

Programming Languages and Compilers (CS 421)



Elsa L Gunter

2112 SC, UIUC

<http://courses.engr.illinois.edu/cs421>

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



Natural Semantics

- Aka Structural Operational Semantics, aka “Big Step Semantics”
- Provide value for a program by rules and derivations, similar to type derivations
- Rule conclusions look like

$$(C, m) \Downarrow m'$$

or

$$(E, m) \Downarrow v$$



Simple Imperative Programming Language

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



Natural Semantics of Atomic Expressions

- Identifiers: $(I, m) \Downarrow m(I)$
- Numerals are values: $(N, m) \Downarrow N$
- Booleans: $(\text{true}, m) \Downarrow \text{true}$
 $(\text{false}, m) \Downarrow \text{false}$



Booleans:

$$\frac{(B, m) \Downarrow \text{false}}{(B \& B', m) \Downarrow \text{false}}$$

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

$$\frac{(B, m) \Downarrow \text{true}}{(B \text{ or } B', m) \Downarrow \text{true}}$$

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

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

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



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 \sim hold on the meaning of U and V
- May be specified by a mathematical expression/equation or rules matching U and V



Arithmetic Expressions

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

where N is the specified value for $U \text{ op } V$



Commands

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

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

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



If Then Else Command

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

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



While Command

$$(B, m) \Downarrow \text{false}$$

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

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



Example: If Then Else Rule

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



Example: If Then Else Rule

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

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



Example: Arith Relation

? > ? = ?

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

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

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



Example: Identifier(s)

$7 > 5 = \text{true}$

$(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5$

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

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



Example: Arith Relation

$7 > 5 = \text{true}$

$(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5$

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

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

Example: If Then Else Rule

$7 > 5 = \text{true}$

$\frac{(x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5}{(x > 5, \{x \rightarrow 7\}) \Downarrow \text{true}}$

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

$\Downarrow ?$

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

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

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

Example: Assignment

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

Example: Numerals

$$\begin{array}{r}
 2 + 3 = 5 \\
 \hline
 (2, \{x \rightarrow 7\}) \Downarrow 2 \quad (3, \{x \rightarrow 7\}) \Downarrow 3 \\
 \hline
 (2+3, \{x \rightarrow 7\}) \Downarrow ? \\
 \hline
 (y := 2 + 3, \{x \rightarrow 7\}) \\
 \Downarrow ? \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 7 > 5 = \text{true} \\
 \hline
 (x, \{x \rightarrow 7\}) \Downarrow 7 \quad (5, \{x \rightarrow 7\}) \Downarrow 5 \\
 \hline
 (x > 5, \{x \rightarrow 7\}) \Downarrow \text{true} \\
 \hline
 \end{array}$$

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

Example: Arith Op

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

Example: Assignment

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

Example: If Then Else Rule

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



Let in Command

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

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

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



Example

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



Comment

- Simple Imperative Programming Language introduces variables *implicitly* through assignment
- The let-in command introduces scoped variables *explicitly*
- 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

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



Natural Semantics Example

- $\text{compute_com}(\text{While}(b,c), m) =$
if $\text{compute_exp}(b,m) = \text{Bool}(\text{false})$
then m
else compute_com
 $(\text{While}(b,c), \text{compute_com}(c,m))$
- 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
$$(C, m) \rightarrow (C', m') \quad \text{or} \quad (C, m) \rightarrow m'$$
- 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



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



Simple Imperative Programming Language

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



Transitions for Expressions

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



Boolean Operations:

- Operators: (short-circuit)

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



Relations

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

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

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



Arithmetic Expressions

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

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

$(U \text{ op } V, m) \dashrightarrow (N, m)$ 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

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

$$\frac{(E, m) \dashrightarrow (E', m)}{(I ::= E, m) \dashrightarrow (I ::= E', m)}$$

$$(I ::= V, m) \dashrightarrow m[I \leftarrow V]$$

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



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

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

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

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



What should while transition to?

