Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC

https://courses.engr.illinois.edu/cs421/sp2023

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

Example: If Then Else Rule

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

Example: If Then Else Rule

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

Example: Arith Relation

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

Example: Identifier(s)

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

Example: Arith Relation

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

Example: If Then Else Rule

7 > 5 = true

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true   ⇓ ?   .

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

{x -> 7}) ⇓ ?

Example: Assignment

7 > 5 = true   (2+3, {x->7})⇓?

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true   ⇓ ?

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

{x -> 7}) ⇓ ?

Example: Arith Op

? + ? = ?

(2,{x->7})⇓?   (3,{x->7}) ⇓?

7 > 5 = true   (2+3, {x->7})⇓?

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true   ⇓?   .

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

{x -> 7}) ⇓ ?

Example: Numerals

2 + 3 = 5

(2,{x->7})⇓2   (3,{x->7}) ⇓3

7 > 5 = true   (2+3, {x->7})⇓?

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true   ⇓ ?

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

{x -> 7}) ⇓ ?

Example: Arith Op

2 + 3 = 5

(2,{x->7})⇓2   (3,{x->7}) ⇓3

7 > 5 = true   (2+3, {x->7})⇓5

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true   ⇓?

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

{x -> 7}) ⇓ ?

Example: Assignment

2 + 3 = 5

(2,{x->7})⇓2   (3,{x->7}) ⇓3

7 > 5 = true   (2+3, {x->7})⇓5

(x,{x->7})⇓7   (5,{x->7})⇓5   (y:= 2 + 3,{x-> 7})

(x > 5, {x -> 7})⇓true   ⇓ {x->7, y->5}

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

{x -> 7}) ⇓?

Example: If Then Else Rule

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

Let in Command

$$\frac{(E,m) \Downarrow v \quad (C,m[I\text{<-}v]) \Downarrow m'}{(\text{let } I = E \text{ in } C,\ m) \Downarrow m''}$$

Where $m''(y) = m'(y)$ for $y \neq I$ and $m''(I) = m(I)$ if $m(I)$ is defined, and $m''(I)$ is undefined otherwise

Example

$$\frac{(5,\{x\text{->}17\}) \Downarrow 5 \quad \dfrac{\dfrac{(x,\{x\text{->}5\}) \Downarrow 5 \quad (3,\{x\text{->}5\}) \Downarrow 3}{(x+3,\{x\text{->}5\}) \Downarrow 8}}{(x:=x+3,\{x\text{->}5\}) \Downarrow \{x\text{->}8\}}}{(\text{let } x = 5 \text{ in } (x:=x+3), \{x \text{->} 17\}) \Downarrow ?}$$

Example

$$\frac{(5,\{x\text{->}17\}) \Downarrow 5 \quad \dfrac{\dfrac{(x,\{x\text{->}5\}) \Downarrow 5 \quad (3,\{x\text{->}5\}) \Downarrow 3}{(x+3,\{x\text{->}5\}) \Downarrow 8}}{(x:=x+3,\{x\text{->}5\}) \Downarrow \{x\text{->}8\}}}{(\text{let } x = 5 \text{ in } (x:=x+3), \{x \text{->} 17\}) \Downarrow \{x\text{->}17\}}$$

Comment

- Simple Imperative Programming Language introduces variables *implicitly* through assignment
- The let-in command introduces scoped variables *explictly*
- Clash of constructs apparent in awkward semantics

Interpretation Versus Compilation

- A **compiler** from language L1 to language L2 is a program that takes an L1 program and for each piece of code in L1 generates a piece of code in L2 of same meaning
- An **interpreter** of L1 in L2 is an L2 program that executes the meaning of a given L1 program
- Compiler would examine the body of a loop once; an interpreter would examine it every time the loop was executed

Interpreter

- An *Interpreter* represents the operational semantics of a language L1 (source language) in the language of implementation L2 (target language)
- Built incrementally
  - Start with literals
  - Variables
  - Primitive operations
  - Evaluation of expressions
  - Evaluation of commands/declarations

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

Natural Semantics Example

- compute_exp (Var(v), m) = look_up v m
- compute_exp (Int(n), _) = Num (n)
- …
- compute_com(IfExp(b,c1,c2),m) =

  if compute_exp (b,m) = Bool(true)

  then compute_com (c1,m)

  else compute_com (c2,m)

Natural Semantics Example

- compute_com(While(b,c), m) =

  if compute_exp (b,m) = Bool(false)

  then m

  else compute_com

  (While(b,c), compute_com(c,m))

- May fail to terminate - exceed stack limits
- Returns no useful information then

Transition Semantics

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

  (C, *m*) --> (C', *m'*) or (C,*m*) --> *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

- *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

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

1525 minutes

Simple Imperative Programming Language

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

Transitions for Expressions

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

Boolean Operations:

- Operators: (short-circuit)

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

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

(true or B, m) --> (true,m)
(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,m)
(not false, m) --> (true,m)

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

Relations

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

$$\frac{(E, m) \text{ --> } (E',m)}{(V \sim E, m) \text{ --> } (V\sim E',m)}$$

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

Arithmetic Expressions

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

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

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

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

$$(\text{skip},\ m) \text{ --> } m$$

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

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

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