Working example

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

TINY

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

Syntactic categories

- *numerals*

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

with syntax given by:

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

- *variables*

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

with syntax given by:

$$x ::= \cdots \textit{ sequences of letters and digits beginning with a letter } \cdots$$

- *(arithmetic) expressions*

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

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*

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

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 each expression is of exactly 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

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 **Var** to **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

  $$s\,x$$

  $s(x)$ often written as $s\,x$

  – 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

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

defined in the obvious way:

$$\mathcal{B}[\![\mathbf{true}]\!]\, s = \mathbf{tt}$$

$$\mathcal{B}[\![\mathbf{false}]\!]\, s = \mathbf{ff}$$

$$\mathcal{B}[\![e_1 \leq e_2]\!]\, s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{E}[\![e_1]\!]\, s \leq \mathcal{E}[\![e_2]\!]\, s \\ \mathbf{ff} & \text{if } \mathcal{E}[\![e_1]\!]\, s \not\leq \mathcal{E}[\![e_2]\!]\, s \end{cases}$$

$$\mathcal{B}[\![\neg b]\!]\, s = \begin{cases} \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]\, s = \mathbf{tt} \\ \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]\, s = \mathbf{ff} \end{cases}$$

$$\mathcal{B}[\![b_1 \wedge b_2]\!]\, s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]\, s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]\, s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]\, s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]\, s = \mathbf{ff} \end{cases}$$

Semantics of statements

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

Consider the previous definitions as auxiliary

Operational semantics

small-step semantics

1960's abstract machines: Landin, McCarthy, VDL
1981 SOS: Plotkin

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 $\gamma \in \mathrm{T}$, assume $\gamma \not\Rightarrow$
- 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:

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

Computations

Maximal 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:

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

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

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

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

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

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

. . . plus transitions of sequential composition

Sequential composition

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

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

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

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

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

$$\langle (\mathbf{while}\ b\ \mathbf{do}\ S); S', s\rangle \Rightarrow \langle S', s\rangle \qquad \text{if } \mathcal{B}[\![b]\!]\, s = \mathbf{ff}$$

$$\langle (S_1; S_2); S, s\rangle \Rightarrow \langle S_1; (S_2; S), s\rangle$$

Sequential composition differently

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

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

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

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

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

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

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

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

To be read as: $\Rightarrow \subseteq \Gamma \times \Gamma$ *is the least relation such that*

- $\langle x := e, s\rangle \Rightarrow s[x \mapsto (\mathcal{E}[\![e]\!]\, s)]$, for all $x \in \mathbf{Var}$, $e \in \mathbf{Exp}$, $s \in \mathbf{State}$
- ...
- $\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle$ if $\langle S_1, s\rangle \Rightarrow s'$, for all $S_1, S_2 \in \mathbf{Stmt}$, $s, s' \in \mathbf{State}$
- ...

Rules to derive transitions

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

$$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle} \qquad \frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle}$$

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

$$\frac{\mathcal{B}[\![b]\!]\, s = \mathbf{tt}}{\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow \langle S; \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle} \qquad \frac{\mathcal{B}[\![b]\!]\, s = \mathbf{ff}}{\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow s}$$

Notational variants:

- axioms vs. rules without premises: $\langle \mathbf{skip}, s\rangle \Rightarrow s$
- side-conditions vs. premises: $\dfrac{}{\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \Rightarrow s}$ if $\mathcal{B}[\![b]\!]\, s = \mathbf{ff}$

Proof-theoretic reading

We give

- axioms, like $\langle x := e, s\rangle \Rightarrow s[x \mapsto (\mathcal{E}[\![e]\!]\, s)]$, and
- rules, like $\dfrac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$

to *derive* (or better: *prove*) *judgements* of the form $\gamma \Rightarrow \gamma'$, i.e.

