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

Semantic equivalence

Statements $S_1, S_2 \in \mathbf{Stmt}$ are *naturally equivalent* (equivalent w.r.t. the natural semantics)

$$S_1 \equiv_{\mathcal{NS}} S_2$$

if for all states $s, s' \in \mathbf{State}$,

$$\vdash \langle S_1, s \rangle \rightsquigarrow s' \quad \text{iff} \quad \vdash \langle S_2, s \rangle \rightsquigarrow s'$$

Fact: *For instance, the following can be proved:*

- $S; \mathbf{skip} \equiv_{\mathcal{NS}} \mathbf{skip}; S \equiv_{\mathcal{NS}} S$
- $(S_1; S_2); S_3 \equiv_{\mathcal{NS}} S_1; (S_2; S_3)$
- $\mathbf{while}\ b\ \mathbf{do}\ S \equiv_{\mathcal{NS}} \mathbf{if}\ b\ \mathbf{then}\ (S; \mathbf{while}\ b\ \mathbf{do}\ S)\ \mathbf{else\ skip}$
- $\mathbf{if}\ b\ \mathbf{then}\ (\mathbf{if}\ b'\ \mathbf{then}\ S_1\ \mathbf{else}\ S_1')\ \mathbf{else}\ S_2$
  $\equiv_{\mathcal{NS}} \mathbf{if}\ b \wedge b'\ \mathbf{then}\ S_1\ \mathbf{else}\ (\mathbf{if}\ b \wedge \neg b'\ \mathbf{then}\ S_1'\ \mathbf{else}\ S_2)$

Operational vs. natural semantics for TINY

"They are essentially the same"

Fact: *The two semantics are equivalent w.r.t. the final results described:*

$$\vdash \langle S, s\rangle \rightsquigarrow s' \quad \text{iff} \quad \langle S, s\rangle \Rightarrow^* s'$$

for all statements $S \in \mathbf{Stmt}$ *and states* $s, s' \in \mathbf{State}$.

Proof:

"$\Longrightarrow$": By induction on the structure of the derivation for $\langle S, s\rangle \rightsquigarrow s'$.

"$\Longleftarrow$": By induction on the length of the computation $\langle S, s\rangle \Rightarrow^* s'$.

"Denotational" semantics of statements

$$\mathcal{S}_{\mathcal{OS}} : \mathbf{Stmt} \to (\mathbf{State} \rightharpoonup \mathbf{State})$$

extracted from the natural or operational semantics as follows:

$$\mathcal{S}_{\mathcal{OS}}[\![S]\!]\, s = s' \;\text{ iff }\; \langle S, s\rangle \rightsquigarrow s' \quad (\text{ iff } \langle S, s\rangle \Rightarrow^* s')$$

BTW: TINY is deterministic, so this indeed defines a function

$$\mathcal{S}_{\mathcal{OS}}[\![S]\!] : \mathbf{State} \rightharpoonup \mathbf{State}$$

However, this function in general is *partial*.

So, in fact we define:

$$\mathcal{S}_{\mathcal{OS}}[\![S]\!]\, s = \begin{cases} s' & \text{if } \langle S, s\rangle \rightsquigarrow s', \text{ i.e. } \langle S, s\rangle \Rightarrow^* s' \\ \text{undefined} & \text{if } \langle S, s\rangle \not\rightsquigarrow \end{cases}$$

Operational vs. natural semantics

"They are quite different"

Natural semantics is more abstract than operational semantics

There are naturally equivalent statements with quite different sets of computations given by the operational semantics.

- Natural semantics disregards all computations that "block" or "loop".
- Natural semantics does not provide detailed view of computations.

Operational equivalence, naively

Statements $S_1, S_2 \in \mathbf{Stmt}$ are *operationally equivalent* (equivalent w.r.t. the operational semantics)

$$S_1 \equiv_{\mathcal{OS}} S_2$$

if for all states $s \in \mathbf{State}$, $\langle S_1, s\rangle \approx \langle S_2, s\rangle$, where:
configurations $\gamma_1, \gamma_2 \in \Gamma$ are equivalent, $\gamma_1 \approx \gamma_2$, if:

- $\gamma_1 = s'$ iff $\gamma_2 = s'$
- if $\gamma_1 \Rightarrow \gamma_1'$ then $\gamma_2 \Rightarrow^* \gamma_2'$ with $\gamma_1' \approx \gamma_2'$
- if $\gamma_2 \Rightarrow \gamma_2'$ then $\gamma_1 \Rightarrow^* \gamma_1'$ with $\gamma_1' \approx \gamma_2'$

THIS IS WRONG:

a circular definition!

