Operational Semantics

CS 6371: Advanced Programming Languages

Kevin W. Hamlen

January 30, 2024

Formally Specifing Language Syntax

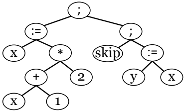
- Let's create a new language: Simple IMPerative Language (SIMPL)
- Backus-Naur Form (BNF)
  - Invented by John Backus and Peter Naur (inventors of ALGOL-60 and later FORTRAN)
  - Notation for expressing context-free grammars (CFGs)
- Convention: Teletype font for symbols vs. mathematical font for mathematical operators
  - `+`, `-`, and `*` are symbols from your keyboard that have no particular mathematical meaning
  - $+$, $-$, and $*$ are the mathematical operators for addition, subtraction, and multiplication
  - (To make things easier for our OCaml implementation, we will define them to be 31-bit integer addition, subtraction, and multiplication operators, which is how OCaml performs those operations natively.)

Syntax of SIMPL

commands	$c ::= \texttt{skip} \mid c_1\,\texttt{;}\,c_2 \mid v\,\texttt{:=}\,a \mid \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid \texttt{while}\ b\ \texttt{do}\ c$
boolean expressions	$b ::= \texttt{true} \mid \texttt{false} \mid a_1\,\texttt{<=}\,a_2 \mid b_1\,\texttt{\&\&}\,b_2 \mid b_1\,\texttt{\|\|}\,b_2 \mid \texttt{!}b$
arithmetic expressions	$a ::= n \mid v \mid a_1\,\texttt{+}\,a_2 \mid a_1\,\texttt{-}\,a_2 \mid a_1\,\texttt{*}\,a_2$
variable names	v
integer constants	n

Syntax vs. Semantics

- Syntactic definition imparts **no meaning** (semantics) of programs
  - Symbol "+" might not mean addition.
  - (Can you think of a language where it does not?)
- Elements of CFGs (e.g., defined by BNF) are *abstract syntax trees* (ASTs)
  - Use parentheses to describe AST's structure
  - Example: $\mathtt{x} := (\mathtt{x} + 1) * 2; (\mathtt{skip}; \mathtt{y} := \mathtt{x})$



- Parser transforms symbol stream into AST
  - Uses various precedence and associativity rules to auto-insert parenths
  - I'll assume you know how that works (use a parser generator or take compilers/automata theory class).
  - When I write a program, it denotes an already-parsed AST.

To Infinity (and Beyond)

- CFG elements (e.g., programs) are *finite* but *unbounded*.
  - **Finite:** The number of nodes in the AST equals a natural number (not infinity).
  - **Unbounded:** For every program c there exists a larger program c'.
- The *set of all programs* is countably infinite.
  - Countably infinite = set has cardinality equal to the set of all natural numbers
- But each individual program is finite.
  - An infinite-sized program is actually a syntax error, because the CFG has no infinite-sized members.

Operational Semantics

- Operational semantics - mathematically define the meanings of programs in terms of the operation of an abstract machine

Stores

A **store** (machine state) in SIMPL is a partial function from variable names to integers:

$$\Sigma = v \rightharpoonup \mathbb{Z}$$

$$\sigma \in \Sigma$$

- (Partial) functions can be written as sets of input-output pairs:
  - Example: $\sigma = \{(\mathtt{x}, 8), (\mathtt{y}, -10), (\mathtt{z}, 0)\}$
  - Not every set of input-output pairs is a function though, so be careful.
  - Non-function: $\{(\mathtt{x}, 8), (\mathtt{x}, 10)\}$

Configurations and Judgments

Configurations

A **configuration** is a command or expression paired with a store:

command configurations	$\langle c, \sigma \rangle$
arithmetic configurations	$\langle a, \sigma \rangle$
boolean configurations	$\langle b, \sigma \rangle$

Judgments

A **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	$\langle b, \sigma \rangle \Downarrow p$	$(p \in \{T, F\})$

"Converges to" ($\Downarrow$) informally means "terminates and returns a value of ..."

