Semantyka i weryfikacja programów

Bartosz Klin

(slajdy Andrzeja Tarleckiego)

Instytut Informatyki
Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski

`http://www.mimuw.edu.pl/~klin` pok. 5680

`klin@mimuw.edu.pl`

Strona tego wykładu:

`http://www.mimuw.edu.pl/~klin/sem19-20.html`

Program Semantics & Verification

Bartosz Klin

(slides courtesy of Andrzej Tarlecki)

Institute of Informatics
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw

`http://www.mimuw.edu.pl/~klin` office: 5680

`klin@mimuw.edu.pl`

This course: `http://www.mimuw.edu.pl/~klin/sem19-20.html`

Working example

For a while, we will work with a trivial iterative programming language:

$$\boxed{\textsc{Tiny}}$$

- simple arithmetic expressions
- simple boolean expressions
- simple statements (assignment, conditional, loop)

Syntactic categories

- *numerals*

$$\boxed{N \in \mathbf{Num}}$$

with syntax given by:

$$N ::= 0 \mid 1 \mid 2 \mid \cdots$$

- *variables*

$$\boxed{x \in \mathbf{Var}}$$

with syntax given by:

$x ::= \cdots$ *sequences of letters and digits beginning with a letter* $\cdots$

- *(arithmetic) expressions*

$$\boxed{e \in \mathbf{Exp}}$$

with syntax given by:

$$e ::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2$$

- *boolean expressions*

$$\boxed{b \in \mathbf{BExp}}$$

with syntax given by:

$$b ::= \mathbf{true} \mid \mathbf{false} \mid e_1 \leq e_2 \mid \neg b' \mid b_1 \wedge b_2$$

- *statements*

$$\boxed{S \in \mathbf{Stmt}}$$

with syntax given by:

$$S ::= x := e \mid \mathbf{skip} \mid S_1 ; S_2 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \mid \mathbf{while}\ b\ \mathbf{do}\ S'$$

Before we move on

(to the semantics)

The definition of syntax, like:

- *(arithmetic) expressions*

$$\boxed{e \in \mathbf{Exp}}$$

with syntax given by:

$$e ::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2$$

implies that all the expressions are of one of the forms given above, all these forms are distinct, and all the expressions can be built by using the above constructs consecutively.

Things can be defined and proved by

(STRUCTURAL) INDUCTION

Structural induction

Given a property $P(\_)$ of expressions:

IF

- $P(N)$, for all $N \in \mathbf{Num}$
- $P(x)$, for all $x \in \mathbf{Var}$
- $P(e_1 + e_2)$ follows from $P(e_1)$ and $P(e_2)$, for all $e_1, e_2 \in \mathbf{Exp}$
- $P(e_1 * e_2)$ follows from $P(e_1)$ and $P(e_2)$, for all $e_1, e_2 \in \mathbf{Exp}$
- $P(e_1 - e_2)$ follows from $P(e_1)$ and $P(e_2)$, for all $e_1, e_2 \in \mathbf{Exp}$

THEN

- $P(e)$ for all $e \in \mathbf{Exp}$.

Inductive definitions

Free variables in expressions $FV(e) \subset \mathbf{Var}$:

$$\begin{aligned}
FV(N) &= \emptyset \\
FV(x) &= \{x\} \\
FV(e_1 + e_2) &= FV(e_1) \cup FV(e_2) \\
FV(e_1 * e_2) &= FV(e_1) \cup FV(e_2) \\
FV(e_1 - e_2) &= FV(e_1) \cup FV(e_2)
\end{aligned}$$

Fact: *For each expression $e \in \mathbf{Exp}$, the set $FV(e)$ of its free variables is finite.*

Semantic categories

Easy things first:

- *boolean values*

$$\mathbf{Bool} = \{\mathbf{tt}, \mathbf{ff}\}$$

- *integers*

$$\mathbf{Int} = \{0, 1, -1, 2, -2, \ldots\}$$

with the obvious semantic function:

$$\mathcal{N} : \mathbf{Num} \to \mathbf{Int}$$

$$\mathcal{N}[\![0]\!] = 0$$
$$\mathcal{N}[\![1]\!] = 1$$
$$\mathcal{N}[\![2]\!] = 2$$
$$\ldots$$

BTW: $\_[\![\_]\!]$ is just a semantic function application, with $[\![\ ]\!]$ used to separate syntactic phrases from the semantic context.

