

Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC
<http://www.cs.uiuc.edu/class/fa06/cs421/>

Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Semantics

- Expresses the meaning of syntax
- Static semantics
 - Meaning based only on the form of the expression without executing it
 - Usually restricted to type checking / type inference

Elsa L. Gunter

Dynamic semantics

- Method of describing meaning of executing a program
- Several different types:
 - Operational Semantics
 - Axiomatic Semantics
 - Denotational Semantics

Elsa L. Gunter

Dynamic Semantics

- Different languages better suited to different types of semantics
- Different types of semantics serve different purposes

Elsa L. Gunter

Operational Semantics

- Start with a simple notion of machine
- Describe how to execute (implement) programs of language on virtual machine, by describing how to execute each program statement (ie, following the *structure* of the program)
- Meaning of program is how its execution changes the state of the machine
- Useful as basis for implementations

Elsa L. Gunter

Axiomatic Semantics

- Also called Floyd-Hoare Logic
- Based on formal logic (first order predicate calculus)
- Axiomatic Semantics is a logical system built from *axioms* and *inference rules*
- Mainly suited to simple imperative programming languages

Elsa L. Gunter

Axiomatic Semantics

- Used to formally prove a property (*post-condition*) of the *state* (the values of the program variables) after the execution of program, assuming another property (*pre-condition*) of the state before execution
- Written :
 {Precondition} Program {Postcondition}
- Source of idea of *loop invariant*

Elsa L. Gunter

Denotational Semantics

- Construct a function \mathcal{M} assigning a mathematical meaning to each program construct
- Meaning function is compositional: meaning of construct built from meaning of parts
- Useful for proving properties of programs

Elsa L. Gunter

Transition Semantics

- Form of operational semantics
- Describes how each program construct transforms machine state by *transitions*
- Rules look like
$$(C, m) \rightarrow (C', m')$$
- C, C' is code remaining to be executed
- m, m' represent the state/store/memory/environment
 - Partial mapping from identifiers to values
 - Sometimes m (or C) not needed
- Indicates exactly one step of computation

Elsa L. Gunter

Expressions and Values

- Special class of expressions designated as *values*
 - Eg 2, 3 are values, but 2+3 is only an expression
- Memory only holds values
- Transitions stop when C is a value
- Value is the final *meaning* of original expression (in the given state)
- C, C' used for commands; E, E' for expressions; U, V for values

Elsa L. Gunter

Simple Imperative Programming Language

- $I \in \text{Identifiers}$
- $N \in \text{Numerals}$
- $B ::= \text{true} \mid \text{false} \mid B \ \& \ B \mid B \ \text{or} \ B \mid \text{not } B \mid E < E \mid E = E$
- $E ::= N \mid I \mid E + E \mid E * E \mid E - E \mid -E$
- $C ::= \text{skip} \mid C; C \mid I ::= E \mid \text{if } B \text{ then } C \text{ else } C \text{ fi} \mid \text{while } B \text{ do } C \text{ od}$

Elsa L. Gunter

Transitions for Expressions

- Identifiers: $(I, m) \rightarrow m(I)$
- Numerals are values: $(N, m) \rightarrow N$
- Notation - Function update:
 $m[I \leftarrow V] = \lambda y. \text{if } y = I \text{ then } V \text{ else } m(y)$

Elsa L. Gunter

Booleans:

- Values = {true, false}

- Operators: (short-circuit)

$$\begin{array}{l} (false \ \& \ B, m) \rightarrow false \\ (true \ \& \ B, m) \rightarrow B \end{array} \quad \frac{(B, m) \rightarrow (B'', m)}{(B \ \& \ B', m) \rightarrow (B'' \ \& \ B, m)}$$

$$\begin{array}{l} (true \ or \ B, m) \rightarrow true \\ (false \ or \ B, m) \rightarrow B \end{array} \quad \frac{(B, m) \rightarrow (B'', m)}{(B \ or \ B', m) \rightarrow (B'' \ or \ B, m)}$$

$$\begin{array}{l} (not \ true, m) \rightarrow false \\ (not \ false, m) \rightarrow true \end{array} \quad \frac{(B, m) \rightarrow (B', m)}{(not \ B, m) \rightarrow (not \ B', m)}$$

