CS 611: Lecture 5

Small-step semantics and Induction
September 6, 1999

Cornell University Computer Science Department
Andrew Myers

Administration

- No class this Friday, September 10
- Problem set #1 change: problem 3 is Winskel problem 3.5

Large-step "continue" semantics

- Last time: this relation doesn't work:

$$\langle c, l, \sigma \rangle \rightarrow \sigma'$$

(with l a stack of while commands to restart)

- Problem: semicolon rule always runs second command, so continue; c_2 doesn't work:

$$\frac{\langle c, l, \sigma \rangle \rightarrow \sigma'}{\langle \text{continue}, c :: l, \sigma \rangle \rightarrow \sigma'} \qquad \frac{\langle c_1, l, \sigma \rangle \rightarrow \sigma'' \quad \langle c_2, l, \sigma'' \rangle \rightarrow \sigma'}{\langle c_1; c_2, l, \sigma \rangle \rightarrow \sigma'}$$ Oops!

Solution

- Need to change command evaluation relation:

$$\langle c, l, \sigma \rangle \rightarrow \sigma', \text{run+halt}$$

"current enclosing while loop has finished"

$$\frac{\langle c, l, \sigma \rangle \rightarrow \sigma', \text{run}}{\langle \text{continue}, c :: l, \sigma \rangle \rightarrow \sigma', \text{halt}} \qquad \frac{\langle c_1, l, \sigma \rangle \rightarrow \sigma', \text{halt}}{\langle c_1; c_2, l, \sigma \rangle \rightarrow \sigma', \text{halt}}$$

$$\frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, (\text{while } b \text{ do } c :: l), \sigma \rangle \rightarrow \sigma', \text{halt}}{\langle \text{while } b \text{ do } c, l, \sigma \rangle \rightarrow \sigma', \text{run}} \qquad \frac{\langle c_1, l, \sigma \rangle \rightarrow \sigma'', \text{run} \quad \langle c_2, l, \sigma'' \rangle \rightarrow \sigma', \text{x}}{\langle c_1; c_2, l, \sigma \rangle \rightarrow \sigma', \text{x}}$$

$$\frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, (\text{while } b \text{ do } c :: l), \sigma \rangle \rightarrow \sigma'', \text{run} \quad \langle \text{while } b \text{ do } c, l, \sigma'' \rangle \rightarrow \sigma', \text{run}}{\langle \text{while } b \text{ do } c, l, \sigma \rangle \rightarrow \sigma', \text{run}}$$

+ propagate run, halt to if rules...

Problems with large-step semantics

- Large-step semantics don't model what an interpreter does that well
- Fails to pin down evaluation order:

$$\frac{\langle a_1, \sigma \rangle \rightarrow n_1 \quad \langle a_2, \sigma \rangle \rightarrow n_2}{\langle a_1 + a_2, \sigma \rangle \rightarrow n_1 + n_2}$$

- Have to know which rule to use; can't express (implementable) non-deterministic choice:

$$\frac{\langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle c_1 \,[]\, c_2, \sigma \rangle \rightarrow \sigma'} \qquad \frac{\langle c_2, \sigma \rangle \rightarrow \sigma'}{\langle c_1 \,[]\, c_2, \sigma \rangle \rightarrow \sigma'}$$

Small-step semantics

- Idea: represent computation by transitions *between* configurations rather than evaluation to a result
- Arithmetic:

$$\langle a_1, \sigma \rangle \rightarrow \langle a_2, \sigma \rangle \rightarrow \ldots \rightarrow \langle n, \sigma \rangle$$

- Example:

$$\langle (2+3) \times 4, \sigma \rangle \rightarrow \langle 5 \times 4, \sigma \rangle \rightarrow \langle 20, \sigma \rangle$$

Other expressions

- Booleans:

$$\langle b_1, \sigma\rangle \rightarrow \langle b_2, \sigma\rangle \rightarrow \ldots \rightarrow \langle \text{true}, \sigma\rangle$$

- Commands:

$$\langle c_1, \sigma\rangle \rightarrow \langle c_2, \sigma\rangle \rightarrow \ldots \rightarrow \langle \text{skip}, \sigma\rangle$$

- Final configuration represents a completed computation

Small-step rules

- Inference rules for small-step semantics of arithmetic:

$$\frac{\langle a_1, \sigma\rangle \rightarrow \langle a'_1, \sigma\rangle}{\langle a_1 + a_2, \sigma\rangle \rightarrow \langle a'_1 + a_2, \sigma\rangle}$$

