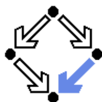
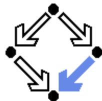
Reasoning about Programs as State Relations

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



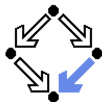


1. **Core Idea**

2. The Semantics of Programs

3. Outlook

Reasoning about Programs



How to describe the behavior of programs?

- Hoare calculus: triples of statements.

  $\{x = a\}$ `x=x+1` $\{x = a + 1\}$

- Dynamic logic: formulas with modalities.

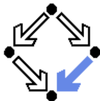
  $\forall a : x = a \Rightarrow [\texttt{x=x+1}]\, x = a + 1$

- My approach: formulas with primed variables.

  `x=x+1`: $x' = x + 1$

Core idea: translate programs to state relations described by formulas in classical predicate logic with classical rules of reasoning.

Example



- Program

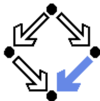
```
x=x+1; if (x == 0) return 1; else y = x*x
```

- Formula

```
x’ = ADD32(x,1) AND
IF x’ = 0
  THEN next.returns AND next.value = 1 AND y’ = y
  ELSE next.executes AND y’ = MULT32(x,x)
```

Effect of command fully described in a classical logical framework.

Semantics of Commands



Take command C, context c, states s and s'.

- Transition relation $[\![_]\!]$:

  $$[\![C]\!]^c(s, s') \Leftrightarrow \ldots$$

  - $[\![C]\!]^c$ defines a relation on states.
  - Which state transitions are possible by execution of C in c?

- Termination condition $[\![_]\!]_{\mathrm{T}}$:

  $$[\![C]\!]^c_{\mathrm{T}}(s) \Leftrightarrow \ldots$$

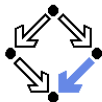
  - $[\![C]\!]^c_{\mathrm{T}}$ defines a condition on states.
  - For which prestates must the execution of C yield a poststate?

- Constraint:

  $$\forall s : [\![C]\!]^c_{\mathrm{T}}(s) \Rightarrow \exists s' : [\![C]\!]^c(s, s').$$

The semantics of a command is described by a pair of a state relation and a state condition.

Semantics of Formulas



Take logical formula F, context c, environment e, states s and s'

- Formula semantics $[\![_]\!]$:

  $[\![F]\!]^c(e)(s, s') \Leftrightarrow \ldots$

  - $[\![F]\!]^c(e)$ defines a relation on states.

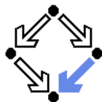
- Formula semantics $[\![_]\!]$:

  $[\![F]\!]^c(e)(s) \Leftrightarrow \ldots [\![F]\!]^c(e)(s, s)$

  - $[\![F]\!]^c(e)$ defines a condition on states.

We use a classical logical language for specifying state relations and state conditions.

Judgements



Various kinds of judgements that describe properties of commands.

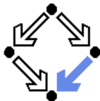
$$C : F \;\Leftrightarrow$$
$$\forall c, s, s', e : \ldots \Rightarrow$$
$$[\![C]\!]^c(s, s') \Rightarrow [\![F]\!]^c(e)(s, s')$$

$$C \downarrow F \;\Leftrightarrow$$
$$\forall c, s, e : \ldots \Rightarrow$$
$$[\![F]\!]^c(s) \Rightarrow [\![C]\!]^c_{\mathrm{T}}(e)(s)$$

...

We have a calculus for deriving only true judgements.

Specifications of Commands



$\texttt{requires } F_C \texttt{ ensures } F_R \; \{C\}$

- Verification of partial correctness:
  1. Derivation of transition relation F of C

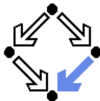
     $C : F$ (bottom-up application of calculus)
  2. Proof that transition relation meets specification:

     $\forall c, es, s' : \ldots [\![F]\!]^c(e)(s,s') \Rightarrow [\![F_C \texttt{ => } F_R]\!]^c(e)(s,s')$
- Verification of total correctness:
  - Derivation of termination condition F_C for C.

    $C \downarrow F_C$ (top-down application of calculus)
- ...

As usual, the details are a bit more complicated.