$(\text{while } B \text{ do } C \text{ od}, m) \rightarrow ?$



Wrong! BAD

$$(B, m) \rightarrow (B', m)$$

$$(\text{while } B \text{ do } C \text{ od}, m) \not\rightarrow (\text{while } B' \text{ do } C \text{ od}, m)$$



While Command

$(\text{while } B \text{ do } C \text{ od}, m) \dashrightarrow$

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

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:

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



Example Evaluation

- First step:

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

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



Example Evaluation

- First step:

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

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

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

$\{x \rightarrow 7\}$)

$\rightarrow ?$



Example Evaluation

- First step:

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

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

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

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

$$\rightarrow ?$$



Example Evaluation

- First step:

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

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

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

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



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:

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



Example Evaluation

- Fourth Step:

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

- Fifth Step:

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



Example Evaluation

- Bottom Line:

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

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

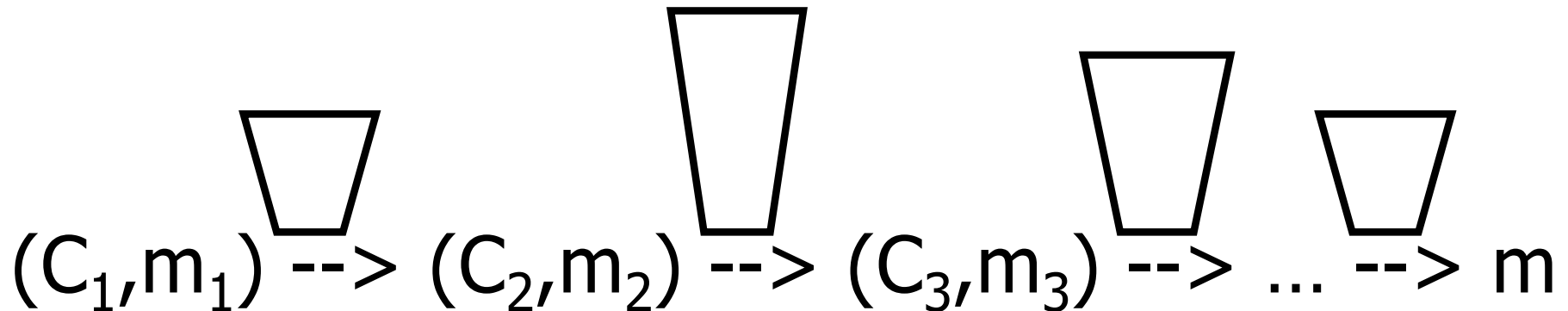
--> (if true then $y := 2 + 3$ else $y := 3 + 4$ fi,
 $\{x \rightarrow 7\}$)

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

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

Transition Semantics Evaluation

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



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

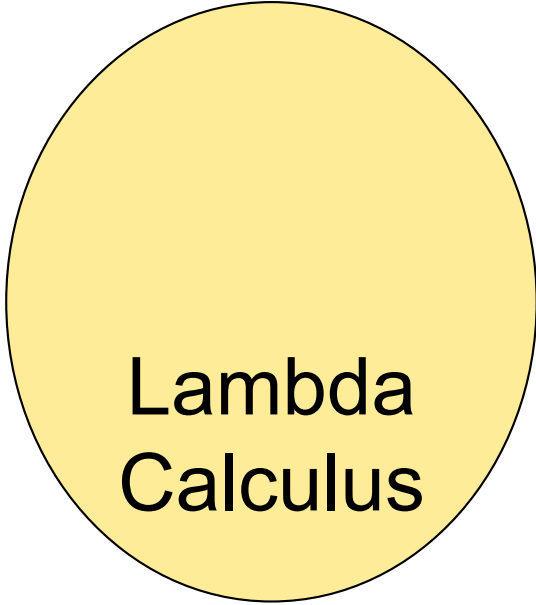


Programming Languages & Compilers

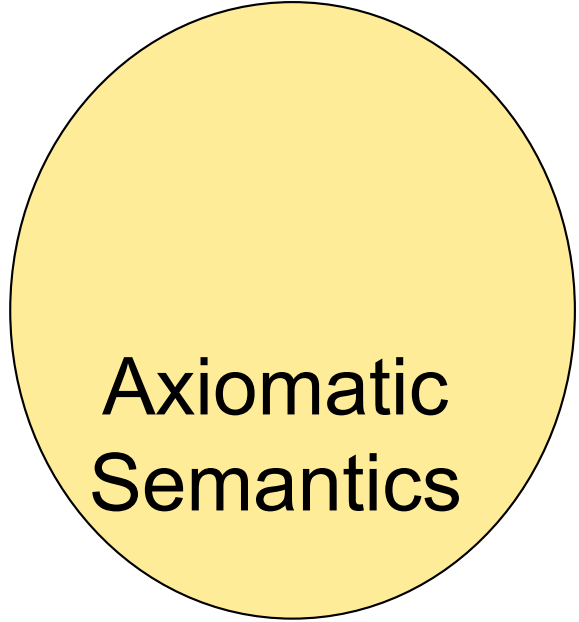
III : Language Semantics



Operational
Semantics



Lambda
Calculus



Axiomatic
Semantics



Lambda Calculus - Motivation

- Aim is to capture the essence of functions, function applications, and evaluation
- λ -calculus is a theory of computation
- “The Lambda Calculus: Its Syntax and Semantics”. H. P. Barendregt. North Holland, 1984