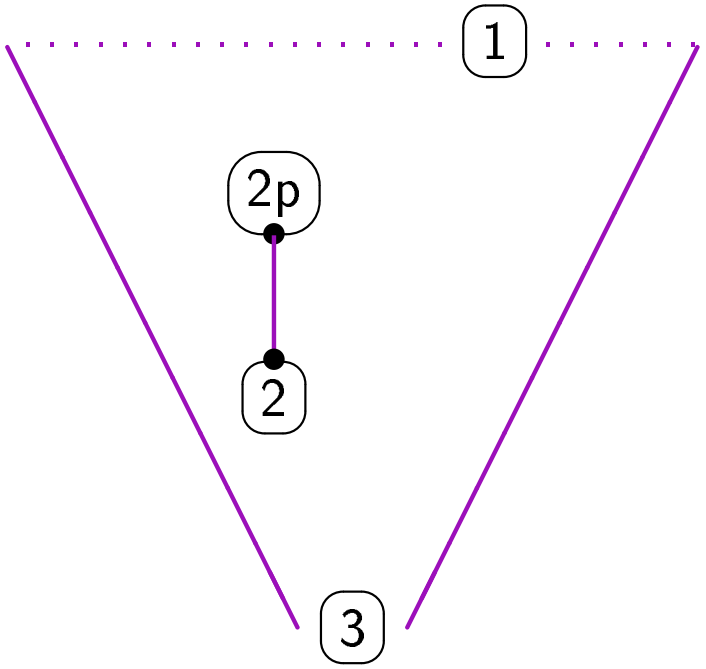
$$\boxed{\langle S, s\rangle \Rightarrow s'} \quad \text{or} \quad \boxed{\langle S, s\rangle \Rightarrow \langle S', s'\rangle}$$

Actually: we give axiom and rule *schemata*, which are *generic* in the choice of elements to be substituted for meta-variables used ($x \in \mathbf{Var}$, $e \in \mathbf{Exp}$, $s, s' \in \mathbf{State}$, $S_1, S_2 \in \mathbf{Stmt}$, etc).

We may write $\boxed{\vdash \gamma \Rightarrow \gamma'}$ to indicate that there exists a proof of $\gamma \Rightarrow \gamma'$.

Proofs/derivations

Finite *proof tree* (or *derivation tree*):



- leaves: labelled by axioms, e.g.
  $\boxed{1}: \ \langle x := e, s\rangle \Rightarrow s[x \mapsto (\mathcal{E}[\![e]\!]\, s)]$
- other nodes: labelled according to the rules, e.g.
  $$\frac{\boxed{2p}: \ \langle S_1, s\rangle \Rightarrow s'}{\boxed{2}: \ \langle S_1; S_2, s\rangle \Rightarrow s''}$$
- root: judgement proved, e.g. $\boxed{3}: \ \langle S, s\rangle \Rightarrow s'$

Another proof technique

Induction on the structure of proof/derivation trees

Some properties

Fact: 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* 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

On nondeterminism of computations

Nondeterministic small-step semantics for arithmetic expressions: $\Gamma = \mathbf{Exp} \times \mathbf{State}$

$$\frac{}{\langle x, s\rangle \Rightarrow \langle N, s\rangle} \text{ if } \mathcal{N}[\![N]\!] = s\, x$$

$$\frac{\langle e_1, s\rangle \Rightarrow \langle e_1', s'\rangle}{\langle e_1 + e_2, s\rangle \Rightarrow \langle e_1' + e_2, s'\rangle} \qquad \frac{\langle e_2, s\rangle \Rightarrow \langle e_2', s'\rangle}{\langle e_1 + e_2, s\rangle \Rightarrow \langle e_1 + e_2', s'\rangle}$$

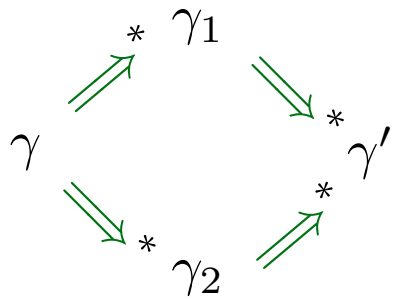
"numerals" for negative integers needed

$$\frac{}{\langle N_1 + N_2, s\rangle \Rightarrow \langle M, s\rangle} \text{ if } \mathcal{N}[\![N_1]\!] + \mathcal{N}[\![N_2]\!] = \mathcal{N}[\![M]\!]$$

BUT: $\cdots$ and similarly for $e_1 * e_2$ and $e_1 - e_2 \cdots$

Fact: *if* $\langle e, s\rangle \Rightarrow^* \langle N, s'\rangle$ *and* $\langle e, s\rangle \Rightarrow^* \langle N', s''\rangle$ *then* $\mathcal{N}[\![N]\!] = \mathcal{N}[\![N']\!]$ *(and* $s = s' = s''$*).*

