Small-step Operational Semantics

CS 6371: Advanced Programming Languages

Kevin W. Hamlen

February 6, 2024

Large-step Operational Semantics

Large-step Judgments

A large-step judgment declares that a configuration **converges to** a store or value:

command judgments	$\langle c, \sigma \rangle \Downarrow \sigma'$	$(\sigma' \in \Sigma)$
arithmetic judgments	$\langle a, \sigma \rangle \Downarrow n$	$(n \in \mathbb{Z})$
boolean judgments	$\langle b, \sigma \rangle \Downarrow p$	$(p \in \{T, F\})$

where "converges to" ($\Downarrow$) means "terminates and returns a value of ..."

- Advantages:
  - relatively simple to reason about (few inference rules)
  - good when code correctness means returning the correct result
- Disadvantages:
  - mostly cannot prove things about non-terminating programs
  - insufficient when code correctness depends on what the program does as it executes (e.g., side-effects)

Small-step Operational Semantics

Alternative: Small-step Operational Semantics

Small-step Judgments

A small-step judgment declares that a configuration **steps to** a new configuration:

command judgments	$\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$
arithmetic judgments	$\langle a, \sigma \rangle \rightarrow_1 \langle a', \sigma' \rangle$
boolean judgments	$\langle b, \sigma \rangle \rightarrow_1 \langle b', \sigma' \rangle$

where "steps to" ($\rightarrow_1$) means "keeps executing in this next configuration."

- Advantages:
  - can prove things about non-terminating code
  - can prove things about all machine states realized by a computation
- Disadvantage:
  - more complex (more rules)
  - harder to reason about terminating programs (more induction)

Small-step Rule for `skip`

$$\frac{}{\langle \texttt{skip}, \sigma\rangle \rightarrow_1 \langle ?, ?\rangle}$$

Small-step Rule for `skip`

$$\frac{}{\langle \mathtt{skip}, \sigma \rangle \rightarrow_1 \langle \mathtt{skip}, \sigma \rangle}$$

This is incorrect. Why? What does this rule actually say?

Small-step Rule for `skip`

Need a way to say that `skip` has no "next configuration." It's done.

Solution: No rule for `skip`!

Sometimes write: $\langle \mathtt{skip}, \sigma \rangle \not\to_1$

Definition (final configuration): $\langle \mathtt{skip}, \sigma \rangle$, $\langle n, \sigma \rangle$, $\langle \mathtt{true}, \sigma \rangle$ and $\langle \mathtt{false}, \sigma \rangle$ are final configurations for all $\sigma \in \Sigma$.

Small-step Rule for Sequence

$$\frac{?}{\langle c_1\,;c_2, \sigma\rangle \rightarrow_1 \langle ?, ?\rangle}$$

Small-step Rule for Sequence

$$\frac{\langle c_1, \sigma \rangle \rightarrow_1 \langle c_1', \sigma' \rangle}{\langle c_1\,; c_2, \sigma \rangle \rightarrow_1 \langle ?, ? \rangle}$$

Small-step Rule for Sequence

$$\frac{\langle c_1, \sigma \rangle \rightarrow_1 \langle c_1', \sigma' \rangle}{\langle c_1 \,;\, c_2, \sigma \rangle \rightarrow_1 \langle c_1' \,;\, c_2, \sigma' \rangle}$$

Small-step Rule for Sequence

$$\frac{\langle c_1, \sigma\rangle \rightarrow_1 \langle c_1', \sigma'\rangle}{\langle c_1 ; c_2, \sigma\rangle \rightarrow_1 \langle c_1' ; c_2, \sigma'\rangle}\text{(S1)}$$

But how do we ever execute c_2?

Need some way to say, "If c_1 can't take any more steps, then work on c_2."

"can't take any more steps" $=$ "final configuration"

Small-step Rules for Sequence

Solution: Two rules

$$\frac{\langle c_1, \sigma\rangle \rightarrow_1 \langle c_1', \sigma'\rangle}{\langle c_1 ; c_2, \sigma\rangle \rightarrow_1 \langle c_1' ; c_2, \sigma'\rangle}\text{(S1)}$$

$$\frac{}{\langle \texttt{skip}; c_2, \sigma\rangle \rightarrow_1 \langle c_2, \sigma\rangle}\text{(S2)}$$

Small-step Rule for Assignment

$$\frac{}{\langle v := a, \sigma \rangle \rightarrow_1 \langle ?, ? \rangle}$$

Small-step Rule for Assignment

$$\frac{}{\langle v \,\texttt{:=}\, a, \sigma\rangle \to_1 \langle \texttt{skip}, \sigma[v \mapsto a]\rangle}$$

Small-step Rule for Assignment

$$\overline{\langle v \mathtt{:=} a, \sigma\rangle \rightarrow_1 \langle \mathtt{skip}, \sigma[v \mapsto a]\rangle}$$

This is type-incorrect because a is not necessarily an integer.

Small-step Rule for Assignment

$$\frac{\langle a, \sigma \rangle \rightarrow_1 \langle n, \sigma \rangle}{\langle v \; \texttt{:=} \; a, \sigma \rangle \rightarrow_1 \langle \texttt{skip}, \sigma[v \mapsto n] \rangle}$$

Still wrong: What if a takes many steps to finally yield an answer n?

Don't confuse large-step and small-step semantics!