Pre/Postcondition Reasoning

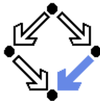


$$\frac{C : [\,F\,]_{I_1,\ldots,I_n} \quad \ldots}{\begin{array}{l}\textsc{pre}(C, Q) = \\ \quad \texttt{FORALL } \$J_1, \ldots, \$J_n \texttt{:} \\ \qquad F[\$J_1/I_1\texttt{'}, \ldots, \$J_n/I_n\texttt{'}] \texttt{ => } Q[\$J_1/I_1, \ldots, \$J_n/I_n]\end{array}}$$

$$\frac{C : [\,F\,]_{I_1,\ldots,I_n} \quad \ldots}{\begin{array}{l}\textsc{post}(C, P) = \\ \quad \texttt{EXISTS } \$J_1, \ldots, \$J_n \texttt{:}\ P[\$J_1/I_1, \ldots, \$J_n/I_n] \texttt{ AND} \\ \qquad F[\$J_1/I_1, \ldots, \$J_n/I_n, I_1/I_1\texttt{'}, \ldots, I_n/I_n\texttt{'}]\end{array}}$$

Pre/post-conditions can be computed from transition relations.

Advantages of Framework



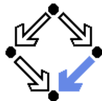
- Translation of programs to classical logical formulas.
  - Well-understood semantics and rules of reasoning.
    - Only logic most people writing programs have ever learned.
  - May be investigated by programmer to yield insight.
    - Should be instructive (after appropriate simplification).
  - Application of general-purpose tools possible.
    - Simplification and verification.
- Relational semantical framework.
  - Simpler and more abstract semantics than functional framework.

    No need for unique poststates, no need to be constructive.
  - Programs and specifications on same level.

    Not functions versus relations.
  - Non-determinism covered.

    Prepared for integration into concurrency framework.

Framework fully covers control flow interruptions.

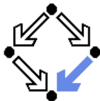


1. Core Idea

2. The Semantics of Programs

3. Outlook

What are Programs?



$P \in$ Program, $Ms \in$ Methods, $M \in$ Method, $S \in$ Specification, $C \in$ Command, $E \in$ Expression, $I, J, K \in$ Identifier, $F \in$ Formula.

$P ::= Ms\ S\ \{C\}.$

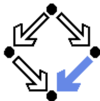
$Ms ::= \text{␣} \mid Ms\ M.$

$M ::= \texttt{method}\ I_m(J_1, \ldots, J_p)\ S\ \{C\}.$

$S ::= \texttt{writesonly}\ I_1, \ldots, I_n\ \texttt{throwsonly}\ K_1, \ldots, K_m$
$\quad \texttt{requires}\ F_C\ \texttt{ensures}\ F_R$

$C ::=$
$\quad I = E \mid \texttt{var}\ I;\ C \mid \texttt{var}\ I\texttt{=}E;\ C \mid C_1; C_2$
$\quad \texttt{if}\ (E)\ C \mid \texttt{if}\ (E)\ C_1\ \texttt{else}\ C_2$
$\quad \texttt{while}\ (E)\ C$
$\quad \texttt{continue} \mid \texttt{break} \mid \texttt{return}\ E \mid \texttt{throw}\ I\ E$
$\quad \texttt{try}\ C_1\ \texttt{catch}(I_k\ I_v)\ C_2 \mid$
$\quad I_r = I_m(E_1, \ldots, E_p).$

What is a State?



- Approximation 1: a state is a store.

  $$\textit{Store} := \mathrm{Variable} \to \mathrm{Value}$$
  $$\textit{State} := \mathrm{Store}$$

  - Sufficient for modeling basic programs.

- Approximation 2: a store also holds control data.

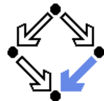
  $$\textit{Flag} := \{\mathrm{E}, \mathrm{C}, \mathrm{B}, \mathrm{R}, \mathrm{T}\}$$
  $$\textit{Key} := \textit{Value}$$
  $$\textit{Control} := \textit{Flag} \times \textit{Key} \times \textit{Value}$$
  $$\textit{State} := \textit{Store} \times \textit{Control}$$

  - Needed for modeling programs with control flow interruptions.

    `continue`, `break`, `return`, `throw`.

First model can be in some sense "embedded" into second one.

What is the Meaning Program Identifiers?



How are they related to variables in the store?

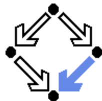
- Approximation 0: identifiers = variables.
  - Used in basic presentations of program semantics and reasoning.
- Approximation 1: identifiers denote variables.
  - Different identifiers denote different variables.
  - Not all variables need to be denoted by identifiers.