Bisimulation

Consider any directed graph $\langle \Gamma, \Rightarrow \rangle$ with some basic observation $\mathrm{Obs}(\gamma)$ associated to every $\gamma \in \Gamma$.

A binary relation $R \subseteq \Gamma \times \Gamma$ is a strong (weak) bisimulation iff, for every $\gamma_1, \gamma_2 \in \Gamma$, if $\gamma_1 \, R \, \gamma_2$ then:

- $\mathrm{Obs}(\gamma_1) = \mathrm{Obs}(\gamma_2)$,
- for every $\gamma_1 \Rightarrow \gamma_1'$ exists $\gamma_2 \Rightarrow \gamma_2'$ ($\gamma_2 \Rightarrow^* \gamma_2'$) such that $\gamma_1' \, R \, \gamma_2'$,
- for every $\gamma_2 \Rightarrow \gamma_2'$ exists $\gamma_1 \Rightarrow \gamma_1'$ ($\gamma_1 \Rightarrow^* \gamma_1'$) such that $\gamma_1' \, R \, \gamma_2'$.

Then $\gamma_1, \gamma_2 \in \Gamma$ are strongly (weakly) bisimilar iff there exists a strong (weak) bisimulation R such that $\gamma_1 \, R, \gamma_2$.

Fact: *Strong (weak) bisimilarity is an equivalence relation and it is the largest strong (weak) bisimulation.*

Operational equivalence, correctly

Statements $S_1, S_2 \in \mathbf{Stmt}$ are *operationally equivalent* (equivalent w.r.t. the operational semantics)

$$S_1 \equiv_{\mathcal{OS}} S_2$$

if for all $s \in \mathbf{State}$, configurations $\langle S_1, s\rangle, \langle S_2, s\rangle$ are weakly bisimilar, where:

- $\mathrm{Obs}(s') = s'$,
- $\mathrm{Obs}(\langle S', s'\rangle) = \bot$

for some $\bot \notin \mathbf{State}$.

Exercise: *Prove* $\mathbf{while\ true\ do\ skip} \equiv_{\mathcal{OS}} \mathbf{while\ true\ do}\ x := x + 1$

Fact: *For the language* TINY, $S_1 \equiv_{\mathcal{OS}} S_2$ *iff* $S_1 \equiv_{\mathcal{NS}} S_2$

Adding nondeterminism and blocking

Extend the (syntax for) statements $\boxed{S \in \mathbf{Stmt}}$ as follows:

$$S ::= \cdots \mid \mathbf{abort} \mid S_1 \ \mathbf{or} \ S_2$$

Operational semantics

$$\langle S_1 \ \mathbf{or} \ S_2, s\rangle \Rightarrow \langle S_1, s\rangle \qquad \langle S_1 \ \mathbf{or} \ S_2, s\rangle \Rightarrow \langle S_2, s\rangle$$

Natural semantics

$$\frac{\langle S_1, s\rangle \rightsquigarrow s'}{\langle S_1 \ \mathbf{or} \ S_2, s\rangle \rightsquigarrow s'} \qquad \frac{\langle S_2, s\rangle \rightsquigarrow s'}{\langle S_1 \ \mathbf{or} \ S_2, s\rangle \rightsquigarrow s'}$$

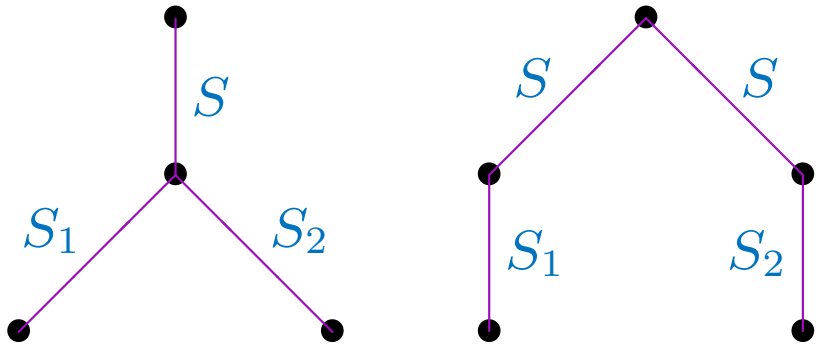
BTW: In either case, **abort** blocks (aborts?)...