Include "semantic" integers as expressions and modify the semantics above

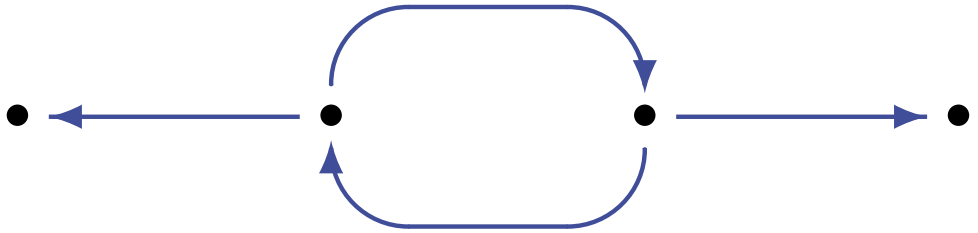


Church-Rosser property

Confluence: if $\gamma \Rightarrow^* \gamma_1$ and $\gamma \Rightarrow^* \gamma_2$ then $\gamma_1 \Rightarrow^* \gamma'$ and $\gamma_2 \Rightarrow^* \gamma'$ for some γ'

Weak confluence: if $\gamma \Rightarrow \gamma_1$ and $\gamma \Rightarrow \gamma_2$ then $\gamma_1 \Rightarrow^* \gamma'$ and $\gamma_2 \Rightarrow^* \gamma'$ for some γ'

Warning: weak confluence does not entail confluence:



Fact: *If $\Rightarrow \subseteq \Gamma \times \Gamma$ is strongly normalizing (i.e., no infinite computations) and is weakly confluent then it is confluent.*

Newman's Lemma

Some variants

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

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

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

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

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

- similarly for **while**, the first case
- instead of the current rules for **while**:

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

- . . .
- in fact: two distinct variants of $\Rightarrow$ are given at the previous slides

Natural semantics

big-step operational semantics

mid 1980's: Gilles Kahn

Overall idea:

- define *configurations*: $\gamma \in \Gamma$
- indicate which of them are *terminal*: $\mathrm{T} \subseteq \Gamma$
- instead of computations, consider (define) *transitions* directly *to final configurations* that are reached by computations: $\rightsquigarrow \subseteq \Gamma \times \mathrm{T}$

Informally:

– if $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$, $\gamma_n \in \mathrm{T}$, then $\gamma_0 \rightsquigarrow \gamma_n$

– if $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n$, $\gamma_n \notin \mathrm{T}$ and $\gamma_n \not\Rightarrow$, then $\gamma_0 \not\rightsquigarrow$

– if $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots$ then $\gamma_0 \not\rightsquigarrow$

TINY: natural semantics

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

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

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

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

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

$$\frac{\langle S, s\rangle \rightsquigarrow s' \qquad \langle \mathbf{while}\ b\ \mathbf{do}\ S, s'\rangle \rightsquigarrow s''}{\langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \rightsquigarrow s''} \text{ if } \mathcal{B}[\![b]\!]\, s = \mathbf{tt}$$

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

Configurations:

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

Terminal configurations: as before

$\mathrm{T} = \mathbf{State}$

Transitions: as given here

Some properties

Fact: TINY *is deterministic, i.e.:*

$$\textit{for each } \vdash \langle S, s\rangle \rightsquigarrow s', \textit{ if } \vdash \langle S, s\rangle \rightsquigarrow s'' \textit{ then } s' = s''.$$

Proof: By (easy) induction on the proof of $\vdash \langle S, s\rangle \rightsquigarrow s'$.

BTW: Try also to prove this by induction on the structure of S — *is there a trouble?*

- structural induction fails here: the semantics of **while** is *not compositional*.

$$\frac{\langle S, s\rangle \rightsquigarrow s' \qquad \langle \textbf{while } b \textbf{ do } S, s'\rangle \rightsquigarrow s''}{\langle \textbf{while } b \textbf{ do } S, s\rangle \rightsquigarrow s''} \text{ if } \mathcal{B}[\![b]\!]\, s = \textbf{tt}$$

More on compositionality later

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 by induction of the derivation:*

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

Congruence properties

Fact: *The semantic equivalence is preserved by the linguistic constructs:*

- *if* $S_1 \equiv_{\mathcal{NS}} S_1'$ *and* $S_2 \equiv_{\mathcal{NS}} S_2'$ *then*

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

- *if* $S_1 \equiv_{\mathcal{NS}} S_1'$ *and* $S_2 \equiv_{\mathcal{NS}} S_2'$ *then*

$$\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \equiv_{\mathcal{NS}} \textbf{if } b \textbf{ then } S_1' \textbf{ else } S_2'$$

- *if* $S \equiv_{\mathcal{NS}} S'$ *then*

$$\textbf{while } b \textbf{ do } S \equiv_{\mathcal{NS}} \textbf{while } b \textbf{ do } S'$$

BTW: This can be extended to incorporate a similarly defined equivalence for expressions and boolean expressions.