"First argument may be evaluated before second"

$$\frac{\langle a_2, \sigma\rangle \rightarrow \langle a'_2, \sigma\rangle}{\langle n + a_2, \sigma\rangle \rightarrow \langle n + a'_2, \sigma\rangle}$$

"Second argument is evaluated (only) if first argument is fully evaluated"

$$\frac{n = n_1 + n_2}{\langle n_1 + n_2, \sigma\rangle \rightarrow \langle n, \sigma\rangle}$$

"Result is computed once both arguments are fully evaluated"

Small-step command rules

Assignment:

$$\frac{\langle a, \sigma\rangle \rightarrow \langle a', \sigma\rangle}{\langle X := a, \sigma\rangle \rightarrow \langle X := a', \sigma\rangle}$$

$$\frac{}{\langle X := n, \sigma\rangle \rightarrow \langle \text{skip}, \sigma[X \mapsto n]\rangle}$$

$$\frac{\langle b, \sigma\rangle \rightarrow \langle b', \sigma\rangle}{\langle \text{if } b \text{ then } c \text{ else } c', \sigma\rangle \rightarrow \langle \text{if } b' \text{ then } c \text{ else } c', \sigma\rangle}$$

if:

$$\frac{}{\langle \text{if true then } c \text{ else } c', \sigma\rangle \rightarrow \langle c, \sigma\rangle}$$

$$\frac{}{\langle \text{if false then } c \text{ else } c', \sigma\rangle \rightarrow \langle c', \sigma\rangle}$$

Implementable non-determinism

$$\frac{}{\langle c \,[]\, c', \sigma\rangle \rightarrow \langle c, \sigma\rangle}$$

$$\frac{}{\langle c \,[]\, c', \sigma\rangle \rightarrow \langle c', \sigma\rangle}$$

... a non-deterministic choice operator that does not "back out" when a computation gets stuck.

Small-step vs. large-step

- How do we know that the small-step semantics and large-step semantics describe the same language?
- Define relation $\rightarrow^+$ on configurations representing many single steps...

$$\langle a, \sigma\rangle \rightarrow^+ \langle a', \sigma\rangle$$

iff

$$\langle a, \sigma\rangle \rightarrow \langle a_1, \sigma\rangle \rightarrow \ldots \rightarrow \langle a_n, \sigma\rangle \rightarrow \langle a', \sigma\rangle$$

- Arithmetic is equivalent if

$$\langle a, \sigma\rangle \rightarrow n \;\Leftrightarrow\; \langle a, \sigma\rangle \rightarrow^+ \langle n, \sigma\rangle$$

- How can we prove this?

Need for new proof techniques

- Proofs not possible with tools shown so far:
  - equivalence of different semantics
    - small-step vs. large-step
    - while vs. while+continue
  - termination of expressions
  - deterministic evaluation of programs
- Need to be able to unwind a structural derivation an unbounded number of steps!
- Need various forms of *induction*

Proving termination

- Argument for termination of arithmetic:
  - Expressions of the form X or n always terminate in one step (evaluation defined by axioms)
  - Expressions of the form $a_1 + a_2$, $a_1 \times a_2$, $a_1 - a_2$ terminate if their constituent arithmetic expressions terminate
- Circular!

Induction

- Simple mathematical induction: a property P(x) holds for all $x \geq 1$ if

$$P(1)$$
$$\forall x \geq 1 \,.\, (P(x) \Rightarrow P(x+1))$$

- Define $P(x) = \forall x' \in \{1, \ldots, x\} \,.\, P'(x)$

$$\forall x \geq 1 \,.\, P(x) \Rightarrow \forall x \geq 1 \,.\, P'(x)$$
$$P(1) \Leftrightarrow P'(1)$$
$$\forall x \geq 1 \,.\, (P(x) \Rightarrow P(x+1)) \Leftrightarrow$$
$$\underbrace{\forall x \geq 1 \,.\, [\, \forall x' \in \{1, \ldots, x-1\} \,.\, P'(x') \Rightarrow P'(x) \,]}_{\text{course-of-values induction}}$$

Structural Induction

- Idea: every arithmetic expression is built up by applying the rules

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

- Consider the *size* of an expression to be the number of times these rules are applied to construct an expression:

$$\text{size}(n) = 1$$
$$\text{size}(X) = 1$$
$$\text{size}(a_0 + a_1) = 1 + \max(\text{size}(a_0), \text{size}(a_1))$$

- All expressions have finite size

Finishing proof by structural induction

- Use course-of-values induction on size of expression

$$\forall x \geq 1 \,.\, [\, \forall x' \in \{1, \ldots, x-1\} \,.\, P(x') \Rightarrow P(x) \,]$$

  - P(x) is the property "all expressions of size x terminate"
  - P(1) is clearly true
  - If P(x') is true for all $x' < x$, then an expression of size x and form $a_0 + a_1$, $a_0 - a_1$, $a_0 \times a_1$ must terminate: its constituent expressions a_0, a_1 are necessarily of size x-1 or smaller.

