

CS421 Lecture 17: Natural Semantics¹

Mark Hills

`mhills@cs.uiuc.edu`

University of Illinois at Urbana-Champaign

July 18, 2006

¹Based on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, and Elsa Gunter

Natural Semantics

Natural Semantics: Additional Features

Objectives

By the end of this lecture, you should

- ▶ be able to read and understand language definitions given using natural semantics
- ▶ understand some of the differences between natural and transition semantics

Natural Semantics

Natural semantics are similar to transition semantics except

- ▶ transition semantics specifies a relation between individual steps of a computation
- ▶ while natural semantics specifies a relation between a state in the computation and a final result

Rules will be of the form

$$\langle C, m \rangle \Downarrow m'$$

And it should be the case that

$$\langle C, m \rangle \rightarrow^* m' \text{ iff } \langle C, m \rangle \Downarrow m'$$

An aside

Both transition semantics and natural semantics are forms of *operational semantics*. Transition semantics is also known as *structural operational semantics*, or *small-step semantics*. Natural semantics is also known as *big-step semantics*.

Natural Semantics of Atomic Expressions

In both cases we only take one step, so the semantics are roughly the same:

- ▶ Identifiers: $\langle X, \sigma \rangle \Downarrow \sigma(X)$
- ▶ Numerals: $\langle n, \sigma \rangle \Downarrow n$
- ▶ Booleans:
 - ▶ $\langle \mathbf{true}, \sigma \rangle \Downarrow \mathbf{true}$
 - ▶ $\langle \mathbf{false}, \sigma \rangle \Downarrow \mathbf{false}$

Arithmetic Expressions

Since the rules for addition, subtraction, and multiplication all look the same except for the operator used, we will just let $\text{op} = \{+, -, \times\}$ below:

$$\frac{\langle a_0, \sigma \rangle \Downarrow n_0 \quad \langle a_1, \sigma \rangle \Downarrow n_1 \quad n = n_0 \text{ op}_{int} n_1}{\langle a_0 \text{ op } a_1, \sigma \rangle \Downarrow n}$$

Relational Operations

We again use op here to stand for a number of operations which all have similar rules. Let $\text{op} = \{=, \leq\}$ below:

$$\frac{\langle a_0, \sigma \rangle \Downarrow n_0 \quad \langle a_1, \sigma \rangle \Downarrow n_1 \quad b = n_0 \text{ op}_{int} n_1}{\langle a_0 \text{ op } a_1, \sigma \rangle \Downarrow n}$$

Logical Operations: And

Since our and operation “short-circuits”, we need two rules, one that tells us the result when the first operation is **false**, and one that tells us the result when the first operation is **true**:

$$\frac{\langle b_0, \sigma \rangle \Downarrow \mathbf{false}}{\langle b_0 \wedge b_1, \sigma \rangle \Downarrow \mathbf{false}}$$

$$\frac{\langle b_0, \sigma \rangle \Downarrow \mathbf{true} \quad \langle b_1, \sigma \rangle \Downarrow b}{\langle b_0 \wedge b_1, \sigma \rangle \Downarrow b}$$

Logical Operations: Or

Or is similar to and – both “short-circuit”, so we also need two rules here. Note the main difference is that we flip around the value used to decide whether to continue with the second argument:

$$\frac{\langle b_0, \sigma \rangle \Downarrow \mathbf{true}}{\langle b_0 \vee b_1, \sigma \rangle \Downarrow \mathbf{true}}$$

$$\frac{\langle b_0, \sigma \rangle \Downarrow \mathbf{false} \quad \langle b_1, \sigma \rangle \Downarrow b}{\langle b_0 \vee b_1, \sigma \rangle \Downarrow b}$$

Logical Operations: Not

Negation is fairly simple – we can just handle it with two cases, one for evaluating the first argument to **true**, the other for **false**:

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{false}}{\langle \neg b, \sigma \rangle \Downarrow \mathbf{true}}$$

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{true}}{\langle \neg b, \sigma \rangle \Downarrow \mathbf{false}}$$

Commands: Skip

Similarly to how expressions are handled, we will evaluate commands completely, yielding the state after the command is executed. Skip is defined similarly to the definition in transition semantics:

$$\{\mathbf{skip}, \sigma\} \Downarrow \sigma$$

Commands: Assignment

With assignment, we build in the evaluation of the expression directly into the assignment rule, instead of requiring two steps.