    $[\![I]\!] \in \text{Variable}, [\![I]\!] = \ldots$
    $\text{DifferentVars} \Leftrightarrow \forall I, J \in \text{Identifier} : I \neq J \Rightarrow [\![I]\!] \neq [\![J]\!]$
- Approximation 2: given a context, identifiers denote variables.
  - Contexts may change.
  - In each context, different identifiers denote different variables.

    $[\![I]\!] : \mathit{Context} \rightarrow \mathit{Variable}, [\![I]\!]^c = \mathit{view}(c)(I)$
    $\text{DifferentVars}(c) \Leftrightarrow \forall I, J \in \text{Identifier} : I \neq J \Rightarrow [\![I]\!]^c \neq [\![J]\!]^c$

For the moment, we stick to a semantics without contexts.

The Algebra of States



$$store : State \rightarrow Store, store(s, c) = s$$

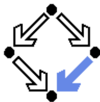
$$read : State \times \mathrm{Identifier} \rightarrow Value$$
$$read(s, I) = store(s)([\![I]\!])$$

$$write : State \times \mathrm{Identifier} \times Value \rightarrow State$$
$$write(s, I, v) = (store(s)[[\![I]\!] \mapsto v], control(s))$$

$$writes(s, I_1, v_1, \ldots, I_n, v_n) \equiv$$
$$\quad (store(s)[[\![I_1]\!] \mapsto v_1] \ldots [[\![I_n]\!] \mapsto v_n], control(s))$$

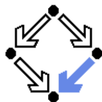
$$s_0 = s_1 \text{ EXCEPT } I_1, \ldots, I_n \equiv$$
$$\quad \forall I \in \mathrm{Identifier} : I \neq I_1 \wedge \ldots \wedge I \neq I_n \Rightarrow read(s_0, I) = read(s_1, I)$$

Semantics of Commands



$$
\begin{aligned}
&[\![\, I \texttt{ = } E \,]\!](s,s') \Leftrightarrow \\
&\quad s' = write(s, I, [\![\, E \,]\!](s)) \\
&[\![\, \texttt{var } I\texttt{;}\ C \,]\!](s,s') \Leftrightarrow \\
&\quad \exists s_0, s_1 \in State: \\
&\qquad s_0 = s \text{ EXCEPT } I \wedge [\![\, C \,]\!](s_0,s_1) \wedge \\
&\qquad\quad s' = write(s_1, I, read(s, I)) \\
&[\![\, \texttt{var } I\texttt{=}E\texttt{;}\ C \,]\!](s,s') \Leftrightarrow \\
&\quad \exists s_0, s_1 \in State: \\
&\qquad s_0 = write(s, I, [\![\, E \,]\!](s)) \wedge [\![\, C \,]\!](s_0,s_1) \wedge \\
&\qquad\quad s' = write(s_1, I, read(s, I)) \\
&[\![\, C_1\texttt{;}C_2 \,]\!](s,s') \Leftrightarrow \\
&\quad \exists s_0 \in State: [\![\, C_1 \,]\!](s,s_0) \wedge [\![\, C_2 \,]\!](s_0,s') \\
&[\![\, \texttt{if (}E\texttt{)}\ C \,]\!](s,s') \Leftrightarrow \\
&\quad \text{IF } [\![\, E \,]\!](s) = \text{TRUE THEN } [\![\, C \,]\!](s,s') \text{ ELSE } s' = s \\
&[\![\, \texttt{if (}E\texttt{)}\ C_1 \texttt{ else } C_2 \,]\!](s,s') \Leftrightarrow \\
&\quad \text{IF } [\![\, E \,]\!](s) = \text{TRUE THEN } [\![\, C_1 \,]\!](s,s') \text{ ELSE } [\![\, C_2 \,]\!](s,s')
\end{aligned}
$$

Semantics of Loops



For the moment, we stick to a semantics without interruptions.

