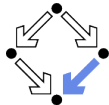
Program Reasoning (Alternative)



$$P^A \longrightarrow F$$

- Annotated program P^A
- Transition formula F

Program Reasoning (Alternative)

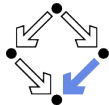


$$P^A \longrightarrow \begin{array}{c} \curvearrowright \\ F \end{array}$$

- Annotated program P^A
- Transition formula F

F represents the “semantic essence” of P^A that is open for investigation.

Program Reasoning (Alternative)

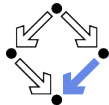


$$P^A \longrightarrow \overset{\Omega}{F} + S$$

- Annotated program P^A
- Transition formula F
- Specification S

F represents the “semantic essence” of P^A that is open for investigation.

Program Reasoning (Alternative)

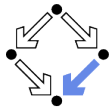


$$P^A \longrightarrow F \overset{\curvearrowright}{+} S \longrightarrow F \Rightarrow S$$

- Annotated program P^A
- Transition formula F
- Specification S
- Verification Condition $F \Rightarrow S$

F represents the “semantic essence” of P^A that is open for investigation.

Program Reasoning (Alternative)

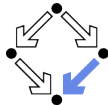


$$P^A \longrightarrow \begin{array}{c} \curvearrowright \\ F \\ \searrow \\ TC_1 \\ \dots \\ TC_n \end{array} + S \longrightarrow F \Rightarrow S$$

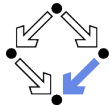
- Annotated program P^A
- Transition formula F
- Specification S
- Verification Condition $F \Rightarrow S$
- Translation Conditions TC

F represents the “semantic essence” of P^A that is open for investigation.

Soundness of Translation

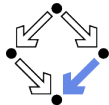


Soundness of Translation

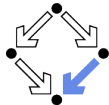


- Program P ... command

Soundness of Translation

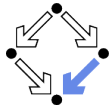


- Program P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$



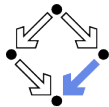
Soundness of Translation

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq \text{State} \times \text{State}$
- **Formula** F ... formula



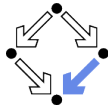
Soundness of Translation

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$
- **Formula** F ... formula
 - Formula semantics $\llbracket F \rrbracket \subseteq State \times State$



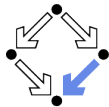
Soundness of Translation

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq \text{State} \times \text{State}$
- **Formula** F ... formula
 - Formula semantics $\llbracket F \rrbracket \subseteq \text{State} \times \text{State}$
- **Translation Conditions** TC ... formula



Soundness of Translation

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$
- **Formula** F ... formula
 - Formula semantics $\llbracket F \rrbracket \subseteq State \times State$
- **Translation Conditions** TC ... formula
 - Condition semantics $\llbracket TC \rrbracket \in \{\text{true}, \text{false}\}$



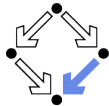
Soundness of Translation

- **Program** P ... command
 - Program semantics $\llbracket P \rrbracket \subseteq State \times State$
- **Formula** F ... formula
 - Formula semantics $\llbracket F \rrbracket \subseteq State \times State$
- **Translation Conditions** TC ... formula
 - Condition semantics $\llbracket TC \rrbracket \in \{\text{true}, \text{false}\}$

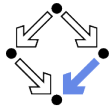
$$\llbracket TC \rrbracket \Rightarrow \forall s, s' \in State : \llbracket P \rrbracket(s, s') \Rightarrow \llbracket F \rrbracket(s, s')$$

Formula F captures the state relation denoted by program P .

Example



Example



```
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 }

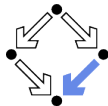
F_w

F_v

F_e

F_c

Example



```
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

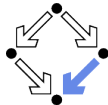
F_v

F_e

F_c

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

Example



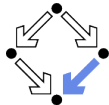
```
if (n < 0)
  s = -1;
else {
  var i;
  s = 0;
  i = 1;
  while (i <= n) {
    s = s+i;
    i = i+1;
  }F, T
}
```