Valuations of variables

- *states* (for now: total functions from $\mathbf{Var}$ to $\mathbf{Int}$)

  $$s \in \mathbf{State} = \mathbf{Var} \to \mathbf{Int}$$

  - lookup (of the value of a variable x in a state s) is function application

    $$s\;x$$

  - update a state: $s' = s[y \mapsto n]$

    $$s'\;x = \begin{cases} s\;x & \text{if } x \neq y \\ n & \text{if } x = y \end{cases}$$

Semantics of expressions

$$\mathcal{E} : \mathbf{Exp} \to (\mathbf{State} \to \mathbf{Int})$$

defined in the obvious way:

$$\begin{aligned}
\mathcal{E}[\![N]\!]\ s &= \mathcal{N}[\![N]\!] \\
\mathcal{E}[\![x]\!]\ s &= s\ x \\
\mathcal{E}[\![e_1 + e_2]\!]\ s &= \mathcal{E}[\![e_1]\!]\ s + \mathcal{E}[\![e_2]\!]\ s \\
\mathcal{E}[\![e_1 * e_2]\!]\ s &= \mathcal{E}[\![e_1]\!]\ s * \mathcal{E}[\![e_2]\!]\ s \\
\mathcal{E}[\![e_1 - e_2]\!]\ s &= \mathcal{E}[\![e_1]\!]\ s - \mathcal{E}[\![e_2]\!]\ s
\end{aligned}$$

BTW: Higher-order functions will be used very frequently!

No further warnings!

Semantics of boolean expressions

$$\mathcal{B}\colon \mathbf{BExp} \to (\mathbf{State} \to \mathbf{Bool})$$

defined in the obvious way:

$$\begin{aligned}
\mathcal{B}[\![\mathbf{true}]\!]\, s &= \mathbf{tt} \\
\mathcal{B}[\![\mathbf{false}]\!]\, s &= \mathbf{ff} \\
\mathcal{B}[\![e_1 \leq e_2]\!]\, s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{E}[\![e_1]\!]\, s \leq \mathcal{E}[\![e_2]\!]\, s \\ \mathbf{ff} & \text{if } \mathcal{E}[\![e_1]\!]\, s \not\leq \mathcal{E}[\![e_2]\!]\, s \end{cases} \\
\mathcal{B}[\![\neg b]\!]\, s &= \begin{cases} \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]\, s = \mathbf{tt} \\ \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]\, s = \mathbf{ff} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]\, s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]\, s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]\, s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]\, s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]\, s = \mathbf{ff} \end{cases}
\end{aligned}$$

Semantics of statements

This will be given in various styles to illustrate various approaches to formal semantics.

Consider the previous definitions as auxiliary

Before we move on

(to the semantics of statements)

Fact: *The meaning of expression depends only on the valuation of its free variables: for any* $e \in \mathbf{Exp}$ *and* $s, s' \in \mathbf{State}$, *if* $s\,x = s'\,x$ *for all* $x \in FV(e)$ *then*

$$\mathcal{E}[\![e]\!]s = \mathcal{E}[\![e]\!]s'.$$

Proof in a moment. . .