Elsa L. Gunter

Relations

$$\frac{(E, m) \rightarrow (E'', m)}{(E \sim E', m) \rightarrow (E'' \sim E', m)}$$

$$\frac{(E, m) \rightarrow (E', m)}{(V \sim E, m) \rightarrow (V \sim E', m)}$$

$(U \sim V, m) \rightarrow$ true or false, depending on whether $U \sim V$ holds or not

Elsa L. Gunter

Arithmetic Expressions

$$\frac{(E, m) \rightarrow (E'', m)}{(E \ op \ E', m) \rightarrow (E'' \ op \ E', m)}$$

$$\frac{(E, m) \rightarrow (E', m)}{(V \ op \ E, m) \rightarrow (V \ op \ E', m)}$$

$(U \ op \ V, m) \rightarrow N$ where N is the specified value for $U \ op \ V$

Elsa L. Gunter

Commands - in English

- skip is done evaluating
- When evaluating an assignment, evaluate the expression first
- If the expression being assigned is already a value, update the memory with the new value for the identifier
- When evaluating a sequence, work on the first command in the sequence first
- If the first command evaluates to a new memory (ie completes), evaluate remainder with new memory

Elsa L. Gunter

Commands

$$(skip, m) \rightarrow m$$

$$\frac{(E, m) \rightarrow (E', m)}{(I ::= E, m) \rightarrow (I ::= E', m)}$$

$$(I ::= V, m) \rightarrow m[I \leftarrow V]$$

$$\frac{(C, m) \rightarrow (C'', m')}{(C; C', m) \rightarrow (C''; C', m')} \quad \frac{(C, m) \rightarrow m'}{(C; C', m) \rightarrow (C', m')}$$

Elsa L. Gunter

If Then Else Command - in English

- If the boolean guard in and if_then_else is true, then evaluate the first branch
- If it is false, evaluate the second branch
- If the boolean guard is not a value, then start by evaluating it first.

Elsa L. Gunter

If Then Else Command

(if true then C else C' fi, m) \rightarrow (C , m)

(if false then C else C' fi, m) \rightarrow (C' , m)

$$\frac{(B,m) \rightarrow (B',m)}{\text{(if } B \text{ then } C \text{ else } C' \text{ fi, } m)} \rightarrow \text{(if } B' \text{ then } C \text{ else } C' \text{ fi, } m)$$

Elsa L. Gunter

While Command

(while B do C od, m)

\rightarrow (if B then C ; while B do C od else skip fi, m)

In English: Expand a While into a test of the boolean guard, with the true case being to do the body and the try the while loop again, and the false case being to stop.

Elsa L. Gunter

Example Evaluation

- First step:

$$\frac{}{\text{(if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \rightarrow ?}$$

Elsa L. Gunter

Example Evaluation

- First step:

$$\frac{(x > 5, \{x \rightarrow 7\}) \rightarrow ?}{\text{(if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \rightarrow ?}$$

Elsa L. Gunter

Example Evaluation

- First step:

$$\frac{\frac{(x, \{x \rightarrow 7\}) \rightarrow 7}{(x > 5, \{x \rightarrow 7\}) \rightarrow ?}}{\text{(if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \rightarrow ?}$$

Elsa L. Gunter

Example Evaluation

- First step:

$$\frac{\frac{(x, \{x \rightarrow 7\}) \rightarrow 7}{(x > 5, \{x \rightarrow 7\}) \rightarrow (7 > 5, \{x \rightarrow 7\})}}{\text{(if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \rightarrow ?}$$

Elsa L. Gunter

Example Evaluation

- First step:

$$\frac{(x, \{x \rightarrow 7\}) \rightarrow 7}{(x > 5, \{x \rightarrow 7\}) \rightarrow (7 > 5, \{x \rightarrow 7\})}$$

(if $x > 5$ then $y := 2 + 3$ else $y := 3 + 4$ fi,
 $\{x \rightarrow 7\}$)
 \rightarrow (if $7 > 5$ then $y := 2 + 3$ else $y := 3 + 4$ fi,
 $\{x \rightarrow 7\}$)

Elsa L. Gunter

Example Evaluation

- Second Step:

$$\frac{(7 > 5, \{x \rightarrow 7\}) \rightarrow \text{true}}{(\text{if } 7 > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \rightarrow (\text{if true then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\})}$$