$$\frac{\langle a, \sigma \rangle \Downarrow n}{\langle X := a, \sigma \rangle \Downarrow \sigma[n/X]}$$

Commands: Sequencing

Sequencing again requires we completely evaluate the component commands. Here, we can directly see the result of the first command being “passed on” to the second.

$$\frac{\langle c_0, \sigma \rangle \Downarrow \sigma' \quad \langle c_1, \sigma' \rangle \Downarrow \sigma''}{\langle c_0; c_1, \sigma \rangle \Downarrow \sigma''}$$

Commands: Conditional

We make the choice directly in the rules premises, since we are required to evaluate to the final state.

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{true} \quad \langle c_0, \sigma \rangle \Downarrow \sigma'}{\langle \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1, \sigma \rangle \Downarrow \sigma'}$$

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{false} \quad \langle c_1, \sigma \rangle \Downarrow \sigma'}{\langle \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1, \sigma \rangle \Downarrow \sigma'}$$

Commands: While

The rule for the while command is “recursive”, in that it refers to itself. One important point to note is that we can use while directly here, instead of needing to wrap it inside an **if** like we did with the transition semantics.

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{false}}{\langle \mathbf{while} \ b \ \mathbf{do} \ c, \sigma \rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{true} \quad \langle c, \sigma \rangle \Downarrow \sigma' \quad \langle \mathbf{while} \ b \ \mathbf{do} \ c, \sigma' \rangle \Downarrow \sigma''}{\langle \mathbf{while} \ b \ \mathbf{do} \ c, \sigma \rangle \Downarrow \sigma''}$$

Example

To see how we can “evaluate” something with natural semantics, start with:

$$\langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle$$

In this state, we have at some point assigned the value 7 to location x .

Example

$\langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?$

Example

$$\frac{\langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow?}{\langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow?}$$

Example

$\langle x, \{x \mapsto 7\} \rangle \Downarrow?$ $\langle 5, \{x \mapsto 7\} \rangle \Downarrow?$

$\langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow?$

$\langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow?$

Example

$$\langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5$$

$$\langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow ?$$

$$\langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?$$

Example

$$\langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5$$

$$\langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false}$$

$$\langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?$$

Example

$$\frac{\langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5}{\langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false} \quad \langle y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?}$$

$$\langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?$$

Example

$$\frac{\langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5}{\langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false}} \quad \frac{\langle 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?}{\langle y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?}$$
$$\langle \mathbf{if } x \leq 5 \mathbf{ then } y := 2 + 3 \mathbf{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?$$

Example

$$\frac{\langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5}{\langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false}}$$

$$\frac{\langle 3, \{x \mapsto 7\} \rangle \Downarrow? \quad \langle 4, \{x \mapsto 7\} \rangle \Downarrow?}{\langle 3 + 4, \{x \mapsto 7\} \rangle \Downarrow?}$$

$$\frac{\langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false} \quad \langle y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow?}{\langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow?}$$

Example

$$\begin{array}{c}
 \langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5 \\
 \hline
 \langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false} \\
 \hline
 \langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?
 \end{array}
 \qquad
 \begin{array}{c}
 \langle 3, \{x \mapsto 7\} \rangle \Downarrow 3 \quad \langle 4, \{x \mapsto 7\} \rangle \Downarrow 4 \\
 \hline
 \langle 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ? \\
 \hline
 \langle y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?
 \end{array}$$

Example

$$\begin{array}{c}
 \langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5 \\
 \hline
 \langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false} \\
 \hline
 \langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?
 \end{array}
 \qquad
 \begin{array}{c}
 \langle 3, \{x \mapsto 7\} \rangle \Downarrow 3 \quad \langle 4, \{x \mapsto 7\} \rangle \Downarrow 4 \\
 \hline
 \langle 3 + 4, \{x \mapsto 7\} \rangle \Downarrow 7 \\
 \hline
 \langle y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?
 \end{array}$$