Induction on Derivations

- Sometimes proof requires induction on height of structural derivation (intuitively: length of execution)
- Lets us prove some property is never destroyed during execution
- Example: proving commands in IMP are deterministic
- Want to show:

$$\forall \sigma, \sigma', \sigma'', c \,.$$
$$(\langle c, \sigma \rangle \rightarrow \sigma' \wedge \langle c, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma'')$$

Proof of Determinism

- Every command that terminates has a derivation with finite height
- Height of derivation tree is longest chain from conclusion (root) to any axiom (leaf)
- Let P(x) be statement "all statements whose derivation has a height of x are deterministic"
- P(1): skip, $X := a$
- Now consider commands if, while with derivation tree height x

if

- Assume $\langle$if b then c_0 else $c_1, \sigma\rangle \to \sigma'$,
  $\langle$if b then c_0 else $c_1, \sigma\rangle \to \sigma''$
- Since booleans are deterministic, b evaluates the same way for both
- WLOG derivations look like

$$\frac{\dots \quad \dots \quad \dots}{\langle b, \sigma\rangle \to \text{true} \quad \langle c_0, \sigma\rangle \to \sigma' \quad \langle \text{if } b \text{ then } c_0 \text{ else } c_1, \sigma\rangle \to \sigma'} \qquad \frac{\dots \quad \dots \quad \dots}{\langle b, \sigma\rangle \to \text{true} \quad \langle c_0, \sigma\rangle \to \sigma'' \quad \langle \text{if } b \text{ then } c_0 \text{ else } c_1, \sigma\rangle \to \sigma''}$$

- Sub-derivations: $\dfrac{\dots}{\langle c_0, \sigma\rangle \to \sigma'}$ $\dfrac{\dots}{\langle c_0, \sigma\rangle \to \sigma''}$
- Therefore, $\sigma' = \sigma''$!

while

- Assume $\langle$while b do $c, \sigma\rangle \to \sigma_1$,
  $\langle$while b do $c, \sigma\rangle \to \sigma_2$
- Derivations look like

$$\frac{\dots \quad \dots \quad \dots}{\langle b, \sigma\rangle \to \text{true} \quad \langle c, \sigma\rangle \to \sigma'_1 \quad \langle \text{while } b \text{ do } c, \sigma'_1\rangle \to \sigma_1}{\langle \text{while } b \text{ do } c, \sigma\rangle \to \sigma_1}$$

$$\frac{\dots \quad \dots \quad \dots}{\langle b, \sigma\rangle \to \text{true} \quad \langle c, \sigma\rangle \to \sigma'_2 \quad \langle \text{while } b \text{ do } c, \sigma'_2\rangle \to \sigma_2}{\langle \text{while } b \text{ do } c, \sigma\rangle \to \sigma_2}$$

while

- Assume $\langle$while b do $c, \sigma\rangle \to \sigma_1$,
  $\langle$while b do $c, \sigma\rangle \to \sigma_2$
- Derivations look like

$\sigma'_1 = \sigma'_2$,
$\sigma_1 = \sigma_2$

$$\frac{\dots \quad \dots \quad \dots}{\langle b, \sigma\rangle \to \text{true} \quad \langle c, \sigma\rangle \to \sigma'_1 \quad \langle \text{while } b \text{ do } c, \sigma'_1\rangle \to \sigma_1}{\langle \text{while } b \text{ do } c, \sigma\rangle \to \sigma_1}$$

$$\frac{\dots \quad \dots \quad \dots}{\langle b, \sigma\rangle \to \text{true} \quad \langle c, \sigma\rangle \to \sigma'_2 \quad \langle \text{while } b \text{ do } c, \sigma'_2\rangle \to \sigma_2}{\langle \text{while } b \text{ do } c, \sigma\rangle \to \sigma_2}$$

Next time

- Recursive definitions of sets
- Rule induction
- Equivalence of small-step and large-step semantics