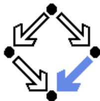
$$StateFunction := State \rightarrow Value$$

$$\begin{array}{l} finiteExecution : \\ \quad \mathbb{P}(\mathbb{N} \times State^{\infty} \times State \times StateFunction \times StateRelation) \\ finiteExecution(k, t, s, E, C) \Leftrightarrow \\ \quad t(0) = s \wedge \forall i \in \mathbb{N}_k : E(t(i)) = \text{TRUE} \wedge C(t(i), t(i+1)) \end{array}$$

$$\begin{array}{l} [\![\texttt{while (}E\texttt{) } C]\!](s, s') \Leftrightarrow \\ \quad \exists k \in \mathbb{N}, t \in State^{\infty} : \\ \quad\quad finiteExecution(k, t, s, [\![E]\!], [\![C]\!]) \wedge \\ \quad\quad [\![E]\!](t(k)) \neq \text{TRUE} \wedge t(k) = s' \end{array}$$

The semantics of a loop can be described by a finite sequence of states.

The Algebra of Control Data

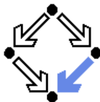


$$control : State \rightarrow Control, control(s, c) = c$$

$$flag : Control \rightarrow Flag, flag(f, k, v) = f$$
$$key : Control \rightarrow Key, key(f, k, v) = k$$
$$value : Control \rightarrow Value, value(f, k, v) = v$$

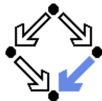
$$executes : \mathbb{P}(Control), executes(c) \Leftrightarrow flag(c) = \mathrm{E}$$
$$continues : \mathbb{P}(Control), continues(c) \Leftrightarrow flag(c) = \mathrm{C}$$
$$breaks : \mathbb{P}(Control), breaks(c) \Leftrightarrow flag(c) = \mathrm{B}$$
$$returns : \mathbb{P}(Control), returns(c) \Leftrightarrow flag(c) = \mathrm{R}$$
$$throws : \mathbb{P}(Control), throws(c) \Leftrightarrow flag(c) = \mathrm{T}$$

The Algebra of Control Data



$$
\begin{aligned}
&\mathit{execute} : \mathit{State} \rightarrow \mathit{State}\\
&\mathit{execute}(s) =\\
&\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{E}, \mathit{key}(c), \mathit{value}(c)))\\
&\mathit{continue} : \mathit{State} \rightarrow \mathit{State}\\
&\mathit{continue}(s) =\\
&\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{C}, \mathit{key}(c), \mathit{value}(c)))\\
&\mathit{break} : \mathit{State} \rightarrow \mathit{State}\\
&\mathit{break}(s) =\\
&\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{B}, \mathit{key}(c), \mathit{value}(c)))\\
&\mathit{return} : \mathit{State} \times \mathit{Value} \rightarrow \mathit{State}\\
&\mathit{return}(s, v) =\\
&\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{R}, \mathit{key}(c), v))\\
&\mathit{throw} : \mathit{State} \times \mathit{Key} \times \mathit{Value} \rightarrow \mathit{State}\\
&\mathit{throw}(s, k, v) =\\
&\quad \text{LET } c = \mathit{control}(s) \text{ IN } (\mathit{store}(s), (\mathrm{B}, k, v))
\end{aligned}
$$

Semantics of Commands with Interruptions

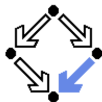


The semantics of some commands has to be redefined.

$$
\begin{array}{l}
[\![\texttt{var } I;\ C]\!](s, s') \Leftrightarrow \\
\quad \exists s_0, s_1 \in \mathit{State} : \\
\qquad s_0 = s \text{ EXCEPT } I \wedge \mathit{control}(s_0) = \mathit{control}(s) \wedge \\
\qquad [\![C]\!](s_0, s_1) \wedge s' = \mathit{write}(s_1, I, \mathit{read}(s, I))
\end{array}
$$

$$
\begin{array}{l}
[\![C_1 ; C_2]\!](s, s') \Leftrightarrow \\
\quad \exists s_0 \in \mathit{State} : \\
\qquad [\![C_1]\!](s, s_0) \wedge \\
\qquad \text{IF } \mathit{executes}(\mathit{control}(s_0)) \text{ THEN } [\![C_2]\!](s_0, s') \text{ ELSE } s' = s_0
\end{array}
$$

Semantics of Commands with Interruptions



$$[\![\, \texttt{continue} \,]\!](s, s') \Leftrightarrow s' = \mathit{continue}(s)$$