Small-step Rules for Assignment

Solution: Again, two rules

$$\frac{\langle a, \sigma \rangle \rightarrow_1 \langle a', \sigma' \rangle}{\langle v \,\texttt{:=}\, a, \sigma \rangle \rightarrow_1 \langle v \,\texttt{:=}\, a', \sigma' \rangle}\text{(S3)}$$

$$\frac{}{\langle v \,\texttt{:=}\, n, \sigma \rangle \rightarrow_1 \langle \texttt{skip}, \sigma[v \mapsto n] \rangle}\text{(S4)}$$

Small-step Rules for Conditionals

For conditionals we need three rules:

$$\frac{\langle b, \sigma \rangle \rightarrow_1 \langle b', \sigma' \rangle}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \rightarrow_1 \langle \texttt{if } b' \texttt{ then } c_1 \texttt{ else } c_2, \sigma' \rangle}\text{(S5)}$$

$$\frac{}{\langle \texttt{if true then } c_1 \texttt{ else } c_2, \sigma \rangle \rightarrow_1 \langle c_1, \sigma \rangle}\text{(S6)}$$

$$\frac{}{\langle \texttt{if false then } c_1 \texttt{ else } c_2, \sigma \rangle \rightarrow_1 \langle c_2, \sigma \rangle}\text{(S7)}$$

Small-step Rule for While-loop

For while-loop we'll use the same trick from large-step semantics:

$$\frac{}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \rightarrow_1 \langle \texttt{if}\ b\ \texttt{then}\ (c\texttt{;while}\ b\ \texttt{do}\ c)\ \texttt{else}\ \texttt{skip}, \sigma\rangle}\ (\text{S8})$$

Other Small-step Rules

- This completes the small-step rules for commands.
- Also need rules for arithmetic and boolean judgments.
  - Nothing particularly surprising, but requires lots of rules (26 total)
  - See online lecture notes for full list.
  - Exercise: See if you can figure them out on your own, then check the notes.

Totality

Definition (total relation): A relation $\mathcal{R}$ is **total** if every domain element x is related to a range element y (i.e., $\forall x, \exists y, x \,\mathcal{R}\, y$).

Question: Is $\Downarrow$ a total relation?

In other words, is there any program c and store σ for which there is **no** σ' satisfying $\langle c, \sigma \rangle \Downarrow \sigma'$?

Totality

Definition (total relation): A relation $\mathcal{R}$ is **total** if every domain element x is related to a range element y (i.e., $\forall x, \exists y, x \,\mathcal{R}\, y$).

Question: Is $\Downarrow$ a total relation?

In other words, is there any program c and store σ for which there is **no** σ' satisfying $\langle c, \sigma \rangle \Downarrow \sigma'$?

Answer: $\Downarrow$ is not total. For example, if we choose $c = \texttt{while true do skip}$ (and any σ), there is no σ' satisfying $\langle c, \sigma \rangle \Downarrow \sigma'$.

Follow-up question: What about aside from infinite loops?

Totality

Definition (total relation): A relation $\mathcal{R}$ is **total** if every domain element x is related to a range element y (i.e., $\forall x, \exists y, x \, \mathcal{R} \, y$).

Question: Is $\Downarrow$ a total relation?

Answer: $\Downarrow$ is not total. For example, if we choose $c = \texttt{while true do skip}$ (and any σ), there is no σ' satisfying $\langle c, \sigma \rangle \Downarrow \sigma'$.

Follow-up question: What about aside from infinite loops?

Answer: We could also choose $c = (\texttt{x := y})$ and any σ such that $y \notin \sigma^{\leftarrow}$.

Two cases of non-totality:

1. infinite loops (limitation of large-step semantics)
2. uninitialized reads (intentionally implementation-defined)

Ambiguity

Definition (ambiguity): A derivation system is said to be **ambiguous** if there exists a judgment having multiple distinct derivations.

Question: Are our large-step semantics ambiguous?

In other words, is there some judgment $\langle c, \sigma \rangle \Downarrow \sigma'$ that is derivable in two different ways?

Ambiguity

Definition (ambiguity): A derivation system is said to be **ambiguous** if there exists a judgment having multiple distinct derivations.

Question: Are our large-step semantics ambiguous?

In other words, is there some judgment $\langle c, \sigma \rangle \Downarrow \sigma'$ that is derivable in two different ways?

Answer: No. For every judgment that's derivable, there's only one way to derive it.

Ambiguity

Derivation systems for real languages usually have ambiguity (and that's okay because it gives implementors choices).

Example: Adding these rules makes our system ambiguous.

$$\frac{\langle b_1, \sigma\rangle \Downarrow F}{\langle b_1 \;\texttt{\&\&}\; b_2, \sigma\rangle \Downarrow F}$$

$$\frac{\langle b_2, \sigma\rangle \Downarrow F}{\langle b_1 \;\texttt{\&\&}\; b_2, \sigma\rangle \Downarrow F}$$

Determinism

Definition (deterministic): A relation $\mathcal{R}$ is deterministic (also called a function) if every domain element x is related to at most one range element y (i.e., $\forall x, \forall y_1, y_2, (x \, \mathcal{R} \, y_1) \wedge (x \, \mathcal{R} \, y_2) \Rightarrow y_1 = y_2$).

Question: Is $\Downarrow$ deterministic?

Determinism

Definition (deterministic): A relation $\mathcal{R}$ is deterministic (also called a function) if every domain element x is related to at most one range element y (i.e., $\forall x, \forall y_1, y_2, (x \, \mathcal{R} \, y_1) \wedge (x \, \mathcal{R} \, y_2) \Rightarrow y_1 = y_2$).

Question: Is $\Downarrow$ deterministic?

Answer: Yes.

Determinism

Our system would become non-deterministic if we added something like this:

$$a ::= \cdots \mid \texttt{rand}$$

$$\frac{n \in \mathbb{Z}}{\langle \texttt{rand}, \sigma \rangle \Downarrow n}$$