Operational Semantics

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Outline

- Operational Semantics
  - Topics
    - The purpose of formal systems
    - Transitions
    - Substitution
    - The Church Rosser Theorem
  - Objectives
    - Evaluate an expression according to the given transition semantic rules
    - Write some semantic rules for a common Haskell expression
    - Explain the significance of the Church Rosser property.
  - Activities
    - Transition Activity
      - Evaluate an expression
      - Write rules for and, or, and tail
    - Church Rosser Activity
      - Demonstrate that C++ does not have the Church Rosser property.

Objectives

You should be able to...

In order to express the meaning of a program, we need a formal language to capture these meanings. Today's semantics will use *transitions* to specify the value of an expression. By the end of lecture, you should know how to use transitional semantics.

- what the word "semantics" means.
- determine the value of an expression (i.e., be able to read)
- specify the meaning of a language (i.e., be able to write).

You should also know the Church-Rosser property and be able to give examples of languages that have it and languages that don't have it.

Parts of a Formal System

To create a formal system, you must specify the following:

- A set of *symbols* or an *alphabet.*
- A definition of a *valid sentence.*
- A set of *transformation rules* to make new valid sentences out of old ones.
- A set of *initial valid sentences.*

You do **NOT** need:

- An *interpretation* of those symbols.
  They are highly recommended, but the formal system can exist and do its work without one.

Example

Symbols S, (,), Z, P, x, y.

Definition of a furbitz

- Z is a furbitz. x and y are variables of type furbitz.
- if x is a furbitz, then $S(x)$ is a furbitz.
- if x and y are furbitzi, then $P(x, y)$ is a furbitz.

Definition of the gloppit relation

- Z has the gloppit relation with Z.
- If x and y have the gloppit relation, then $S(x)$ and $S(y)$ have the gloppit relation.
- If α and β, then we can write $\alpha g \beta$.

True Sentences If $\alpha g \beta$, then also

- $P(S(\alpha), \beta) g P(\alpha, S(\beta))$, and $P(Z, \alpha) g \alpha$

Example

Symbols S, (,), Z, P, x, y.

Definition of an integer

- 0 is an integer. x and y are variables of type integer.
- if x is an integer, then $S(x)$ is an integer.
- if x and y are integers, then $P(x, y)$ is an integer.

Definition of the equality relation

- 0 has the equality relation with 0.
- If x and y have the equality relation, then $S(x)$ and $S(y)$ have the equality relation.
- If α and β, then we can write $\alpha = \beta$.

True Sentences If $\alpha = \beta$, then also

- $P(S(\alpha), \beta) = P(\alpha, S(\beta))$, and $P(0, \alpha) = \alpha$

Example

Symbols S, (,), Z, P, x, y.

Definition of an integer

- 1 is an integer. x and y are variables of type integer.
- if x is an integer, then $S(x)$ is an integer.
- if x and y are integers, then $P(x, y)$ is an integer.

Definition of the equality relation

- 1 has the equality relation with 1.
- If x and y have the equality relation, then $S(x)$ and $S(y)$ have the equality relation.
- If α and β, then we can write $\alpha = \beta$.

True Sentences If $\alpha = \beta$, then also

- $P(S(\alpha), \beta) = P(\alpha, S(\beta))$, and $P(1, \alpha) = \alpha$

Transformations

- There are many ways we can specify the meaning of an expression. One way is to specify the steps that the computer will take during an evaluation.
- An *evaluation* has the following form:

  $$e_1 \longrightarrow e_2$$

  where e is some expression, and e_2 is another expression, possibly a value.
  Examples:
  - `if true then 4 else 38` $\longrightarrow$ `4`
  - `13 + 4 * 5` $\longrightarrow$ `13 + 20`
- Note well: $\longrightarrow$ indicates *exactly one* step of evaluation.

Preliminaries

- In transition semantics we need to be able to distinguish between *values* and *expressions*.
    - A *value* is a valid *expression* that can not be evaluated any further.
    - (Note, the converse is not true.)
- Use letters U, V, and W to represent values.
- Use letters M, N, and L to represent expressions.

If Statements

Here are three semantic rules for the `if` statement.

- $\texttt{if true then } M \texttt{ else } N \rightarrow M$
- $\texttt{if false then } M \texttt{ else } N \rightarrow N$
- $$\frac{L \rightarrow L'}{\texttt{if } L \texttt{ then } M \texttt{ else } N \ \rightarrow\ \texttt{if } L' \texttt{ then } M \texttt{ else } N}$$

In English:

- If the conditional part is `true`, evaluate the first branch.
- If the conditional part is `false`, evaluate the second branch.
- Otherwise, if the conditional part is not yet evaluated, evaluate it one step.

Obvious Rules

- These rules are boring. But we need to include them anyway.

$$\frac{M \rightarrow M'}{M \oplus N \rightarrow M' \oplus N} \qquad \frac{N \rightarrow N'}{V \oplus N \rightarrow V \oplus N'}$$

  Where $\oplus$ is +, -, >, <, ...

- These rules are so boring that we don't include them.