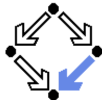
$$[\![\, \texttt{break} \,]\!](s, s') \Leftrightarrow s' = \mathit{break}(s)$$

$$[\![\, \texttt{return}\ E \,]\!](s, s') \Leftrightarrow s' = \mathit{return}(s, [\![\, E \,]\!](s))$$

$$[\![\, \texttt{throw}\ I\ E \,]\!](s, s') \Leftrightarrow s' = \mathit{throw}(s, I, [\![\, E \,]\!](s))$$

$$
\begin{array}{l}
[\![\, \texttt{try}\ C_1\ \texttt{catch}(I_k\ I_v)\ C_2 \,]\!](s, s') \Leftrightarrow \\
\quad \exists s_0, s_1, s_2 \in \mathit{State} : \\
\qquad [\![\, C_1 \,]\!](s, s_0) \wedge \\
\qquad \text{IF } \mathit{throws}(\mathit{control}(s_0)) \wedge \mathit{key}(\mathit{control}(s_0)) = I_k \text{ THEN} \\
\qquad\quad s_1 = \mathit{write}(\mathit{execute}(s_0), I_v, \mathit{value}(\mathit{control}(s_0))) \wedge \\
\qquad\quad [\![\, C_2 \,]\!](s_1, s_2) \wedge \\
\qquad\quad s' = \mathit{write}(s_2, I_v, \mathit{read}(s_0, I_v)) \\
\qquad \text{ELSE } s' = s_0
\end{array}
$$

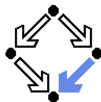
Semantics of Loops with Interruptions



$$
\begin{array}{l}
\mathit{finiteExecution} : \\
\mathbb{P}(\mathbb{N} \times \mathit{State}^{\infty} \times \mathit{State}^{\infty} \times \mathit{State} \times \mathit{StateFunction} \times \mathit{StateRelation}) \\
\mathit{finiteExecution}(k, t, u, s, E, C) \Leftrightarrow \\
\quad t(0) = s \land u(0) = s \land \\
\quad \forall i \in \mathbb{N}_k : \\
\quad\quad \neg \mathit{breaks}(\mathit{control}(u(i))) \land \mathit{executes}(\mathit{control}(t(i))) \land \\
\quad\quad E(t(i)) = \text{TRUE} \land C(t(i), u(i+1)) \land \\
\quad\quad \text{IF } \mathit{continues}(\mathit{control}(u(i+1))) \lor \mathit{breaks}(\mathit{control}(u(i+1))) \\
\quad\quad\quad \text{THEN } t(i+1) = \mathit{execute}(u(i+1)) \\
\quad\quad\quad \text{ELSE } t(i+1) = u(i+1)
\end{array}
$$

$$
\begin{array}{l}
[\![\texttt{while (}E\texttt{) } C]\!](s, s') \Leftrightarrow \\
\quad \exists k \in \mathbb{N}, t, u \in \mathit{State}^{\infty} : \\
\quad\quad \mathit{finiteExecution}(k, t, u, s, [\![E]\!], [\![C]\!]) \land \\
\quad\quad ([\![E]\!](t(k)) \neq \text{TRUE} \lor \\
\quad\quad\quad \neg(\mathit{executes}(\mathit{control}(u(k))) \lor \\
\quad\quad\quad\quad \mathit{continues}(\mathit{control}(u(k))))) \land \\
\quad\quad t(k) = s'
\end{array}
$$

Expressions and Interruptions

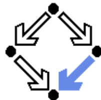


What happens on the evaluation of 1/0?

- Mathematics: "undefined"
  - A value about which nothing is known.
- Programs: "abortion".
  - Nowadays: "division by zero exception".

For the moment, we stick to the mathematical view.

The Algebra of Contexts



$$\begin{aligned}
&\mathit{View} = \text{Identifier} \to \mathit{Variable}\\
&\mathit{Space} = \mathbb{P}(\mathit{Variable})^\infty\\
&\mathit{Context} = \mathit{View} \times \mathit{Space}
\end{aligned}$$

$$\begin{aligned}
&\mathit{context} : \mathit{View} \times \mathit{Space} \to \mathit{Context}, \mathit{context}(v,s) = \langle v, s\rangle\\
&\mathit{view} : \mathit{Context} \to \mathit{View}, \mathit{view}(v,s) = v\\
&\mathit{space} : \mathit{Context} \to \mathit{Space}, \mathit{space}(v,s) = s
\end{aligned}$$