Looking at equivalences

- S_1 **or** $S_2 \equiv_{\mathcal{OS}} S_2$ **or** S_1
- **abort** $\equiv_{\mathcal{NS}}$ **while true do skip**
- **abort** $\equiv_{\mathcal{OS}}$ **while true do skip**

  (??? this does not hold under strong bisimulation)
- S **or abort** $\equiv_{\mathcal{NS}} S$ (*angelic nondeterminism*)
- S **or abort** $\not\equiv_{\mathcal{OS}} S$ (unless $S \equiv_{\mathcal{OS}}$ **abort**)
- In general, the point of choice matters for operational equivalence:

  $S; (S_1$ **or** $S_2) \not\equiv_{\mathcal{OS}} (S; S_1)$ **or** $(S; S_2)$
- $S; (S_1$ **or** $S_2) \equiv_{\mathcal{NS}} (S; S_1)$ **or** $(S; S_2)$



Adding "parallelism"

Extend the statements $S \in \mathbf{Stmt}$ with a "parallel" (interleaving) construct:

$$S ::= \cdots \mid S_1 \mid\mid S_2$$

Operational semantics

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

Acceptable

Natural semantics

$$???\ \frac{\langle S_1, s\rangle \rightsquigarrow s' \quad \langle S_2, s'\rangle \rightsquigarrow s''}{\langle S_1 \mid\mid S_2, s\rangle \rightsquigarrow s''} \qquad \frac{\langle S_1, s'\rangle \rightsquigarrow s'' \quad \langle S_2, s\rangle \rightsquigarrow s'}{\langle S_1 \mid\mid S_2, s\rangle \rightsquigarrow s''}\ ???$$

Makes no sense

Denotational semantics

The method

- define syntax (*syntactic domains*)
- define *semantic domains*
- define *semantic functions*
- use *compositional* definitions

Syntactic domains

Each syntactic category of the language forms a *syntactic domain*, which has as elements all the syntactic phrases in this category.

Semantic domains

Semantic domains capture the forms of the intended meanings (*denotations)* for syntactic phrases of the language. All the denotations live in semantic domains, but typically not all elements in semantic domains are denotable.

Semantic domains are defined from *basic domains* ($\mathbf{Int}, \mathbf{Bool}$) using *domain constructors*: product, (disjoint) sum, function spaces, etc.

There is a semantic domain for each key syntactic category of the language.

Semantic functions

For each syntactic category $\mathbf{Cat}$, define a *semantic function*

$$\mathcal{C} : \mathbf{Cat} \to \mathbf{CAT}$$

which assigns to the syntactic phrases $ph \in \mathbf{Cat}$ their *denotations* in the corresponding semantic domain $\mathbf{CAT}$:

$$\mathcal{C}[\![ph]\!] \in \mathbf{CAT}$$

BTW: This defines a semantic equivalence: phrases $ph_1, ph_2 \in \mathbf{Cat}$ are *semantically equivalent* (equivalent w.r.t. the denotational semantics)

$$ph_1 \equiv_{\mathcal{DS}} ph_2$$

whenever $\mathcal{C}[\![ph_1]\!] = \mathcal{C}[\![ph_2]\!]$.

Compositionality

Semantic functions are defined *compositionally*, so that the denotation of a phrase depends only on the denotations of its immediate components:

$$\mathcal{C}[\![\varphi(ph_1, \ldots, ph_n)]\!] = \Phi(\mathcal{C}[\![ph_1]\!], \ldots, \mathcal{C}[\![ph_n]\!])$$

Such a *semantic clause* is given for each syntactic construct.

Homomorphism property lurking out

Key consequences:

STRUCTURAL INDUCTION

Congruence properties of the semantic equivalence

Denotational semantics for TINY

Syntactic domains

Num (**Var**) **Exp** **BExp** **Stmt**

Somewhat informally:

$$N \in \mathbf{Num} ::= 0 \mid 1 \mid 2 \mid \cdots$$

$$(x \in \mathbf{Var} ::= \cdots)$$

$$e \in \mathbf{Exp} ::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2$$

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

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

Denotational semantics for TINY

Semantic domains

$$\mathbf{Int} \qquad (\mathbf{Bool}) \qquad (\mathbf{State}) \qquad \mathbf{EXP} \qquad \mathbf{BEXP} \qquad \mathbf{STMT}$$

$$\begin{aligned}
\mathbf{Int} &= \{0, 1, -1, 2, -2, \ldots\} \\
\mathbf{Bool} &= \{\mathbf{tt}, \mathbf{ff}\} \\
\mathbf{State} &= \mathbf{Var} \to \mathbf{Int} \\
\mathbf{EXP} &= \mathbf{State} \to \mathbf{Int} \\
\mathbf{BEXP} &= \mathbf{State} \to \mathbf{Bool} \\
\mathbf{STMT} &= \mathbf{State} \rightharpoonup \mathbf{State}
\end{aligned}$$