F_1 }
 F_2 } F_s }
 F_3 } }
 F_4 } } F_v } F_e } F_c
 F_5 } } F_w } }

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

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

Example



```
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_3 } F_s

F_4
 F_5 } F_b

} F_w

} F_v

} F_e

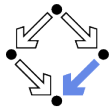
} F_c

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

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

The program computes for non-negative n the sum from 1 to n .

Judgements



- Core judgement $c : [f_r]_{g,h}^{xs}$

$\forall s, s' \in \text{State} :$

$$\llbracket g \rrbracket \wedge \llbracket h \rrbracket (s) \Rightarrow$$

$$(\llbracket c \rrbracket (s, s') \Rightarrow \llbracket f_r \rrbracket (s, s')) \wedge \forall x \in \text{Variable} \setminus x_s : \llbracket x \rrbracket (s) = \llbracket x \rrbracket (s').$$

Command c , state relation f_r , set of program variables x_s , state-independent condition g , state condition h .

- Derived judgements $\text{pre}(c, f_q) = f_p$ and $\text{post}(c, f_p) = f_q$

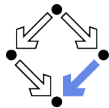
$$\forall s, s' \in \text{State} : \llbracket g \rrbracket \wedge \llbracket h \rrbracket (s) \Rightarrow (\llbracket f_p \rrbracket (s) \wedge \llbracket c \rrbracket (s, s') \Rightarrow \llbracket f_q \rrbracket (s'))$$

$$\frac{c : [f]_{g,h}^{xs}}{\text{pre}(c, f_q) = \forall xs : f[xs/\text{var } xs] \Rightarrow f_q[xs/\text{old } xs]}$$

$$\frac{c : [f]_{g,h}^{xs}}{\text{post}(c, f_p) = \exists xs : f_p[xs/\text{old } xs] \wedge f[xs/\text{old } xs, \text{old } xs/\text{var } xs]}$$

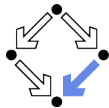
A generalization of pre/post-condition reasoning.

Translation Calculus



$$\begin{array}{c}
 \frac{e \simeq_h t}{x = e : [\text{var } x = t]_{\text{true}, h}^{\{x\}}} \\
 c : [f]_{g, h}^{XS} \\
 \hline
 \{\text{var } x; c\} : [\exists x_0, x_1 : f[x_0/\text{old } x, x_1/\text{var } x]]_{g, \forall x : h[x/\text{old } x]}^{XS \setminus X} \\
 \\
 \frac{c_1 : [f_1]_{g_1, h_1}^{XS} \quad c_2 : [f_2]_{g_2, h_2}^{XS} \quad \text{pre}(c_1, h_2) = h_3}{\{c_1; c_2\} : [\exists ys : f_1[ys/\text{var } xs] \wedge f_2[ys/\text{old } xs]]_{g_1 \wedge g_2, h_1 \wedge h_3}^{XS}} \\
 \\
 \frac{e \simeq_h f_e \quad c_1 : [f_1]_{g_1, h_1}^{XS} \quad c_2 : [f_2]_{g_2, h_2}^{XS}}{\text{if } (e) \text{ then } c_1 \text{ else } c_2 : [\text{if } f_e \text{ then } f_1 \text{ else } f_2]_{g_1 \wedge g_2, h \wedge \text{if } f_e \text{ then } h_1 \text{ else } h_2}^{XS}} \\
 \\
 \frac{e \simeq_h f_e \quad c : [f_c]_{g_c, h_c}^{XS}}{g \equiv \forall xs, ys, zs : f[xs/\text{old } xs, ys/\text{var } xs] \wedge f_e[ys/\text{old } xs] \wedge f_c[ys/\text{old } xs, zs/\text{var } xs] \Rightarrow h[ys/\text{old } xs] \wedge f[xs/\text{old } xs, zs/\text{var } xs]} \\
 \hline
 \text{while } (e)^{f, t} c : [f \wedge \neg f_e[\text{var } xs/\text{old } xs]]_{g_c \wedge g, h \wedge f[\text{old } xs/\text{var } xs]}^{XS}
 \end{array}$$

Termination



Termination judgement $c \downarrow_{g_c} f_c$

$$\forall s \in \text{State} : \llbracket g_c \rrbracket \Rightarrow \llbracket f_c \rrbracket(s) \Rightarrow \langle\langle c \rangle\rangle(s)$$

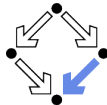
$$x = e \downarrow_{\text{true}} \text{true} \qquad \frac{c \downarrow_g f}{\{\text{var } x; c\} \downarrow_g \forall x : f}$$

$$\frac{c_1 \downarrow_{g_1} f_1 \quad c_2 \downarrow_{g_2} f_2 \quad \text{pre}(c_1, f_2) = f_3}{\{c_1; c_2\} \downarrow_{g_1 \wedge g_2} f_1 \wedge f_3}$$

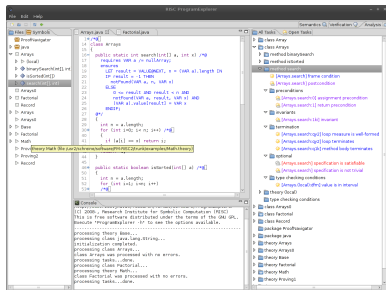
$$\frac{e \simeq_h f_e \quad c_1 \downarrow_{g_1} f_1 \quad c_2 \downarrow_{g_2} f_2}{\text{if } (e) \text{ then } c_1 \text{ else } c_2 \downarrow_{g_1 \wedge g_2} \text{if } f_e \text{ then } f_1 \text{ else } f_2}$$

$$\frac{e \simeq_h f_e \quad c : [f_c]_{g_c, h_c}^{XS} \quad c \downarrow_{g_t} f_t}{g \equiv \forall xs, ys, zs : f[xs/\text{old } xs, ys/\text{var } xs] \wedge f_e[ys/\text{old } xs] \wedge f_c[ys/\text{old } xs, zs/\text{var } xs] \Rightarrow f_t[ys/\text{old } xs] \wedge 0 \leq t[zs/\text{old } xs] < t[ys/\text{old } xs]} \\ \text{while } (e)^{f, t} c \downarrow_g \wedge g_t t \geq 0$$

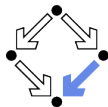
The RISC ProgramExplorer



- An integrated program reasoning environment.
 - Programming language *MiniJava*.
 - Theory/specification language in the style of PVS/CVC.
 - Semi-automatic proving assistant *RISC ProofNavigator*.
- Analysis view (verification tasks).
 - Type checking conditions.
 - Statement preconditions.
 - Loop invariants.
 - Method frame preservation.
 - Method termination.
 - Method postcondition.
- Semantics view.
 - Semantics of a method body.
 - Pre/post-condition reasoning.
- Verification view.
 - Proof construction and management.



Analysis View

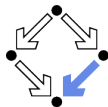


The screenshot displays the RISC Program Explorer interface. The main window is titled "RISC Program Explorer" and shows the "Analysis" view for the "Arrays.java" file. The left sidebar contains a "Files" pane with a tree view showing the project structure, including "ProofNavigator", "java", "Arrays", "Factorial", and "Math". The central pane shows the source code of the "Arrays" class, with the "search" method selected. The right sidebar shows a "Semantics" pane with a tree view of analysis results, including "class Array", "class Arrays", "method binarySearch", "method isSorted", and "method search". The "method search" node is expanded, showing various analysis results such as "frame condition", "postcondition", "preconditions", "invariants", "termination", "optional", and "type checking conditions". The bottom pane shows the "Console" output, which includes the RISC logo, copyright information, and a list of processing tasks completed for the "Arrays" and "Factorial" classes.