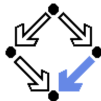
$$\begin{aligned}
&\mathit{range} : \mathit{Context} \to \mathit{Space} \to \mathbb{P}(\mathit{Variable})\\
&\mathit{range}(c) = \text{range}(\mathit{view}(c)) \cup \mathit{space}(c)
\end{aligned}$$

$$\begin{aligned}
&\mathit{take} : \mathit{Space} \to (\mathit{Variable} \times \mathit{Space})\\
&\mathit{take}(s) = \text{LET } x = \text{SUCH } x : x \in s \text{ IN } \langle x, s \backslash \{x\}\rangle
\end{aligned}$$

$$\begin{aligned}
&\mathit{push} : \mathit{Context} \times \text{Identifier} \to \mathit{Context}\\
&\mathit{push}(c, I) = \text{LET } \langle v, s\rangle = c, \langle x, s'\rangle = \mathit{take}(s) \text{ IN } \langle v[I \mapsto x], s'\rangle\\
&\mathit{push}(c, I_1, \ldots, I_n) \equiv \mathit{push}(\ldots \mathit{push}(c, I_1) \ldots, I_n)
\end{aligned}$$

A context consists of a view (mapping of identifiers to variables) and a space (a set of unassigned variables).

Behavior of Contexts



$$\forall c \in Context, I \in \text{Identifier}: \\ \quad DifferentVariables(c) \Rightarrow DifferentVariables(push(c, I))$$

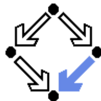
$$\forall c \in Context, I \in \text{Identifier}: \\ \quad range(push(c, I)) = range(c) \backslash \{[\![I]\!]^c\}$$

$$\forall c \in Context, I, J \in \text{Identifier}: \\ \quad DifferentVariables(c) \Rightarrow [\![I]\!]^{push(c,I)} \neq [\![J]\!]^c$$

$$\forall c \in Context, I, J \in \text{Identifier}: \\ \quad DifferentVariables(c) \wedge I \neq J \Rightarrow [\![J]\!]^{push(c,I)} = [\![J]\!]^c$$

Some useful properties for reasoning about contexts.

Contexts and States



$$
\begin{aligned}
&\mathit{read} : \mathit{State} \times \mathrm{Identifier} \times \mathrm{Context} \rightarrow \mathit{Value} \\
&\mathit{read}(s, I)^c = \mathit{store}(s)([\![I]\!]^c)
\end{aligned}
$$

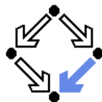
$$
\begin{aligned}
&\mathit{write} : \mathit{State} \times \mathrm{Identifier} \times \mathit{Value} \times \mathrm{Context} \rightarrow \mathit{State} \\
&\mathit{write}(s, I, v)^c = \langle \mathit{store}(s)[\, [\![I]\!]^c \mapsto v \,], \mathit{control}(s) \rangle
\end{aligned}
$$

$$
\mathit{writes}(s, I_1, v_1, \ldots, I_n, v_n)^c \equiv \mathit{write}(\ldots \mathit{write}(s, I_1, v_1)^c \ldots, I_n, v_n)^c
$$

$$
\begin{aligned}
&s_0 = s_1 \text{ EXCEPT}^c \; I_1, \ldots, I_n \;\equiv \\
&\quad \forall I \in \mathrm{Identifier} : I \neq I_1 \wedge \ldots \wedge I \neq I_n \;\Rightarrow \\
&\qquad \mathit{read}(s_0, I)^c = \mathit{read}(s_1, I)^c
\end{aligned}
$$

Reading and writing stores now depends on the context.

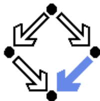
Commands with Contexts



$$
\begin{array}{l}
[\![\, I = E \,]\!]^c(s, s') \Leftrightarrow \\
\quad s' = write(s, I, [\![\, E \,]\!]^c(s))^c \\
[\![\, \texttt{var}\ I;\ C \,]\!]^c(s, s') \Leftrightarrow \\
\quad \text{LET } c' = push(c, I) \text{ IN } [\![\, C \,]\!]^{c'}(s, s') \\
[\![\, \texttt{var}\ I{=}E;\ C \,]\!]^c(s, s') \Leftrightarrow \\
\quad \text{LET } c' = push(c, I), s_0 = write(s, I, [\![\, E \,]\!]^c(s))^{c'} \text{ IN } [\![\, C \,]\!]^{c'}(s_0, s') \\
[\![\, \texttt{try}\ C_1\ \texttt{catch(}I_k\ I_v\texttt{)}\ C_2 \,]\!]^c(s, s') \Leftrightarrow \\
\quad \exists s_0, s_1 \in State : \\
\qquad [\![\, C_1 \,]\!]^c(s, s_0) \wedge \\
\qquad \text{IF } throws(control(s_0)) \wedge key(control(s_0)) = I_k \text{ THEN} \\
\qquad\quad \text{LET } c' = push(c, I_v) \text{ IN} \\
\qquad\quad s_1 = write(execute(s_0), I_v, value(control(s_0)))^{c'} \wedge \\
\qquad\quad [\![\, C_2 \,]\!]^{c'}(s_1, s') \\
\qquad \text{ELSE } s' = s_0
\end{array}
$$