Exercise: *Formulate (and prove) this property for boolean expressions.*

Proof

By structural induction:

- for $N \in \mathbf{Num}$, $\mathcal{E}[\![N]\!]\ s = \mathcal{N}[\![N]\!]$
  $$= \mathcal{E}[\![N]\!]\ s'$$

- for $x \in \mathbf{Var}$, $\mathcal{E}[\![x]\!]\ s = s\ x$
  $$\begin{aligned} &= s'\ x && \text{(since } x \in FV(x)\text{)} \\ &= \mathcal{E}[\![x]\!]\ s' \end{aligned}$$

- for $e_1, e_2 \in \mathbf{Exp}$, $\mathcal{E}[\![e_1 + e_2]\!]\ s = \mathcal{E}[\![e_1]\!]\ s + \mathcal{E}[\![e_2]\!]\ s$
  $$\begin{aligned} &= \mathcal{E}[\![e_1]\!]\ s' + \mathcal{E}[\![e_2]\!]\ s' && \text{(by the inductive hypothesis,} \\ & && \text{since } FV(e_1) \subseteq FV(e_1 + e_2), \\ & && \text{and similarly for } e_2\text{)} \\ &= \mathcal{E}[\![e_1 + e_2]\!]\ s' \end{aligned}$$

- ...

Referential transparency...

Substitution of e' for x in e results in $e[e'/x]$:

$$\begin{aligned}
N[e'/x] &= N \\
x'[e'/x] &= \begin{cases} e' & \text{if } x = x' \\ x' & \text{if } x \neq x' \end{cases} \\
(e_1 + e_2)[e'/x] &= e_1[e'/x] + e_2[e'/x] \\
(e_1 * e_2)[e'/x] &= e_1[e'/x] * e_2[e'/x] \\
(e_1 - e_2)[e'/x] &= e_1[e'/x] - e_2[e'/x]
\end{aligned}$$

Prove:

$$\boxed{\mathcal{E}[\![e[e'/x]]\!]\, s = \mathcal{E}[\![e]\!]\, s[x \mapsto \mathcal{E}[\![e']\!]\, s]}$$

Operational semantics

small-step semantics

Overall idea:

- define *configurations*: $\gamma \in \Gamma$
- indicate which of them are *terminal*: $\mathrm{T} \subseteq \Gamma$
- define a *(one-step) transition relation*: $\Rightarrow \subseteq \Gamma \times \Gamma$
  - for $\gamma \in \mathrm{T}$, typically $\gamma \not\Rightarrow$
- study *computations*: (finite or infinite) sequences of configurations

$$\gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1}, \ldots,$$

such that $\gamma_i \Rightarrow \gamma_{i+1}$, written as:

$$\boxed{\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_i \Rightarrow \gamma_{i+1} \Rightarrow \cdots}$$

Computations

Computations may be:

– terminating: $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$, $\gamma_n \in \mathrm{T}$

– blocking: $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$, $\gamma_n \notin \mathrm{T}$ and $\gamma_n \not\Rightarrow$

– infinite (looping): $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots$

Moreover:

- $\gamma \Rightarrow^k \gamma'$ for $k \geq 0$, if there is a computation $\gamma = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_k = \gamma'$
- $\gamma \Rightarrow^* \gamma'$ if $\gamma \Rightarrow^k \gamma'$ for some $k \geq 0$

BTW: $\Rightarrow^* \subseteq \Gamma \times \Gamma$ is the least reflexive and transitive relation that contains $\Rightarrow$.

TINY: operational semantics

Configurations: $\Gamma = (\mathbf{Stmt} \times \mathbf{State}) \cup \mathbf{State}$

Terminal configurations: $\mathrm{T} = \mathbf{State}$

Transition relation contains only:

$$\langle x := e, s\rangle \Rightarrow s[x \mapsto (\mathcal{E}[\![e]\!]\ s)]$$

$$\langle \mathbf{skip}, s\rangle \Rightarrow s$$

$$\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle \quad \text{if } \langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle$$

$$\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle \quad \text{if } \langle S_1, s\rangle \Rightarrow s'$$

$$\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow \langle S_1, s\rangle \quad \text{if } \mathcal{B}[\![b]\!]\ s = \mathbf{tt}$$

$$\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow \langle S_2, s\rangle \quad \text{if } \mathcal{B}[\![b]\!]\ s = \mathbf{ff}$$

$$\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow \langle S; \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \quad \text{if } \mathcal{B}[\![b]\!]\ s = \mathbf{tt}$$

$$\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow s \quad \text{if } \mathcal{B}[\![b]\!]\ s = \mathbf{ff}$$

Some properties

Fact: $\textsc{Tiny}$ *is deterministic, i.e.: for each configuration* $\langle S, s\rangle$

$$\textit{if } \langle S, s\rangle \Rightarrow \gamma_1 \textit{ and } \langle S, s\rangle \Rightarrow \gamma_2 \textit{ then } \gamma_1 = \gamma_2.$$

Proof: By structural induction on S.