```
14 class Arrays
15 {
16   public static int search(int[] a, int x) /*Q
17     requires VAR a != nullArray;
18     ensures
19       LET result = VALUE@NEXT, n = (VAR a).length IN
20       IF result = -1 THEN
21         notFound(VAR a, n, VAR x)
22       ELSE
23         0 <= result AND result < n AND
24         notFound(VAR a, result, VAR x) AND
25         (VAR a).value[result] = VAR x
26       ENDF;
27   g*/
28   {
29     int n = a.Length;
30     for (int i=0; i<n; i++) /*Q
40     {
41       if (a[i] == x) return i;
42     }
43   }
44 }
45
46 public static boolean isSorted(int[] a) /*Q
50 {
51   int n = a.Length;
52   for (int i=1; i<n; i++)
53     /*Q
```

Console
(C) 2008, Research Institute for Symbolic Computation (RISC)
This is free software distributed under the terms of the GNU GPL.
Execute "ProgramExplorer -h" to see the options available.
.....
processing theory Base...
processing class java.lang.String...
initialization completed.
processing class Arrays...
-class Arrays was processed with no errors.
processing tasks...done.
processing class Factorial...
processing theory Math...
class Factorial was processed with no errors.
processing tasks...done.

Semantics View



The screenshot shows the RISC Program Explorer interface with the Semantics View selected. The left pane displays the source code for the `search` function, and the right pane displays its formal semantics.

Searching.search

```
requires  $\neg \text{old } a.\text{null}$   
ensures let  
  result = value@next,  
  n = var a.length  
in  
  ( if result = (-1) then  
     $\forall i \in \mathbb{Z}: 0 \leq i \wedge i < n \Rightarrow \text{var } a.\text{value}[i] \neq \text{var } x$   
  else  
     $0 \leq \text{result} \wedge \text{result} < n$   
     $\wedge$   
     $(\forall i \in \mathbb{Z}: 0 \leq i \wedge i < \text{result} \Rightarrow \text{var } a.\text{value}[i] \neq \text{var } x)$   
     $\wedge$   
    var a.value[result] = var x  
  endif )
```

