

Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC

<http://www.cs.illinois.edu/class/cs421/>

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

11/3/11

1

Interpreter

- Takes abstract syntax trees as input
 - In simple cases could be just strings
- One procedure for each syntactic category (nonterminal)
 - eg one for expressions, another for commands
- If Natural semantics used, tells how to compute final value from code
- If Transition semantics used, tells how to compute next "state"
 - To get final value, put in a loop

11/3/11

2

Natural Semantics Example

- $\text{compute_exp}(\text{Var}(v), m) = \text{look_up } v \ m$
- $\text{compute_exp}(\text{Int}(n), _) = \text{Num}(n)$
- ...
- $\text{compute_com}(\text{IfExp}(b, c1, c2), m) =$
if $\text{compute_exp}(b, m) = \text{Bool}(\text{true})$
then $\text{compute_com}(c1, m)$
else $\text{compute_com}(c2, m)$

11/3/11

3

Natural Semantics Example

- $\text{compute_com}(\text{While}(b, c), m) =$
if $\text{compute_exp}(b, m) = \text{Bool}(\text{false})$
then m
else $\text{compute_com}(\text{While}(b, c), \text{compute_com}(c, m))$
- May fail to terminate - exceed stack limits
- Returns no useful information then

11/3/11

4

Transition Semantics

- Form of operational semantics
- Describes how each program construct transforms machine state by *transitions*
- Rules look like
 $(C, m) \rightarrow (C', m') \text{ or } (C, m) \rightarrow m'$
- 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

11/3/11

5

Expressions and Values

- C, C' used for commands; E, E' for expressions; U, V for values
- Special class of expressions designated as *values*
 - Eg 2, 3 are values, but 2+3 is only an expression
- Memory only holds values
 - Other possibilities exist

11/3/11

6

Evaluation Semantics

- Transitions successfully stops when E/C is a value/memory
- Evaluation fails if no transition possible, but not at value/memory
- Value/memory is the final *meaning* of original expression/command (in the given state)
- Coarse semantics: final value / memory
- More fine grained: whole transition sequence

11/3/11

7

Simple Imperative Programming Language

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

11/3/11

8

Transitions for Expressions

- Numerals are values
- Boolean values = {true, false}
- Identifiers: $(I, m) \rightarrow (m(I), m)$

11/3/11

9

Boolean Operations:

- Operators: (short-circuit)
- $$\frac{(B, m) \rightarrow (B'', m)}{(\text{false} \ \& \ B, m) \rightarrow (\text{false}, m)} \quad \frac{(B, m) \rightarrow (B'', m)}{(B \ \& \ B', m) \rightarrow (B'' \ \& \ B', m)}$$
- $$\frac{(B, m) \rightarrow (B'', m)}{(\text{true} \ \text{or} \ B, m) \rightarrow (\text{true}, m)} \quad \frac{(B, m) \rightarrow (B'', m)}{(B \ \text{or} \ B', m) \rightarrow (B'' \ \text{or} \ B', m)}$$
- $$\frac{(B, m) \rightarrow (B', m)}{(\text{not} \ \text{true}, m) \rightarrow (\text{false}, m)} \quad \frac{(B, m) \rightarrow (B', m)}{(\text{not} \ \text{false}, m) \rightarrow (\text{true}, m)} \quad \frac{(B, m) \rightarrow (B', m)}{(\text{not} \ B, m) \rightarrow (\text{not} \ B', m)}$$

11/3/11

10

Relations

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

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

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

11/3/11

11

Arithmetic Expressions

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

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

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

11/3/11

12

Commands - in English

- skip means 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

11/3/11

13

Commands

$$\begin{aligned}
 &(\text{skip}, m) \rightarrow m \\
 &\frac{(E, m) \rightarrow (E', m)}{(I ::= E, m) \rightarrow (I ::= E', m)} \\
 &(I ::= V, m) \rightarrow m[I \leftarrow V] \\
 &\frac{(C, m) \rightarrow (C'', m')}{(C; C', m) \rightarrow (C''; C', m')} \quad \frac{(C, m) \rightarrow m'}{(C; C', m) \rightarrow (C', m')}
 \end{aligned}$$

11/3/11

14

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.

11/3/11

15

If Then Else Command

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

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

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

11/3/11

16

While Command

$$\begin{aligned}
 &(\text{while } B \text{ do } C \text{ od}, m) \\
 &\rightarrow (\text{if } B \text{ then } C; \text{ while } B \text{ do } C \text{ od else skip fi}, m)
 \end{aligned}$$

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.

11/3/11

17

Example Evaluation

- First step:

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

11/3/11

18

Example Evaluation

- First step:

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

11/3/11

19

Example Evaluation

- First step:

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

11/3/11

20

Example Evaluation

- First step:

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

11/3/11

21

Example Evaluation

- First step:

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

11/3/11

22

Example Evaluation

- Second Step:

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

- Third Step:

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

11/3/11

23

Example Evaluation

- Fourth Step:

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

- Fifth Step:

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

11/3/11

24

Example Evaluation

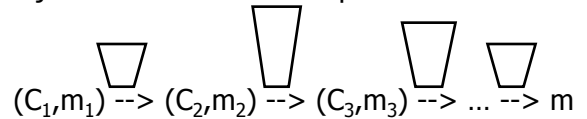
- Bottom Line:
- ```
(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,
 {x -> 7})
--> (if 7 > 5 then y:=2 + 3 else y:=3 + 4 fi,
 {x -> 7})
-->(if true then y:=2 + 3 else y:=3 + 4 fi,
 {x -> 7})
-->(y:=2+3, {x->7})
--> (y:=5, {x->7}) --> {y -> 5, x -> 7}
```

11/3/11

25

## Transition Semantics Evaluation

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



- Let  $\text{-->}^*$  be the transitive closure of  $\text{-->}$
- Ie, the smallest transitive relation containing  $\text{-->}$

11/3/11

26

## Adding Local Declarations

- Add to expressions:
- $E ::= \dots \mid \text{let } I = E \text{ in } E' \mid \text{fun } I \rightarrow E \mid E E'$
- $\text{fun } I \rightarrow E$  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
- Notation:**  $E[E'/I]$  means replace all free occurrence of  $I$  by  $E'$  in  $E$

11/3/11

27

## Call-by-value (Eager Evaluation)

$(\text{let } I = V \text{ in } E, m) \text{ -->} (E[V/I], m)$

$$\frac{(E, m) \text{ -->} (E'', m)}{(\text{let } I = E \text{ in } E', m) \text{ -->} (\text{let } I = E' \text{ in } E')}$$

$((\text{fun } I \rightarrow E) V, m) \text{ -->} (E[V/I], m)$

$$\frac{(E', m) \text{ -->} (E'', m)}{((\text{fun } I \rightarrow E) E', m) \text{ -->} ((\text{fun } I \rightarrow E) E'', m)}$$

11/3/11

28

## Call-by-name (Lazy Evaluation)

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

11/3/11

29

## Church-Rosser Property

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

- Example:
- 

11/3/11

30

## 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)

11/3/11

31

## Transition Semantics for $\lambda$ -Calculus

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

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

11/3/11

32