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

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

extracted from the natural or operational semantics as follows:

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

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

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

However, this function in general is *partial*.

So, in fact we define:

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

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

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 $s \in \mathbf{State}$, $\langle S_1, s\rangle \sim \langle S_2, s\rangle$ for some relation $\sim \subseteq \Gamma \times \Gamma$ such that $s_1 \sim s_2$ iff $s_1 = s_2$, and for all $\gamma_1, \gamma_2 \in \mathbf{Stmt} \times \mathbf{State}$ such that $\gamma_1 \sim \gamma_2$

- if $\gamma_1 \Rightarrow \gamma_1'$ then $\gamma_2 \Rightarrow^* \gamma_2'$ for some γ_2' with $\gamma_1' \sim \gamma_2'$
- if $\gamma_2 \Rightarrow \gamma_2'$ then $\gamma_1 \Rightarrow^* \gamma_1'$ for some γ_1' with $\gamma_1' \sim \gamma_2'$

WEAK BISIMULATION

Fact: *If* $S_1 \equiv_{\mathcal{OS}} S_2$ *then* $S_1 \equiv_{\mathcal{NS}} S_2$

Equivalences given as examples for natural equivalence carry over here as well. In fact, for the language considered so far, natural and operational equivalence coincide.

Bisimulation in general

History: Park'81, Milner

Consider a graph $\langle K, \rightarrow \rangle$ with "local observations" $\mathcal{O}(\kappa)$, for each $\kappa \in K$.

Definition: *$\rho \subseteq K \times K$ is a (strong) bisimulation on $\langle K, \rightarrow \rangle$ w.r.t. $\mathcal{O}(\_)$ if for all $\kappa_1, \kappa_2 \in K$ such that $\kappa_1 \; \rho \; \kappa_2$ we have $\mathcal{O}(\kappa_1) = \mathcal{O}(\kappa_2)$, and*

- *if $\kappa_1 \rightarrow \kappa_1'$ then $\kappa_2 \rightarrow \kappa_2'$ for some κ_2' with $\kappa_1' \; \rho \; \kappa_2'$*
- *if $\kappa_2 \rightarrow \kappa_2'$ then $\kappa_1 \rightarrow \kappa_1'$ for some κ_1' with $\kappa_1' \; \rho \; \kappa_2'$*

Two nodes $\kappa_1, \kappa_2 \in K$ are *(strongly) bisimilar*, written $\kappa_1 \approx \kappa_2$, if $\kappa_1 \; \rho \; \kappa_2$ for some bisimulation $\rho \subseteq K \times K$.

Fact: *$\approx \subseteq K \times K$ is an equivalence and bisimulation.*

Weak bisimilarity, as used for $\equiv_{\mathcal{OS}}$, is defined analogously

Fact: *Every bisimulation is a weak bisimulation, but not vice versa in general.*

Adding nondeterminism and blocking

Extend the (syntax for) statements $S \in \mathbf{Stmt}$ 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

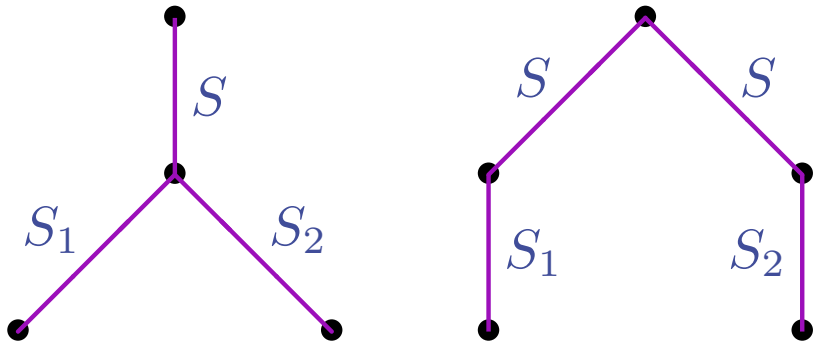
$$\frac{\langle S_1, s\rangle \rightsquigarrow s'}{\langle S_1 \ \mathbf{or} \ S_2, s\rangle \rightsquigarrow s'} \qquad \frac{\langle S_2, s\rangle \rightsquigarrow s'}{\langle S_1 \ \mathbf{or} \ S_2, s\rangle \rightsquigarrow s'}$$

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**

  BTW: 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:

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

Operational semantics

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

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

Acceptable

Natural semantics

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

Makes no sense