```
public static int search(int[] a, int x) /*@  
  requires NOT OLD a.null;  
  ensures LET result = VALUE@NEXT, n = VAR a.length  
  */  
{  
  int n = a.length;  
  int r = -1;  
  int i = 0;  
  while ( i < n && r == -1) /*@  
    invariant NOT VAR a.null AND VAR n = VAR a.length  
    decreases IF OLD r = (-1) THEN OLD n - OLD i ELSE  
  */  
{  
  if (a[i] == x)
```

Body Knowledge

[Show Original Formulas]

Pre-State Knowledge

```
 $\neg \text{old } a.\text{null}$ 
```

Effects

executes: false, continues: false, breaks: false, returns: true
variables: -; exceptions:-

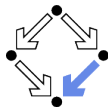
Transition Relation

```
 $(\exists n \in \text{Base.int}, r \in \text{Base.int}, i \in \text{Base.int}, n \in \text{Base.int}, r \in \text{Base.int}, i_0 \in \text{Base.int}:$   
   $n = \text{old } a.\text{length} \wedge (i_0 \geq n \vee r \neq (-1)) \wedge 0 \leq i_0 \wedge i_0 \leq n$   
   $\wedge$   
   $(\forall i \in \mathbb{Z}: 0 \leq i \wedge i < i_0 \Rightarrow \text{old } a.\text{value}[i] \neq \text{old } x)$   
   $\wedge$   
   $(r = (-1) \vee r = i_0 \wedge i_0 < n \wedge \text{old } a.\text{value}[r] = \text{old } x) \wedge \text{value}@next = r \wedge r = r \wedge \text{in} = i_0$   
   $\wedge$   
   $\text{nn} = n) \wedge \neg \text{old } a.\text{null}$   
   $\wedge$   
  returns@next
```

Termination Condition

```
 $\forall n \in \text{Base.int}, r \in \text{Base.int}:$   
  executes@now  $\wedge n = \text{old } a.\text{length} \wedge r = (-1) \Rightarrow ( \text{if } r = (-1) \text{ then } n \text{ else } 0 \text{ endif} ) \geq 0$ 
```

Verification View



The screenshot displays the RISC Program Explorer interface. On the left, the 'Proof Tree' shows a sequence of steps: [wpf]: scatter, [wpf]: proved (CVCL), [wpf]: proved (CVCL), [wpf]: auto ty, and [i4]: proved (CVCL). The main area is divided into 'Proof State' and 'View Declarations'.

Proof State:

```
( $\forall i \text{ nat} : i < x \Rightarrow \text{newBvecOnArray}(x)\text{value}[i] = \text{false}$ )
```

Formula [i4]

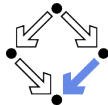
Auto ty: Automatically Instantiate Formula.
instantiate [] in ty: Instantiate Variable(s) in Formula
expand [] in ty: Expand Definition(s) in Formula
goal ty: Make Formula Goal
skip ty: Flip Formula

```
new Array(x) ≠ null Array  $\wedge$  x = new Array(x).length  
 $\wedge$   
( $\forall i \text{ nat} : i < x \Rightarrow \text{null} = \text{new Array}(x)\text{value}[i]$ )
```

View Declarations:

Input/Output:
Value null: [#r:INT#].
Value newObject: INT->[#r:INT#].
Type Array = [#value:ARRAY [MIN_INT..MAX_INT] OF [#r:INT#], length:[0..MAX_INT]#].
Value nullArray: [#value:ARRAY [MIN_INT..MAX_INT] OF [#r:INT#], length:[0..MAX_INT]#].
Value newArray: [0..MAX_INT]->[#value:ARRAY [MIN_INT..MAX_INT] OF [#r:INT#], length:[0..MAX_INT]#].
Value x: [0..MAX_INT].
Value y: [#value:ARRAY [MIN_INT..MAX_INT] OF [#r:INT#], length:[0..MAX_INT]#].
Value i: [0..MAX_INT].
Formula goal_ already has a (skeleton) proof (proof status: trusted, closed, absolute)
Proof state [wpf] is closed by decision procedure.
Proof state [wpf] is closed by decision procedure.
Proof state [i4] is closed by decision procedure.
Proof replay successful.
Use 'proof goal_' to see proof.

Current State and Further Work



- **Development 2008–2011.**
 - RISC ProofNavigator 2005–2007.
 - Uses CVCL as a validity checker.
 - About 140K lines of commented Java code.
- **October 2011: first release.**
 - Tested with various examples.
 - Use in “formal methods” course starting with fall 2011.
- **Future work:**
 - Simplification of transition relations.
 - Improvement of proof automation.

<http://www.risc.jku.at/research/formal/software/ProgramExplorer>