Derivations

- We now have formalisms for talking about program behaviors (judgments), but we haven't defined which judgments are "true".
- Insight: Judgments are like mathematical propositions, but for a new math (computation).
- How do we define "truth" in propositional logic? (Laws or Inference Rules)

Example: Law of Modus Ponens $$\frac{p \quad p \Rightarrow q}{q}\,(\text{MP})$$

  - Each rule written as a "fraction" with zero or more hypotheses on top, and a conclusion on the bottom
  - Free variables in rules are universally quantified
  - Rules can be nested to form tree-shaped **derivations** (proofs) of truth:

$$\frac{\dfrac{p \quad p \Rightarrow q}{q}\,(\text{MP}) \qquad q \Rightarrow r}{r}\,(\text{MP})$$

- Solution: We need logical axioms that define computations in SIMPL!

Rule #1: Skip

Inference Rule (Axiom) for `skip`

$$\frac{}{\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma}\,(1)$$

- no hypotheses = axiom
- True for every σ (i.e., σ is universally quantified)

Warning: Misnamed Variables

The following rule is *completely different*!

A different (wrong) rule for `skip`

$$\frac{}{\langle \mathtt{skip}, \sigma \rangle \Downarrow \sigma'}\,(1)$$

- What's the difference?
- What does this rule effectively say `skip` does?

Rule #2: Sequence

Inference Rule for ;

$$\frac{?}{\langle c_1 \,; c_2, \sigma\rangle \Downarrow \sigma'}\,(2)$$

Rule #2: Sequence

Inference Rule for ;

$$\frac{\langle c_1, \sigma \rangle \Downarrow \sigma_2}{\langle c_1 ; c_2, \sigma \rangle \Downarrow \sigma'} (2)$$

Rule #2: Sequence

Inference Rule for ;

$$\frac{\langle c_1, \sigma\rangle \Downarrow \sigma_2 \quad \langle c_2, \sigma_2\rangle \Downarrow \sigma'}{\langle c_1 ; c_2, \sigma\rangle \Downarrow \sigma'}(2)$$

Example Derivation

$$\frac{}{\langle \texttt{skip;(skip;skip)}, \sigma \rangle \Downarrow ?}$$

Example Derivation

$$\dfrac{\dfrac{}{\langle \texttt{skip}, \sigma\rangle \Downarrow ??} \qquad \dfrac{}{\langle \texttt{skip;skip}, ??\rangle \Downarrow ?}}{\langle \texttt{skip;(skip;skip)}, \sigma\rangle \Downarrow ?}\text{(2)}$$

Example Derivation

$$\dfrac{\dfrac{}{\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma}(1) \qquad \dfrac{}{\langle \texttt{skip;skip}, \sigma \rangle \Downarrow ?}}{\langle \texttt{skip;(skip;skip)}, \sigma \rangle \Downarrow ?}(2)$$

Example Derivation

$$
\dfrac{\dfrac{}{\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma}(1) \qquad \dfrac{\langle \texttt{skip}, \sigma \rangle \Downarrow ?? \quad \langle \texttt{skip}, ?? \rangle \Downarrow ?}{\langle \texttt{skip;skip}, \sigma \rangle \Downarrow ?}(2)}{\langle \texttt{skip;(skip;skip)}, \sigma \rangle \Downarrow ?}(2)
$$

Example Derivation

$$
\dfrac{
  \dfrac{}{\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma}\,(1)
  \qquad
  \dfrac{
    \dfrac{}{\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma}\,(1)
    \qquad
    \langle \texttt{skip}, \sigma \rangle \Downarrow ?
  }{
    \langle \texttt{skip;skip}, \sigma \rangle \Downarrow ?
  }\,(2)
}{
  \langle \texttt{skip;}(\texttt{skip;skip}), \sigma \rangle \Downarrow ?
}\,(2)
$$

Example Derivation

$$\dfrac{\dfrac{}{\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma}(1) \qquad \dfrac{\dfrac{}{\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma}(1) \qquad \dfrac{}{\langle \texttt{skip}, \sigma \rangle \Downarrow \sigma}(1)}{\langle \texttt{skip};\texttt{skip}, \sigma \rangle \Downarrow \sigma}(2)}{\langle \texttt{skip};(\texttt{skip};\texttt{skip}), \sigma \rangle \Downarrow \sigma}(2)$$