Example

$$\begin{array}{c}
 \langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5 \\
 \hline
 \langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false} \\
 \hline
 \langle \mathbf{if } x \leq 5 \mathbf{ then } y := 2 + 3 \mathbf{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?
 \end{array}
 \qquad
 \begin{array}{c}
 \langle 3, \{x \mapsto 7\} \rangle \Downarrow 3 \quad \langle 4, \{x \mapsto 7\} \rangle \Downarrow 4 \\
 \hline
 \langle 3 + 4, \{x \mapsto 7\} \rangle \Downarrow 7 \\
 \hline
 \langle y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow \{x \mapsto 7, y \mapsto 7\} \\
 \hline
 \langle \mathbf{if } x \leq 5 \mathbf{ then } y := 2 + 3 \mathbf{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow ?
 \end{array}$$

Example

$$\begin{array}{c}
 \langle x, \{x \mapsto 7\} \rangle \Downarrow 7 \quad \langle 5, \{x \mapsto 7\} \rangle \Downarrow 5 \\
 \hline
 \langle x \leq 5, \{x \mapsto 7\} \rangle \Downarrow \mathbf{false} \\
 \hline
 \langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow \{x \mapsto 7, y \mapsto 7\}
 \end{array}$$

$$\begin{array}{c}
 \langle 3, \{x \mapsto 7\} \rangle \Downarrow 3 \quad \langle 4, \{x \mapsto 7\} \rangle \Downarrow 4 \\
 \hline
 \langle 3 + 4, \{x \mapsto 7\} \rangle \Downarrow 7 \\
 \hline
 \langle y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow \{x \mapsto 7, y \mapsto 7\} \\
 \hline
 \langle \text{if } x \leq 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4, \{x \mapsto 7\} \rangle \Downarrow \{x \mapsto 7, y \mapsto 7\}
 \end{array}$$

Commands: Let

The rule for let is slightly different than in our transition semantics example – here we allow the let body to be a command, not just an arithmetic expression. This means we now could have side effects in the let body.

$$\frac{\langle a, \sigma \rangle \Downarrow n \quad \langle c, \sigma[n/x] \rangle \Downarrow \sigma'}{\langle \mathbf{let} \ x = a \ \mathbf{in} \ c, \sigma \rangle \Downarrow \sigma''}$$

Note that, overall, we return σ'' , not σ' . σ'' is defined to be identical to σ' everywhere except x – for x , it is defined as identical to σ if x is defined in σ , else it is undefined. Why? Because we need to propagate all state changes, but also need to enforce scope.

Example

$$\frac{\langle 5, \{x \mapsto 17\} \rangle \Downarrow 5 \quad \frac{\langle x, \{x \mapsto 5\} \rangle \Downarrow 5 \quad \langle 3, \{x \mapsto 5\} \rangle \Downarrow 3}{\langle x + 3, \{x \mapsto 5\} \rangle \Downarrow 8}}{\langle x := x + 3, \{x \mapsto 5\} \rangle \Downarrow \{x \mapsto 8\}}}{\langle \text{let } x = 5 \text{ in } (x := x + 3), \{x \mapsto 17\} \rangle \Downarrow ?}$$

Example

$$\frac{\frac{\langle x, \{x \mapsto 5\} \rangle \Downarrow 5 \quad \langle 3, \{x \mapsto 5\} \rangle \Downarrow 3}{\langle x + 3, \{x \mapsto 5\} \rangle \Downarrow 8}}{\langle 5, \{x \mapsto 17\} \rangle \Downarrow 5 \quad \langle x := x + 3, \{x \mapsto 5\} \rangle \Downarrow \{x \mapsto 8\}}$$

$$\langle \mathbf{let} \ x = 5 \ \mathbf{in} \ (x := x + 3), \{x \mapsto 17\} \rangle \Downarrow \{x \mapsto 17\}$$

The let construct, with the need to track state, somewhat conflicts with the assignment operation, leading to semantics that are a bit clumsy.

Why Have Two Semantics?

We now have natural and transition semantics for our language.
Why have both?

- ▶ Natural semantics has a nice correspondence to a recursive interpreter for our programs
- ▶ Transition semantics corresponds nicely to an imperative interpreter, taking individual steps and modifying the state
- ▶ Natural semantics provides added conciseness...
- ▶ ...but cannot model nontermination (and cannot distinguish nontermination from some types of failures)