Fact: *In* $\textsc{Tiny}$*, for each configuration* $\langle S, s\rangle$ *there is exactly one maximal computation starting in* $\langle S, s\rangle$*.*

Another proof technique:

Induction on the length of computation

Some properties

Fact: *If* $\langle S_1; S_2, s\rangle \Rightarrow^k s'$ *then* $\langle S_1, s\rangle \Rightarrow^{k_1} \hat{s}$ *and* $\langle S_2, \hat{s}\rangle \Rightarrow^{k_2} s'$*, for some* $\hat{s} \in \mathbf{State}$ *and* $k_1, k_2 > 0$ *such that* $k = k_1 + k_2$*.*

Proof: By induction on k:

$k = 0$: OK

$k > 0$: Then $\langle S_1; S_2, s\rangle \Rightarrow \gamma \Rightarrow^{k-1} s'$. By the definition of the transitions, two possibilities only:

- $\gamma = \langle S_2, \hat{s}\rangle$, where $\langle S_1, s\rangle \Rightarrow \hat{s}$. OK
- $\gamma = \langle S_1'; S_2, s''\rangle$, where$\langle S_1, s\rangle \Rightarrow \langle S_1', s''\rangle$. By the inductive hypothesis then, $\langle S_1', s''\rangle \Rightarrow^{k_1} \hat{s}$ and $\langle S_2, \hat{s}\rangle \Rightarrow^{k_2} s'$, for some $\hat{s} \in \mathbf{State}$ and $k_1, k_2 > 0$ such that $k - 1 = k_1 + k_2$. OK

Fact: *Further context does not influence computation:*

If $\langle S_1, s\rangle \Rightarrow^k \langle S_1', s'\rangle$ *then* $\langle S_1; S_2, s\rangle \Rightarrow^k \langle S_1'; S_2, s'\rangle$*;*
if $\langle S_1, s\rangle \Rightarrow^k s'$ *then* $\langle S_1; S_2, s\rangle \Rightarrow^k \langle S_2, s'\rangle$*.*

Some variants

- instead of the current rule for $:=$:

  $$\langle x := e, s\rangle \Rightarrow \langle \mathbf{skip}, s[x \mapsto (\mathcal{E}[\![e]\!]\ s)]\rangle$$

- instead of the current rules for **if**:

  $$\begin{array}{ll}
\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow \langle S_1', s'\rangle & \text{if } \mathcal{B}[\![b]\!]\ s = \mathbf{tt} \text{ and } \langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle \\
\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow s' & \text{if } \mathcal{B}[\![b]\!]\ s = \mathbf{tt} \text{ and } \langle S_1, s\rangle \Rightarrow s' \\
\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow \langle S_2', s'\rangle & \text{if } \mathcal{B}[\![b]\!]\ s = \mathbf{ff} \text{ and } \langle S_2, s\rangle \Rightarrow \langle S_2', s'\rangle \\
\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow s' & \text{if } \mathcal{B}[\![b]\!]\ s = \mathbf{ff} \text{ and } \langle S_2, s\rangle \Rightarrow s'
\end{array}$$

- similarly for **while**, the first case

- instead of the current rules for **while**:

  $$\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow \langle \mathbf{if}\ b\ \mathbf{then}\ (S; \mathbf{while}\ b\ \mathbf{do}\ S)\ \mathbf{else\ skip}, s\rangle$$

- ...