Semantic functions:

$$\begin{aligned}
\mathcal{N} &: \mathbf{Num} \to \mathbf{Int} \\
\mathcal{E} &: \mathbf{Exp} \to \mathbf{EXP} \\
\mathcal{B} &: \mathbf{BExp} \to \mathbf{BEXP} \\
\mathcal{S} &: \mathbf{Stmt} \to \mathbf{STMT}
\end{aligned}$$

Basic functions on semantic domains

To define semantic functions, we are free to use various well-known functions on semantic domains:

$$
\begin{aligned}
+, -, *&: \mathbf{Int} \times \mathbf{Int} \to \mathbf{Int} \\
=, \leq&: \mathbf{Int} \times \mathbf{Int} \to \mathbf{Bool} \\
\neg&: \mathbf{Bool} \to \mathbf{Bool} \\
\wedge, \vee&: \mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool} \\
&\ldots
\end{aligned}
$$

Further constructions

Some auxiliary notation:

- *λ-notation*: $\lambda x{:}D.E$ stands for the function that maps any $d \in D$ to $E[d/x]$
- *identity*: $id_D = \lambda x{:}D.x$
- *function composition*: the composition of $f\colon D_1 \to D_2$ and $g\colon D_2 \to D_3$ is written as $f;g\colon D_1 \to D_3$
- *conditional*: $ifte_D\colon \mathbf{Bool} \times D \times D \to D$ is defined by

$$ifte_D(c, d_1, d_2) = \begin{cases} d_1 & \text{if } c = \mathbf{tt} \\ d_2 & \text{if } c = \mathbf{ff} \end{cases}$$

(the index D will often be omitted)

- *indexing*: given any function $f\colon D_1 \times \cdots \times D_n \to D$, for any domain I,

$$lift^I(f)\colon (I \to D_1) \times \cdots \times (I \to D_n) \to (I \to D)$$

is defined as follows:

$$lift^I(f)(fd_1, \ldots, fd_n) = \lambda i{:}I.f(fd_1(i), \ldots, fd_n(i))$$

For instance, the conditional on state-dependent functions, like

$$cond\colon \mathbf{BEXP} \times \mathbf{EXP} \times \mathbf{EXP} \to \mathbf{EXP}$$

given explicitly by

$$cond(B, E_1, E_2)(s) = ifte_{\mathbf{Int}}(B(s), E_1(s), E_2(s)) = \begin{cases} E_1(s) & \text{if } B(s) = \mathbf{tt} \\ E_2(s) & \text{if } B(s) = \mathbf{ff} \end{cases}$$

may be defined as $cond = lift^{\mathbf{State}}(ifte_{\mathbf{Int}})$.

All these carry over
to partial functions as well

Denotational semantics for TINY

Semantic clauses

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

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

$\mathcal{E}: \mathbf{Exp} \to \mathbf{EXP}$

$$\mathcal{E}[\![N]\!]s = \mathcal{N}[\![N]\!] \qquad \mathcal{E}[\![x]\!]s = s\,x$$
$$\mathcal{E}[\![e_1 + e_2]\!]s = \mathcal{E}[\![e_1]\!]s + \mathcal{E}[\![e_2]\!]s$$
$$\mathcal{E}[\![e_1 * e_2]\!]s = \mathcal{E}[\![e_1]\!]s * \mathcal{E}[\![e_2]\!]s$$
$$\mathcal{E}[\![e_1 - e_2]\!]s = \mathcal{E}[\![e_1]\!]s - \mathcal{E}[\![e_2]\!]s$$

