Programming Languages and Compilers (CS 421)

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

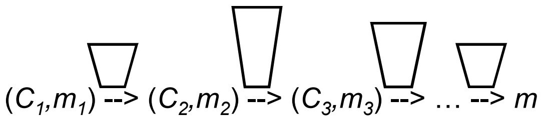
Based in part on slides by Mattox Beckman, as updated by Vikram Adve and Gul Agha

Natural Semantics

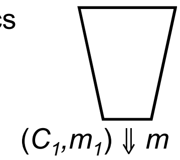
- Similar to transition semantics except
  - Transition semantics is relation between individual steps of computation
  - Natural semantics is relation between computation state and final result
- Rules look like (C, m) ⇓ m'
- Always want
  Lemma: (C,m) -->* m' iff (C, m) ⇓ m'

Picture

- Transition semantics



- Natural Semantics



Natural Semantics of Atomic Expressions

- Same as Transition
- Identifiers: $(I,m) \Downarrow m(I)$
- Numerals are values: $(N,m) \Downarrow N$
- Booleans: (true,m) ⇓ true
  (false ,m) ⇓ false

Booleans:

$$\frac{(B, m) \Downarrow \text{false}}{(B \mathbin{\&} B', m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{true} \quad (B', m) \Downarrow b}{(B \mathbin{\&} B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(B \text{ or } B', m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{true} \quad (B', m) \Downarrow b}{(B \text{ or } B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(\text{not } B, m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{false}}{(\text{not } B, m) \Downarrow \text{true}}$$

Relations

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \sim V = b}{(E \sim E', m) \Downarrow b}$$

- By $U \sim V = b$, we mean does (the meaning of) the relation ~ hold on the meaning of U and V
- May be specified by a mathematical expression/equation or rules matching U and V

Arithmetic Expressions

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \text{ op } V = N}{(E \text{ op } E', m) \dashrightarrow (E'' \text{ op } E', m)}$$

where *N* is the specified value for *U op V*

Elsa L. Gunter

Commands

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

$$\frac{(E,m) \Downarrow V}{(I{::=}E,m) \Downarrow m[I \leftarrow V]}$$

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

Elsa L. Gunter

If Then Else Command

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

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

Elsa L. Gunter

While Command

$$\frac{(B,m) \Downarrow \text{false}}{(\text{while } B \text{ do } C \text{ od}, m) \Downarrow m}$$

$$\frac{(B,m)\Downarrow\text{true} \quad (C,m)\Downarrow m' \quad (\text{while } B \text{ do } C \text{ od}, m')\Downarrow m''}{(\text{while } B \text{ do C od}, m) \Downarrow m''}$$

Elsa L. Gunter

Example

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

Elsa L. Gunter

Example

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

Elsa L. Gunter

Example

(x,{x->7})⇓? (5,{x->})⇓?

(x > 5, {x -> 7})⇓?

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

Example

(x,{x->7})⇓7 (5,{x->})⇓5

(x > 5, {x -> 7})⇓?

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

Example

(x,{x->7})⇓7 (5,{x->})⇓5

(x > 5, {x -> 7})⇓true

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

Example

(x,{x->7})⇓7 (5,{x->})⇓5

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

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

Example

(2+3, {x->7})⇓?

(x,{x->7})⇓7 (5,{x->})⇓5

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

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

Example

(2,{x->7})⇓? (3,{x->7}) ⇓?

(2+3, {x->7})⇓?

(x,{x->7})⇓7 (5,{x->})⇓5

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

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

Example

(2,{x->7})⇓2 (3,{x->7}) ⇓3
———————————————
(2+3, {x->7})⇓?
———————————————
(x,{x->7})⇓7 (5,{x->})⇓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}) ⇓ ?

Elsa L. Gunter

Example

(2,{x->7})⇓2 (3,{x->7}) ⇓3
———————————————
(2+3, {x->7})⇓5
———————————————
(x,{x->7})⇓7 (5,{x->})⇓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}) ⇓ ?

Elsa L. Gunter

Example

(2,{x->7})⇓2 (3,{x->7}) ⇓3
———————————————
(2+3, {x->7})⇓5
———————————————
(x,{x->7})⇓7 (5,{x->})⇓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}) ⇓?

Elsa L. Gunter

Example

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

Elsa L. Gunter

Let in Command

(*E*,*m*) ⇓*V* (*C*,*m*[*x*<-*V*]) ⇓ *m'*
———————————————
(let *x* = *E* in *C, m*) ⇓ *m"*

Where *m''*(*y*) = *m'*(*y*) for *y*≠ *x* and
m''(*x*) = *m*(*x*) if *m*(*x*) is defined,
and *m''*(*x*) is undefined otherwisei

Elsa L. Gunter

Example

(x,{x->5}) ⇓ 5 (3,{x->5}) ⇓ 3
———————————————
(x+3,{x->5}) ⇓ 8
———————————————
(5,{x->17}) ⇓ 5 | (x:=x+3,{x->5}) ⇓ {x->8}
———————————————
(let x = 5 in (x:=x+3), {x -> 17}) ⇓ ?

Elsa L. Gunter

Example

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

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))

```
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

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