Semantics of local declarations can be simplified.

Semantics of Method Calls

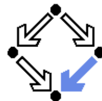


A command is executed within a *method environments*.

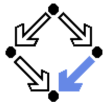
$$
\begin{array}{l}
\mathit{StateCondition} := \mathbb{P}(\mathit{State}) \\
\mathit{Behavior} := \mathit{Value}^* \to (\mathit{StateRelation} \times \mathit{StateCondition}) \\
\mathit{MethodEnv} := \mathrm{Identifier} \to (\mathit{View} \times \mathit{Behavior})
\end{array}
$$

$$
\begin{array}{l}
[\![\ I_r = I_m(E_1, \ldots, E_p)\ ]\!]^{c,me}(s, s') \Leftrightarrow \\
\quad \text{LET } \langle v, b \rangle = me(I_m) \text{ IN} \\
\quad \text{LET } \langle r, t \rangle = b^{context(v, space(c))}([\![\ E_1\ ]\!]^c(s), \ldots, [\![\ E_p\ ]\!]^c(s)) \text{ IN} \\
\quad \exists s_0 \in State : r(s, s_0) \land \\
\qquad \text{IF } throws(control(s_0)) \\
\qquad\quad \text{THEN } s' = s_0 \\
\qquad\quad \text{ELSE } s' = write(s_0, I_r, value(control(s_0)))^c
\end{array}
$$

Semantics of Method Declarations



$$
\begin{array}{l}
[\![\texttt{method}\ I_m\texttt{(}J_1,\ldots,J_p\texttt{)}\ S\ \texttt{\{}C\texttt{\}}]\!]^v(me) = \\
\quad \text{LET} \\
\quad\quad b \in \mathit{Behavior} \\
\quad\quad b^c(v_1,\ldots,v_p) = \\
\quad\quad\quad \text{LET} \\
\quad\quad\quad\quad c' = \mathit{push}(c, J_1,\ldots,J_p) \\
\quad\quad\quad\quad r \in \mathit{StateRelation} \\
\quad\quad\quad\quad r(s,s') \Leftrightarrow \\
\quad\quad\quad\quad\quad \exists s_0 : \mathit{State} : \\
\quad\quad\quad\quad\quad\quad [\![C]\!]^{c',me}(\mathit{writes}(s, J_1, v_1,\ldots,J_p,v_p)^{c'}, s_0) \wedge \\
\quad\quad\quad\quad\quad\quad s' = \text{IF } \mathit{throws}(\mathit{control}(s_0)) \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{THEN } s_0 \text{ ELSE } \mathit{executes}(s_0) \\
\quad\quad\quad\quad t \in \mathit{StateCondition} \\
\quad\quad\quad\quad t(s) \Leftrightarrow [\![C]\!]^{c',me}_{\mathrm{T}}(\mathit{writes}(s, J_1, v_1,\ldots,J_p,v_p)^{c'}) \\
\quad\quad\quad \text{IN } \langle r, t \rangle \\
\quad \text{IN } me[\, I \mapsto \langle v, b \rangle \,]
\end{array}
$$

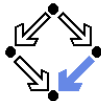


1. Core Idea

2. The Semantics of Programs

3. Outlook

Outlook



- A calculus for $C : F$.
- A calculus for $C \downarrow F$.
- Auxiliary calculi.
  - Preconditions, postconditions, assertions.
- Dealing with "undefined expressions".
  - A calculus for $C \checkmark F$.
- The overall verification framework.
- Dealing with recursive methods.

Stay tuned for more to come in this seminar.