

## Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC

<http://www.cs.uiuc.edu/class/sp07/cs421/>

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

## 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 or E stop when  $E$  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)
- $$\begin{array}{l} (\text{false} \ \& \ B, m) \rightarrow \text{false} \quad \frac{(B, m) \rightarrow (B'', m)}{(B \ \& \ B', m) \rightarrow (B'' \ \& \ B, m)} \\ (\text{true} \ \& \ B, m) \rightarrow (B, m) \\ (\text{true} \ \text{or} \ B, m) \rightarrow \text{true} \quad \frac{(B, m) \rightarrow (B'', m)}{(B \ \text{or} \ B', m) \rightarrow (B'' \ \text{or} \ B, m)} \\ (\text{false} \ \text{or} \ B, m) \rightarrow (B, m) \\ (\text{not true}, m) \rightarrow \text{false} \quad \frac{(B, m) \rightarrow (B', m)}{(\text{not } B, m) \rightarrow (\text{not } B', m)} \\ (\text{not false}, m) \rightarrow \text{true} \end{array}$$

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

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

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

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

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 an `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

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

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

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

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 then try the while loop again, and the false case being to stop.

Elsa L. Gunter

## Example Evaluation

---

- First step:

---

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

Elsa L. Gunter

## Example Evaluation

---

- First step:

---

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


---

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

Elsa L. Gunter

## Example Evaluation

---

- First step:

---

( $x, \{x \rightarrow 7\}$ )  $\rightarrow 7$   


---

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


---

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

Elsa L. Gunter

## Example Evaluation

---

- First step:

---

( $x, \{x \rightarrow 7\}$ )  $\rightarrow 7$   


---

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


---

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

Elsa L. Gunter

## Example Evaluation

---

- First step:

---

( $x, \{x \rightarrow 7\}$ )  $\rightarrow 7$   


---

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


---

 (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

$$(C_1, m_1) \begin{array}{c} \nabla \\ \hline \end{array} \rightarrow (C_2, m_2) \begin{array}{c} \nabla \\ \hline \end{array} \rightarrow (C_3, m_3) \begin{array}{c} \nabla \\ \hline \end{array} \rightarrow \dots \rightarrow m$$

- Let  $\rightarrow^*$  be the transitive closure of  $\rightarrow$
- ie, 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)

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

Elsa L. Gunter

## Call-by-name (Lazy Evaluation)

- $(\text{let } I = E \text{ in } E', m) \rightarrow (E[E/I], m)$
- $((\text{fun } I \rightarrow E') E, m) \rightarrow (E[E/I], m)$
- 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  $\lambda$ -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 $\lambda$ -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

## Natural vs Transition Semantics

- Natural semantics similar to transition semantics except
  - Transition semantics is relation between individual steps of computation
  - Natural semantics is relation between computation state and final result
- NS Rules look like  $(C, m) \Downarrow m'$
- TS Rules look like  $(C, m) \rightarrow (C', m') \mid m''$
- Always want  
Lemma:  $(C, m) \rightarrow^* m' \text{ iff } (C, m) \Downarrow m'$

Elsa L. Gunter

## Picture

- Transition semantics
- Natural Semantics

Elsa L. Gunter

## Why Have Both Semantics?

---

- Natural Semantics corresponds to a recursive program for evaluating
- Transition Semantics corresponds to iterative program for evaluating one step at a time
- Natural Semantics more concise but can't express nonterminating computation