

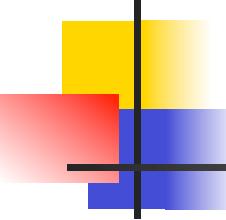
Programming Languages and Compilers (CS 421)



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



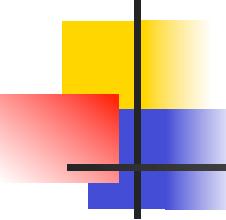
Natural Semantics

- 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

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

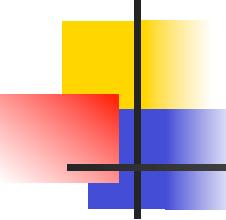
or

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



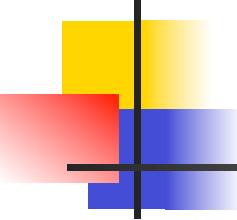
Simple Imperative Programming Language

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



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:

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

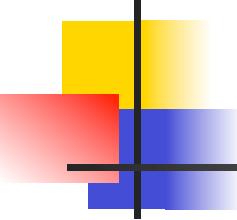
$$\frac{(B, m) \Downarrow \text{true} \quad (B', m) \Downarrow b}{(B \& B', m) \Downarrow b}$$

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

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

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

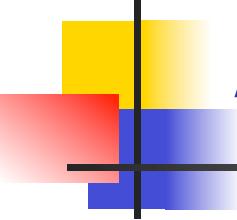
$$\frac{(B, m) \Downarrow \text{false}}{(\text{not } B, m) \Downarrow \text{true}}$$



Relations

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

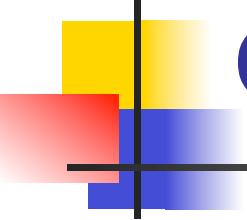
- By $U \sim V = b$, we mean does (the meaning of) the relation \sim hold on the meaning of U and V
- May be specified by a mathematical expression/equation or rules matching U and V



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 $U \text{ op } V$

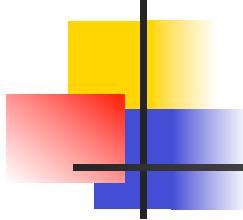


Commands

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

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

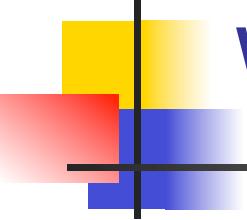
Sequencing:
$$\frac{(C,m) \Downarrow m' \quad (C',m') \Downarrow m''}{(C;C', m) \Downarrow m''}$$



If Then Else Command

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

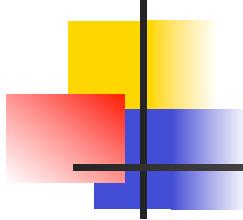
$$\frac{(B,m) \Downarrow \text{false} \quad (C',m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi, } m) \Downarrow m'}$$



While Command

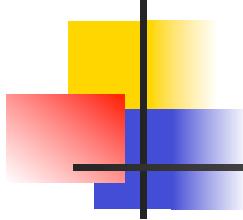
$$\frac{(B,m) \Downarrow \text{false}}{(\text{while } B \text{ do } C \text{ od}, m) \Downarrow m}$$

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



Example: If Then Else Rule

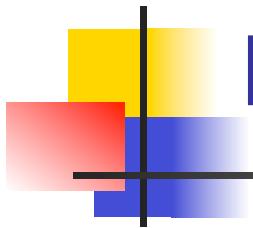
$$(\text{if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi,} \\ \{x \rightarrow 7\}) \Downarrow ?$$



Example: If Then Else Rule

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

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



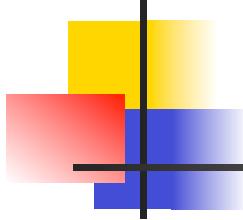
Example: Arith Relation

? > ? = ?