Building Derivations

$$\dfrac{\dfrac{}{\langle \mathtt{skip}, \sigma\rangle \Downarrow \sigma}(1) \qquad \dfrac{\dfrac{}{\langle \mathtt{skip}, \sigma\rangle \Downarrow \sigma}(1) \quad \dfrac{}{\langle \mathtt{skip}, \sigma\rangle \Downarrow \sigma}(1)}{\langle \mathtt{skip};\mathtt{skip}, \sigma\rangle \Downarrow \sigma}(2)}{\langle \mathtt{skip};(\mathtt{skip};\mathtt{skip}), \sigma\rangle \Downarrow \sigma}(2)$$

- Work bottom-up, left-to-right (usually).
- Identify (by number) which rule you're using to the right of the bar.
- Instantiate rule variables consistently and uniformly at each rule use.
  - If $\sigma = \sigma_1$ in this rule instance, then every σ appearing in the rule must be replaced with σ_1.
  - Treat rule literally, not what you expect/want it to say!
- Derivations and infinity
  - No infinite-sized derivations! (Each derivation must have strictly finite height.)
  - The set of all derivations is countably infinite.

Rule #3: Assignment

Inference Rule for :=

$$\frac{}{\langle v := a, \sigma \rangle \Downarrow ?}\,(3)$$

Warning: Type-inconsistent Rules

First attempt at assignment rule:

Malformed (wrong) Rule for :=

$$\frac{}{\langle v := a, \sigma\rangle \Downarrow \sigma[v \mapsto a]}\,(3)$$

Notation (functional update):

$$f[x \mapsto y] = (f - \{(x, z) \mid (x, z) \in f\}) \cup \{(x, y)\}$$

But the rule above is not a mathematically sensible definition. Why?

Rule #3: Assignment

Correct formulation of assignment rule:

Inference Rule for :=

$$\frac{\langle a, \sigma \rangle \Downarrow n}{\langle v := a, \sigma \rangle \Downarrow \sigma[v \mapsto n]} \text{(3)}$$

Notation (functional update):

$$f[x \mapsto y] = (f - \{(x, z) \mid (x, z) \in f\}) \cup \{(x, y)\}$$

Rule #4: Conditional

Inference Rule for if-then-else

$$\frac{}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma\rangle \Downarrow ?}\,(4)$$

Rules #4–5: Conditional

Solution: Multiple rules per syntactic form are perfectly valid and often useful.

Inference Rules for if-then-else

$$\frac{\langle b, \sigma \rangle \Downarrow T \qquad \langle c_1, \sigma \rangle \Downarrow \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \Downarrow \sigma'} \text{(4)}$$

$$\frac{\langle b, \sigma \rangle \Downarrow F \qquad \langle c_2, \sigma \rangle \Downarrow \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \Downarrow \sigma'} \text{(5)}$$

Rule #6: While-loop

Inference Rule for while-loop

$$\frac{}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \Downarrow ?}$$

This is a tough one.

Rule #6: While-loop

The false part is easy, but what about the true part?

Inference Rules for while-loop

$$\frac{\langle b, \sigma \rangle \Downarrow F}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma \rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma \rangle \Downarrow T \qquad ?}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma \rangle \Downarrow ?}$$

Rule #6: While-loop

The false part is easy, but what about the true part?

Inference Rules for while-loop

$$\frac{\langle b, \sigma \rangle \Downarrow F}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma \rangle \Downarrow T \qquad \langle c, \sigma \rangle \Downarrow \sigma_2}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \Downarrow ?}$$

Rule #6: While-loop

Idea: What about using the entire while-loop recursively?

Inference Rules for while-loop

$$\frac{\langle b, \sigma \rangle \Downarrow F}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma \rangle \Downarrow T \qquad \langle c, \sigma \rangle \Downarrow \sigma_2 \qquad \langle \texttt{while } b \texttt{ do } c, \sigma_2 \rangle \Downarrow \sigma'}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \Downarrow \sigma'}$$

Danger: Is this rule circular?

Warning: Circular Rules

Warning: It is easy to create valid yet pointless rules using recursion.

Example of a valid yet pointless inference rule

$$\frac{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma'}$$

- This inference rule is valid and sound.
- But it isn't useful. (Recall that derivations are finite.)

Rule #6: While-loop

Is this rule pointless?