  $0 + 0 \rightarrow 0 \quad 0 + 1 \rightarrow 1 \quad \ldots$

  $1 + 0 \rightarrow 1 \quad 1 + 1 \rightarrow 2 \quad \ldots$

  et cetera...

Example Evaluation

Evaluate: `if 3 > 2 then 5 + 9 else 2 * 4`

```
     if 3 > 2 then 5 + 9 else 2 * 4
 →   if true then 5 + 9 else 2 * 4
 →   5 + 9
 →   14
```

Other Notations

Notations

$$
\begin{array}{lcl}
\rightarrow^0 & \equiv & \text{The identity} \\
\rightarrow^1 & \equiv & \rightarrow \\
\rightarrow^n & \equiv & \rightarrow \cdot \rightarrow^{n-1} \\
\rightarrow^* & \equiv & \bigcup_{i=0}^{\infty} \rightarrow^i \\
\rightarrow^+ & \equiv & \bigcup_{i=1}^{\infty} \rightarrow^i \\
a \leftarrow b & \equiv & b \rightarrow a \\
\leftrightarrow & \equiv & \rightarrow \cup \leftarrow \\
\leftrightarrow^* & \equiv & (\rightarrow \cup \leftarrow)^*
\end{array}
$$

Example

`3` $\rightarrow^*$ `3`, and `if 3 > 2 then 5 + 9 else 2 * 4` $\rightarrow^*$ `14`

Be careful with $\leftrightarrow^*$

$$a \leftrightarrow^* b \not\equiv a \leftarrow^* b \cup a \rightarrow^* b$$

For example $a \leftrightarrow^* b$ when

$$a \leftarrow a_1 \rightarrow a_2 \rightarrow a_3 \leftarrow b_2 \leftarrow b_1 \rightarrow b$$

Substitution

- This particular semantics does not use an environment.
- To express the meaning of variable substitution, we use the substitution operator.
- $[e_1/\mathsf{x}]e_2$ means "Replace all occurrences of x in e_2 with e_1."
- So, $[3/\mathsf{x}](\mathsf{2}+\mathsf{x}) \Rightarrow (\mathsf{2}+\mathsf{3})$

More formally...

$$
\begin{aligned}
[y/x]x &\Rightarrow y \\
[y/x]z &\Rightarrow z \\
[y/x](a \oplus b) &\Rightarrow [y/x]a \oplus [y/x]b \\
[y/x](\texttt{if}\ \ M\,\texttt{then}\ \ N\,\texttt{else}\ \ O) &\Rightarrow (\texttt{if}\ \ [y/x]M\,\texttt{then}\ \ [y/x]N \\
&\qquad\qquad\quad \texttt{else}\ \ [y/x]O)
\end{aligned}
$$

Substitution has to be done more carefully for `let`.

More formally...

$$
\begin{array}{ll}
[y/x](\texttt{let } x = M \texttt{ in } N) & \Rightarrow \texttt{let } x = [y/x]M \texttt{ in } N \\
[y/x](\texttt{let } z = M \texttt{ in } N) & \Rightarrow \texttt{let } z = [y/x]M \texttt{ in } [y/x]N \\
[y/x](\texttt{let rec } x = M \texttt{ in } N) & \Rightarrow \texttt{let rec } x = M \texttt{ in } N \\
[y/x](\texttt{let rec } z = M \texttt{ in } N) & \Rightarrow \texttt{let rec } z = [y/x]M \\
& \qquad\qquad \texttt{in } [y/x]N
\end{array}
$$

Example

Evaluate: `let x = 2 + 3 in let y = x * x in x + y`

```
    let x = 2 + 3 in let y = x * x in x + y
→   let x = 5 in let y = x * x in x + y
→   let y = 5 * 5 in 5 + y
→   let y = 25 in 5 + y
→   5 + 25
→   30
```

Activity

Do the Operational Semantics activity.

Term Rewriting Systems

Transition semantics can be thought of as a *term-rewriting system.* Common questions:

- Does an expression always terminate?
- Can we tell if two expressions are equal?

Church-Rosser Property: If $x \leftrightarrow^* y$ then x and y normalize to the same value.

$$\begin{tikzcd}
x \arrow[rr, leftrightarrow, "*"] \arrow[dr, "*"'] & & y \arrow[dl, "*"] \\
& z &
\end{tikzcd}$$

Example

Confluence

If $x \to y_1$ and $x \to y_2$ then y_1 and y_2 normalize to the same value.
(Confluence and the Church-Rosser Property coincide.)

$$
\begin{tikzcd}
 & x = 2+3+5 \arrow[dl] \arrow[dr] & \\
x = 5+5 \arrow[dr] & & x = 2+8 \arrow[dl] \\
 & x = 10 &
\end{tikzcd}
$$

This is also known as the “diamond property”

Who has it?

- Alonzo Church and J. Barkley Rosser proved that the λ-calculus has these properties in 1936.
- Very important for theorem provers.
- Most programming languages have this property... some of the time...
- One Benefit: you can check for equality of x and y by evaluating them.

Activity

Do the Diamond Property Activity.