Non-deterministic computation

Sometimes non-determinism arises naturally. For example, in an operational semantics for arithmetic expressions, we would likely have:

$$\langle e_1 + e_2, s\rangle \Rightarrow \langle e_1' + e_2, s\rangle \quad \text{if } \langle e_1, s\rangle \Rightarrow \langle e_1', s\rangle$$

$$\langle e_1 + e_2, s\rangle \Rightarrow \langle e_1 + e_2', s\rangle \quad \text{if } \langle e_2, s\rangle \Rightarrow \langle e_2', s\rangle$$

$$\langle N_1 + N_2, s\rangle \Rightarrow \langle M, s\rangle \quad \text{if } N_1 + N_2 = M$$

This does not specify the order of computation (perhaps good!), and the semantics is non-deterministic.

But in this case, non-determinism "does not matter"!

Confluence

Consider any set Γ and a relation $\Rightarrow \subseteq \Gamma \times \Gamma$.

Definition: $\Rightarrow$ *is weakly confluent if for every* $\gamma, \gamma_1, \gamma_2 \in \Gamma$ *such that*

$$\gamma \Rightarrow \gamma_1 \qquad \gamma \Rightarrow \gamma_2$$

there exists $\gamma_3 \in \Gamma$ *such that*

$$\gamma_1 \Rightarrow^* \gamma_3 \qquad \gamma_2 \Rightarrow^* \gamma_3.$$

$\Rightarrow$ *is confluent if for every* $\gamma, \gamma_1, \gamma_2 \in \Gamma$ *such that*

$$\gamma \Rightarrow^* \gamma_1 \qquad \gamma \Rightarrow^* \gamma_2$$

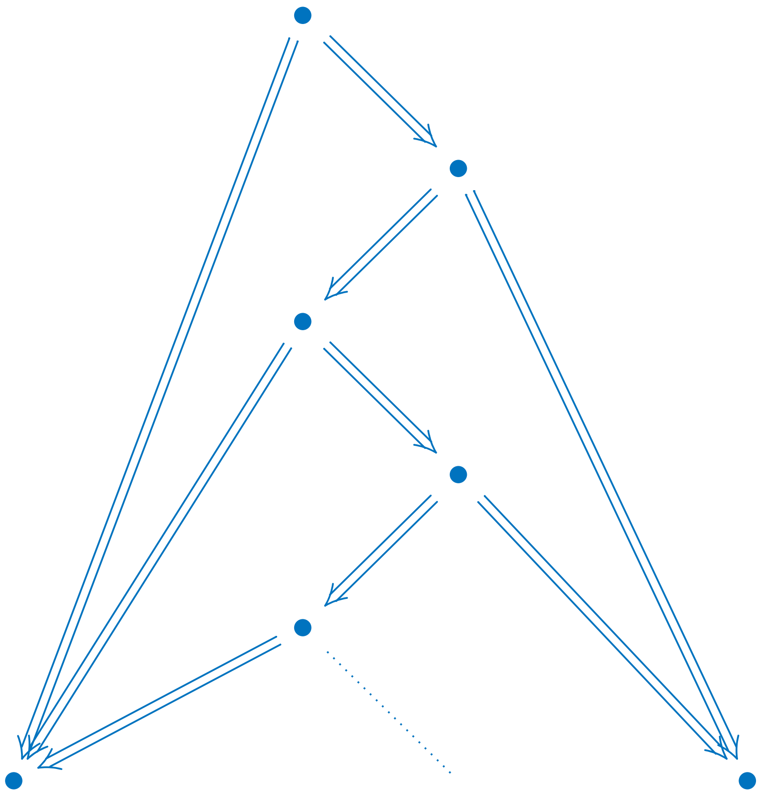
there exists $\gamma_3 \in \Gamma$ *such that*

$$\gamma_1 \Rightarrow^* \gamma_3 \qquad \gamma_2 \Rightarrow^* \gamma_3.$$

Confluence is sometimes called *Church-Rosser property*.

Confluence vs. weak confluence

Fact: *Weak confluence does not imply confluence:*



Fact: **(Newman's Lemma)** *If $\Rightarrow$ is strongly normalizing (i.e., it has no infinite paths) and weakly confluent then it is confluent.*