$\mathcal{B}: \mathbf{BExp} \to \mathbf{BEXP}$

$$\mathcal{B}[\![\mathbf{true}]\!]s = \mathbf{tt} \qquad \mathcal{B}[\![\mathbf{false}]\!]s = \mathbf{ff} \qquad \mathcal{B}[\![\neg b]\!]s = \neg(\mathcal{B}[\![b]\!]s)$$
$$\mathcal{B}[\![e_1 \leq e_2]\!]s = (\mathcal{E}[\![e_1]\!]s \leq \mathcal{E}[\![e_2]\!]s) \qquad \mathcal{B}[\![b_1 \wedge b_2]\!]s = (\mathcal{B}[\![b_1]\!]s \wedge \mathcal{B}[\![b_2]\!]s)$$

Denotational semantics for TINY

The same clauses, using some notational sugar

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

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

$\mathcal{E} : \mathbf{Exp} \to \mathbf{EXP}$

$$\mathcal{E}[\![N]\!] = \lambda s{:}\mathbf{State}.\mathcal{N}[\![N]\!] \quad \mathcal{E}[\![x]\!] = \lambda s{:}\mathbf{State}.s\ x$$
$$\mathcal{E}[\![e_1 + e_2]\!] = lift^{\mathbf{State}}(+)(\mathcal{E}[\![e_1]\!], \mathcal{E}[\![e_2]\!])$$
$$\mathcal{E}[\![e_1 * e_2]\!] = lift^{\mathbf{State}}(*)(\mathcal{E}[\![e_1]\!], \mathcal{E}[\![e_2]\!])$$
$$\mathcal{E}[\![e_1 - e_2]\!] = lift^{\mathbf{State}}(-)(\mathcal{E}[\![e_1]\!], \mathcal{E}[\![e_2]\!])$$

$\mathcal{B} : \mathbf{BExp} \to \mathbf{BEXP}$

$$\mathcal{B}[\![\mathbf{true}]\!] = \lambda s{:}\mathbf{State}.\mathbf{tt} \quad \mathcal{B}[\![\mathbf{false}]\!] = \lambda s{:}\mathbf{State}.\mathbf{ff} \quad \mathcal{B}[\![\neg b]\!] = lift^{\mathbf{State}}(\neg)(\mathcal{B}[\![b]\!])$$
$$\mathcal{B}[\![e_1 \leq e_2]\!] = lift^{\mathbf{State}}(\leq)(\mathcal{E}[\![e_1]\!], \mathcal{E}[\![e_2]\!]) \qquad \mathcal{B}[\![b_1 \wedge b_2]\!] = lift^{\mathbf{State}}(\wedge)(\mathcal{B}[\![b_1]\!], \mathcal{B}[\![b_2]\!])$$

Denotational semantics for TINY

Semantic clauses

$\mathcal{S}: \mathbf{Stmt} \to \mathbf{STMT}$

$$
\begin{aligned}
\mathcal{S}[\![x := e]\!]s &= s[x \mapsto \mathcal{E}[\![e]\!]\ s] \\
\mathcal{S}[\![\mathbf{skip}]\!]s &= s \\
\mathcal{S}[\![S_1; S_2]\!]s &= \mathcal{S}[\![S_2]\!](\mathcal{S}[\![S_1]\!]s) \\
\mathcal{S}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]s &= ifte_{\mathbf{State}}(\mathcal{B}[\![b]\!]s, \mathcal{S}[\![S_1]\!]s, \mathcal{S}[\![S_2]\!]s) \\
\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]s &= ifte_{\mathbf{State}}(\mathcal{B}[\![b]\!]s, \mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]s', s) \\
&\qquad \text{where } s' = \mathcal{S}[\![S]\!]s
\end{aligned}
$$

Denotational semantics for TINY

The same clauses with notational sugar

$\mathcal{S}: \mathbf{Stmt} \to \mathbf{STMT}$

$$
\begin{aligned}
\mathcal{S}[\![x := e]\!] &= \lambda s{:}\mathbf{State}.s[x \mapsto \mathcal{E}[\![e]\!]\ s] \\
\mathcal{S}[\![\mathbf{skip}]\!] &= id_{\mathbf{State}} \\
\mathcal{S}[\![S_1; S_2]\!] &= \mathcal{S}[\![S_1]\!]; \mathcal{S}[\![S_2]\!] \\
\mathcal{S}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!] &= cond(\mathcal{B}[\![b]\!], \mathcal{S}[\![S_1]\!], \mathcal{S}[\![S_2]\!]) \\
\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!] &= cond(\mathcal{B}[\![b]\!], \mathcal{S}[\![S]\!]; \mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!], id_{\mathbf{State}})
\end{aligned}
$$