Lambda Calculus - Motivation

- All *sequential programs* may be viewed as functions from input (initial state and input values) to output (resulting state and output values).
- λ -calculus is a mathematical formalism of functions and functional computations
- Two flavors: typed and untyped



Untyped λ -Calculus

- Only three kinds of expressions:
 - Variables: x, y, z, w, \dots
 - Abstraction: $\lambda x. e$
(Function creation, think `fun x -> e`)
 - Application: $e_1 e_2$
 - Parenthesized expression: (e)



Untyped λ -Calculus Grammar

- Formal BNF Grammar:

- $\langle \text{expression} \rangle ::= \langle \text{variable} \rangle$

- | $\langle \text{abstraction} \rangle$

- | $\langle \text{application} \rangle$

- | $(\langle \text{expression} \rangle)$

- $\langle \text{abstraction} \rangle$

- $::= \lambda \langle \text{variable} \rangle . \langle \text{expression} \rangle$

- $\langle \text{application} \rangle$

- $::= \langle \text{expression} \rangle \langle \text{expression} \rangle$



Untyped λ -Calculus Terminology

- **Occurrence**: a location of a subterm in a term
- **Variable binding**: $\lambda x. e$ is a binding of x in e
- **Bound occurrence**: all occurrences of x in $\lambda x. e$
- **Free occurrence**: one that is not bound
- **Scope of binding**: in $\lambda x. e$, all occurrences in e not in a subterm of the form $\lambda x. e'$ (same x)
- **Free variables**: all variables having free occurrences in a term



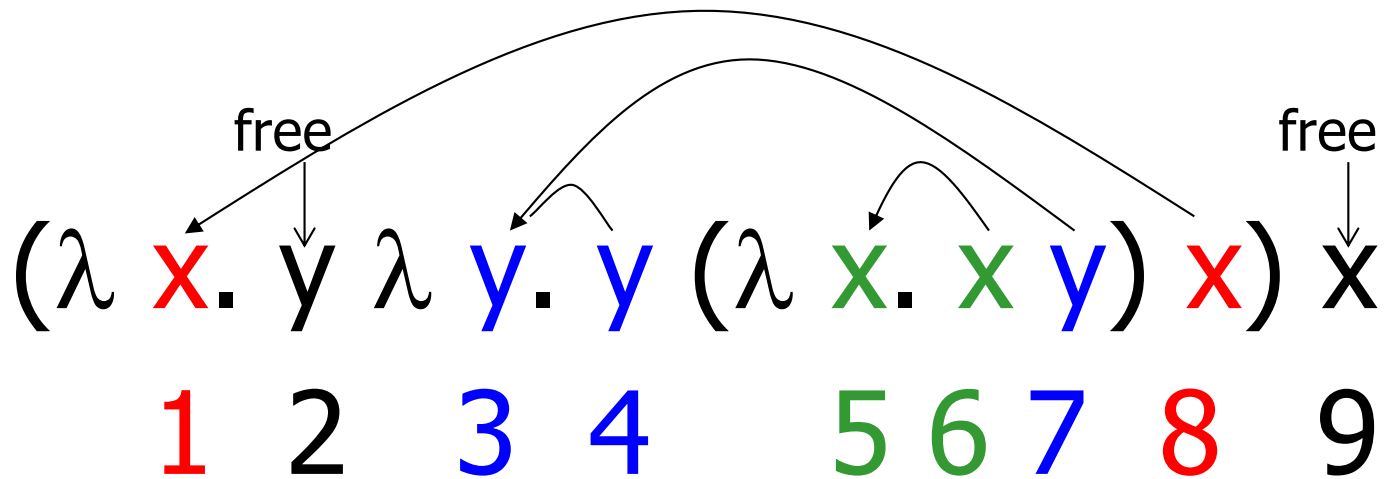
Example

- Label occurrences and scope:

$$\begin{array}{cccccccccc} (\lambda & x. & y & \lambda & y. & y & (\lambda & x. & x & y) & x) & x \\ 1 & 2 & 3 & 4 & & & 5 & 6 & 7 & 8 & 9 \end{array}$$

Example

- Label occurrences and scope:





Untyped λ -Calculus

- How do you compute with the λ -calculus?
- Roughly speaking, by substitution:
- $(\lambda x. e_1) e_2 \Rightarrow^* e_1 [e_2 / x]$
- * Modulo all kinds of subtleties to avoid free variable capture

Transition Semantics for λ -Calculus

$$\frac{E \rightarrow E''}{EE' \twoheadrightarrow E''E'}$$

- Application (version 1 - Lazy Evaluation)

$$(\lambda x. E) E' \twoheadrightarrow E[E'/x]$$

- Application (version 2 - Eager Evaluation)

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

$$\frac{}{(\lambda x. E) V \twoheadrightarrow E[V/x]}$$

V - variable or abstraction (value)



How Powerful is the Untyped λ -Calculus?

