Programming Languages and Compilers (CS 421)

Reza Zamani

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

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

Transition Semantics

- Form of operational semantics
- Describes how each program construct transforms machine state by *transitions*
- Rules look like

$$(C, m) \text{ --> } (C', m')$$

- *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

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

Simple Imperative Programming Language

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

Transitions for Expressions

- Identifiers: $(I, m) \dashrightarrow m(I)$
- Numerals are values: $(N, m) \dashrightarrow N$

- Notation - Function update:
- $m[I \leftarrow\!\!- V] = \lambda\, y.\ \text{if } y = I \text{ then } V \text{ else } m(y)$

Booleans:

- Values = {true, false}
- Operators: (short-circuit)

(false & *B*, *m*) --> false

(true & *B*, *m*) --> (*B*,*m*)

$$\frac{(B, m) \text{ --> } (B'', m)}{(B \mathbin{\&} B', m) \text{ --> } (B'' \mathbin{\&} B, m)}$$

(true or *B*, *m*) --> true

(false or *B*, *m*) --> (*B*,*m*)

$$\frac{(B, m) \text{ --> } (B'', m)}{(B \text{ or } B', m) \text{ --> } (B'' \text{ or } B, m)}$$

(not true, m) --> false

(not false, m) --> true

$$\frac{(B, m) \text{ --> } (B', m)}{(\text{not } B, m) \text{ --> } (\text{not } B', m)}$$

Relations

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

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

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

Arithmetic Expressions

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

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

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

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

Commands

$$(\text{skip}, m) \dashrightarrow m$$

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

$$(I := V,m) \text{ --> } m[I \text{ <-- } V]$$

$$\frac{(C,m) \text{ --> } (C'',m')}{(C;C',\, m) \text{ --> } (C'';C',m')} \qquad \frac{(C,m) \text{ --> } m'}{(C;C',\, m) \text{ --> } (C',m')}$$

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.

If Then Else Command

(if true then C else C' fi, m) --> (C, m)

(if false then C else C' fi, m) --> (C', m)

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

While Command

(while *B* do *C* od, *m*)

--> (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.

Example Evaluation

- First step:

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

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

{x -> 7})

--> ?

Example Evaluation

- First step:

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

Example Evaluation

- First step:

$$\dfrac{\dfrac{(\texttt{x},\{\texttt{x} \to 7\}) \dashrightarrow 7}{(\texttt{x} > 5,\ \{\texttt{x} \to 7\}) \dashrightarrow\ ?}}{\begin{array}{c}(\texttt{if x > 5 then y:= 2 + 3 else y:=3 + 4 fi},\\ \{\texttt{x} \to 7\})\\ \dashrightarrow\ ?\end{array}}$$

Example Evaluation

- First step:

$$\frac{\dfrac{(x, \{x \text{ -> } 7\}) \text{ --> } 7}{(x > 5,\ \{x \text{ -> } 7\}) \text{ --> } (7 > 5,\ \{x \text{ -> } 7\})}}{\begin{array}{c}(\text{if } x > 5 \text{ then } y\text{:= } 2 + 3 \text{ else } y\text{:=}3 + 4 \text{ fi},\\ \{x \text{ -> } 7\})\\ \text{--> ?}\end{array}}$$

Example Evaluation

- First step:

$$\frac{\dfrac{(x,\{x \text{ -> } 7\}) \text{ --> } 7}{(x > 5,\ \{x \text{ -> } 7\}) \text{ --> } (7 > 5,\ \{x \text{ -> } 7\})}}{\begin{array}{c}(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},\\ \{x \text{ -> } 7\})\\ \text{--> } (\text{if } 7 > 5 \text{ then } y{:=}2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},\\ \{x \text{ -> } 7\})\end{array}}$$

Example Evaluation

- Second Step:

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

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

- Third Step:

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

Example Evaluation

- Fourth Step:

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

- Fifth Step:

(y:=5, {x->7}) --> {y -> 5, x -> 7}

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}

Transition Semantics Evaluation

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

$(C_1,m_1) \dashrightarrow (C_2,m_2) \dashrightarrow (C_3,m_3) \dashrightarrow \ldots \dashrightarrow m$

- Let -->* be the transitive closure of -->
- Ie, the smallest transitive relation containing -->

Adding Local Declarations

- Add to expressions:
- E ::= ... | let I = E in E' | fun I -> E | E E'
- fun I -> 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

Call-by-value (Eager Evaluation)

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

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

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

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

Call-by-name (Lazy Evaluation)

- (let I = E in E', m) --> (E' [E / I],m)

- ((fun I -> E') E, m) --> (E' [E / I],m)

- Question: Does it make a difference?
  - It depends on the language

Church-Rosser Property

- Church-Rosser Property:

  Assume E-->* E_1 and E-->* E_2. If there exists a value V such that E_1 -->V, then E_2 --> V
- Also called **confluence** or **diamond property**
- Example: E= 2 + 3 + 4

E_1 = 5 + 4 E_2= 2 + 7

V =9

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)