- Third Step:

$$(\text{if true then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \rightarrow (y := 2 + 3, \{x \rightarrow 7\})$$

Elsa L. Gunter

Example Evaluation

- Fourth Step:

$$\frac{(2 + 3, \{x \rightarrow 7\}) \rightarrow 5}{(y := 2 + 3, \{x \rightarrow 7\}) \rightarrow (y := 5, \{x \rightarrow 7\})}$$

- Fifth Step:

$$(y := 5, \{x \rightarrow 7\}) \rightarrow \{y \rightarrow 5, x \rightarrow 7\}$$

Elsa L. Gunter

Example Evaluation

- Bottom Line:

$$\begin{aligned} & (\text{if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \\ & \rightarrow (\text{if } 7 > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \\ & \rightarrow (\text{if true then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi, } \{x \rightarrow 7\}) \\ & \rightarrow (y := 2 + 3, \{x \rightarrow 7\}) \\ & \rightarrow (y := 5, \{x \rightarrow 7\}) \rightarrow \{y \rightarrow 5, x \rightarrow 7\} \end{aligned}$$

Elsa L. Gunter

Transition Semantics Evaluation

- A sequence of steps with trees of justification for each step

$$(C_1, m_1) \xrightarrow{\text{tree}} (C_1, m_1) \xrightarrow{\text{tree}} (C_1, m_1) \xrightarrow{\text{tree}} \dots \xrightarrow{\text{tree}} m$$

- Let \rightarrow^* be the transitive closure of \rightarrow
- I.e., the smallest transitive relation containing \rightarrow

Elsa L. Gunter

Adding Local Declarations

- Add to expressions:
 - $E ::= \dots \mid \text{let } l = E' \text{ in } E' \mid \text{fun } l \rightarrow E \mid E E'$
 - $\text{fun } l \rightarrow E$ is a value
- Could handle local binding using state, but have assumption that evaluating expressions doesn't alter the environment
- We will use substitution here instead
- Recall: $E[E' / l]$ means replace all free occurrence of l by E' in E

Elsa L. Gunter

Call-by-value (Eager Evaluation)

$$\frac{\begin{array}{l} (\text{let } l = V \text{ in } E, m) \rightarrow (E[V/l], m) \\ (E, m) \rightarrow (E'', m) \end{array}}{(\text{let } l = E \text{ in } E', m) \rightarrow (\text{let } l = E'' \text{ in } E')}$$

$$\frac{\begin{array}{l} ((\text{fun } l \rightarrow E) V, m) \rightarrow (E[V/l], m) \\ (E', m) \rightarrow (E'', m) \end{array}}{((\text{fun } l \rightarrow E) E', m) \rightarrow ((\text{fun } l \rightarrow E) E'', m)}$$

Elsa L. Gunter

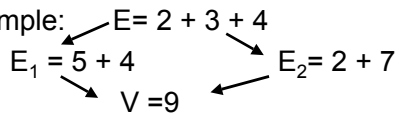
Call-by-name (Lazy Evaluation)

- $(\text{let } l = E \text{ in } E', m) \rightarrow (E'[E/l], m)$
- $((\text{fun } l \rightarrow E') E, m) \rightarrow (E'[E/l], m)$
- Question: Does it make a difference?
- It can depending on the language

Elsa L. Gunter

Church-Rosser Property

- Church-Rosser Property: If $E \rightarrow^* E_1$ and $E \rightarrow^* E_2$, if there exists a value V such that $E_1 \rightarrow V$, then $E_2 \rightarrow V$
- Also called **confluence** or **diamond property**
- Example:



Elsa L. Gunter

Does It always Hold?

- No. Languages with side-effects tend not be Church-Rosser with the combination of call-by-name and call-by-value
- Alonzo Church and Barkley Rosser proved in 1936 the λ -calculus does have it
- Benefit of Church-Rosser: can check equality of terms by evaluating them (Given evaluation strategy might not terminate, though)

Elsa L. Gunter

Transition Semantics for λ -Calculus

- Application (version 1)
 $(\lambda x . E) E' \rightarrow E[E'/x]$
- Application (version 2)
 $(\lambda x . E) V \rightarrow E[V/x]$

$$\frac{E' \rightarrow E''}{(\lambda x . E) E' \rightarrow (\lambda x . E) E''}$$

Elsa L. Gunter