- The untyped λ -calculus is Turing Complete
 - Can express any sequential computation
- Problems:
 - How to express basic data: booleans, integers, etc?
 - How to express recursion?
 - Constants, `if_then_else`, etc, are conveniences; can be added as syntactic sugar



Typed vs Untyped λ -Calculus

- The *pure* λ -calculus has no notion of type: $(f f)$ is a legal expression
- Types restrict which applications are valid
- Types are not syntactic sugar! They disallow some terms
- Simply typed λ -calculus is less powerful than the untyped λ -Calculus: NOT Turing Complete (no recursion)



Uses of λ -Calculus

- Typed and untyped λ -calculus used for theoretical study of sequential programming languages
- Sequential programming languages are essentially the λ -calculus, extended with predefined constructs, constants, types, and syntactic sugar
- Ocaml is close to the λ -Calculus:

$$\begin{aligned} \text{fun } x \text{ -> } \text{exp} & \text{ --> } \lambda x. \text{exp} \\ \text{let } x = e_1 \text{ in } e_2 & \text{ --> } (\lambda x. e_2)e_1 \end{aligned}$$



α Conversion

1. α -conversion:
2. $\lambda x. \text{exp} \xrightarrow{\alpha} \lambda y. (\text{exp} [y/x])$
3. Provided that
 1. y is not free in exp
 2. No free occurrence of x in exp becomes bound in exp when replaced by y

$$\lambda x. x (\lambda y. x y) \xrightarrow{\alpha} \lambda y. y (\lambda y. y y)$$

α Conversion Non-Examples

1. Error: y is not free in term second

$$\lambda x. x y \not\rightarrow_{\alpha} \lambda y. y y$$

2. Error: free occurrence of x becomes bound in wrong way when replaced by y

$$\lambda x. \underbrace{\lambda y. x y}_{\text{exp}} \not\rightarrow_{\alpha} \lambda y. \underbrace{\lambda y. y y}_{\text{exp}[y/x]}$$

But $\lambda x. (\lambda y. y) x \rightarrow_{\alpha} \lambda y. (\lambda y. y) y$

And $\lambda y. (\lambda y. y) y \rightarrow_{\alpha} \lambda x. (\lambda y. y) x$



Congruence

- Let \sim be a relation on lambda terms. \sim is a **congruence** if
- it is an equivalence relation
- If $e_1 \sim e_2$ then
 - $(e e_1) \sim (e e_2)$ and $(e_1 e) \sim (e_2 e)$
 - $\lambda x. e_1 \sim \lambda x. e_2$



α Equivalence

- α equivalence is the smallest congruence containing α conversion
- One usually treats α -equivalent terms as equal - i.e. use α equivalence classes of terms



Example

Show: $\lambda x. (\lambda y. y x) x \sim_{\alpha} \lambda y. (\lambda x. x y) y$

■ $\lambda x. (\lambda y. y x) x \dashrightarrow_{\alpha} \lambda z. (\lambda y. y z) z$ so

$\lambda x. (\lambda y. y x) x \sim_{\alpha} \lambda z. (\lambda y. y z) z$

■ $(\lambda y. y z) \dashrightarrow_{\alpha} (\lambda x. x z)$ so

$(\lambda y. y z) \sim_{\alpha} (\lambda x. x z)$ so

$(\lambda y. y z) z \sim_{\alpha} (\lambda x. x z) z$ so

$\lambda z. (\lambda y. y z) z \sim_{\alpha} \lambda z. (\lambda x. x z) z$

■ $\lambda z. (\lambda x. x z) z \dashrightarrow_{\alpha} \lambda y. (\lambda x. x y) y$ so

$\lambda z. (\lambda x. x z) z \sim_{\alpha} \lambda y. (\lambda x. x y) y$

■ $\lambda x. (\lambda y. y x) x \sim_{\alpha} \lambda y. (\lambda x. x y) y$