Inference Rules for while-loop

$$\frac{\langle b, \sigma \rangle \Downarrow F}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma \rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma \rangle \Downarrow T \qquad \langle c, \sigma \rangle \Downarrow \sigma_2 \qquad \langle \texttt{while}\ b\ \texttt{do}\ c, \sigma_2 \rangle \Downarrow \sigma'}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma \rangle \Downarrow \sigma'}$$

Rule #6: While-loop

Is this rule pointless? No, this works! But let's compact it into a single rule...

Inference Rules for while-loop

$$\frac{\langle b, \sigma\rangle \Downarrow F}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma\rangle \Downarrow T \qquad \langle c, \sigma\rangle \Downarrow \sigma_2 \qquad \langle \texttt{while } b \texttt{ do } c, \sigma_2\rangle \Downarrow \sigma'}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \Downarrow \sigma'}$$

Rule #6: While-loop

Inference Rule for while-loop

$$\frac{\langle \texttt{if } b \texttt{ then } (c;\texttt{while } b \texttt{ do } c) \texttt{ else skip}, \sigma\rangle \Downarrow \sigma'}{\langle \texttt{while } b \texttt{ do } c, \sigma\rangle \Downarrow \sigma'} (6)$$

Rule #6: While-loop

This single rule suffices because with it we can derive:

$$\dfrac{\dfrac{\langle b, \sigma\rangle \Downarrow F \qquad \dfrac{}{\langle \texttt{skip}, \sigma\rangle \Downarrow \sigma}(1)}{\langle \texttt{if}\ b\ \texttt{then}\ (c\texttt{;while}\ b\ \texttt{do}\ c)\ \texttt{else skip}, \sigma\rangle \Downarrow \sigma}(4)}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma'}(6)$$

and

$$\dfrac{\dfrac{\langle b, \sigma\rangle \Downarrow T \qquad \dfrac{\langle c, \sigma\rangle \Downarrow \sigma_2 \qquad \langle \texttt{while}\ b\ \texttt{do}\ c, \sigma_2\rangle \Downarrow \sigma'}{\langle c\texttt{;while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma'}(2)}{\langle \texttt{if}\ b\ \texttt{then}\ (c\texttt{;while}\ b\ \texttt{do}\ c)\ \texttt{else skip}, \sigma\rangle \Downarrow \sigma'}(5)}{\langle \texttt{while}\ b\ \texttt{do}\ c, \sigma\rangle \Downarrow \sigma'}(6)$$

Other Rules

- Also need inference rules for arithmetic and boolean judgments
- See reference section of Assignment #2 for full list
- Mostly "obvious" but I'll mention a few

Symbols vs. Mathematical Operators

Inference Rule for addition

$$\frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1 \mathbf{+} a_2, \sigma \rangle \Downarrow n_1 + n_2} \text{(15)}$$

Recall: "**+**" is a symbol from the input stream (no mathematical significance), whereas "$+$" is the mathematical operator for 31-bit modular integer addition.

Reading the Store

To get variable values, we simply use σ as a function.

Inference Rule for variable-read

$$\frac{}{\langle v, \sigma \rangle \Downarrow \sigma(v)} \quad (14)$$

- Rules with no premises are called **axioms**.
- When writing axioms, feel free to omit the fraction line.

Comprehending Inference Rules

Inference Rule for ;

$$\frac{\langle c_1, \sigma \rangle \Downarrow \sigma_2 \quad \langle c_2, \sigma_2 \rangle \Downarrow \sigma'}{\langle c_1 ; c_2, \sigma \rangle \Downarrow \sigma'} (2)$$

- Two ways to understand each inference rule:
  1. Implementation recipe: *"To compute $c_1 ; c_2$ on σ, first (recursively) compute c_1 on σ to get σ_2, then (recursively) compute c_2 on σ_2 to get σ'."*
  2. Logical specification: *"To prove that $c_1 ; c_2$ on σ converges to σ', it suffices to prove c_1 on σ converges to some σ_2, and c_2 on σ_2 converges to σ'."*
- Big hint: Reading each rule as an implementation recipe essentially solves Assignment #2 for you. Your solution should be a nearly verbatim translation from the rules to code.
- Spanning the semantic gap
  - Rules are definitions, not theorems. So if you get them "wrong", there's no proof of wrongness. You've merely defined a really strange language.
  - Functional languages minimize the chance for error when mapping the math to code.