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

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

$\{x > 7\}) \Downarrow ?$



Example: Identifier(s)

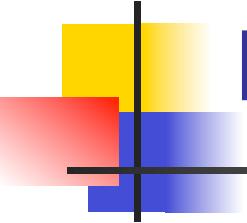
$7 > 5 = \text{true}$

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

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

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

$\{x > 7\}) \Downarrow ?$



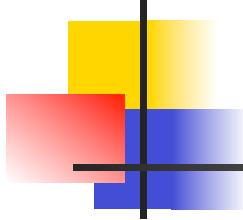
Example: Arith Relation

$$7 > 5 = \text{true}$$

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

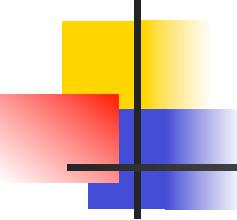
$$\frac{}{\text{(if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi,}}$$

$$\{x > 7\} \Downarrow ?$$



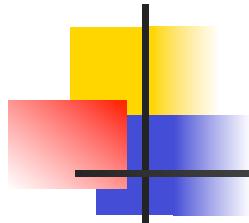
Example: If Then Else Rule

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



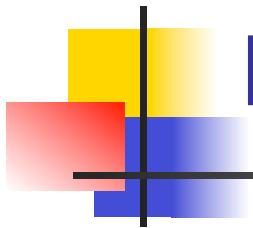
Example: Assignment

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



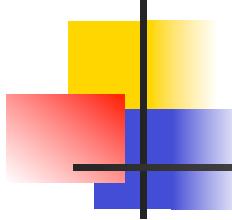
Example: Arith Op

$$\begin{array}{c}
 ? + ? = ?
 \\[1ex]
 \overline{(2,\{x>7\})\Downarrow? \quad (3,\{x>7\})\Downarrow?} \\
 \hline
 7 > 5 = \text{true} \qquad \qquad \qquad \overline{(2+3, \{x>7\})\Downarrow?} \\
 \overline{(x,\{x>7\})\Downarrow 7 \quad (5,\{x>7\})\Downarrow 5} \qquad \qquad \qquad (y := 2 + 3, \{x > 7\}) \\
 \hline
 \overline{(x > 5, \{x -> 7\})\Downarrow \text{true}} \qquad \qquad \qquad \Downarrow? \\
 \hline
 (\text{if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi,} \\
 \qquad \qquad \qquad \{x -> 7\}) \Downarrow ?
 \end{array}$$



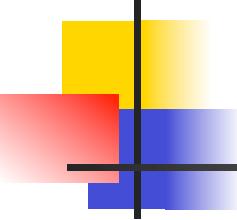
Example: Numerals

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



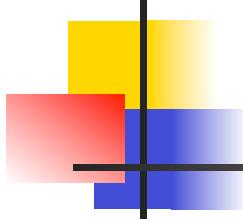
Example: Arith Op

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



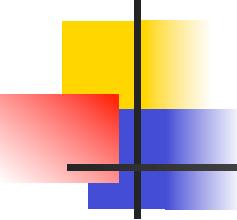
Example: Assignment

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



Example: If Then Else Rule

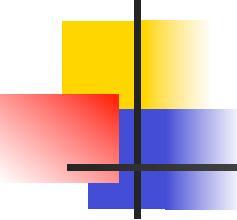
$$\frac{\frac{2 + 3 = 5}{(2, \{x > 7\}) \Downarrow 2 \quad (3, \{x > 7\}) \Downarrow 3} \quad 7 > 5 = \text{true}}{\frac{(2+3, \{x > 7\}) \Downarrow 5}{(y := 2 + 3, \{x > 7\}) \Downarrow \{x > 7, y > 5\}}} \quad \frac{(x, \{x > 7\}) \Downarrow 7 \quad (5, \{x > 7\}) \Downarrow 5}{(x > 5, \{x > 7\}) \Downarrow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



Let in Command

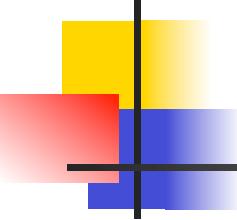
$$\frac{(E, m) \Downarrow \vee (C, m[I \leftarrow v]) \Downarrow m'}{(\text{let } I = E \text{ in } C, m) \Downarrow m''}$$

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



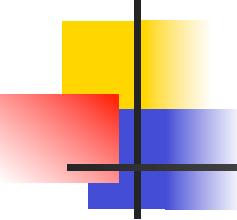
Example

$$\frac{\begin{array}{c} (x, \{x > 5\}) \Downarrow 5 \quad (3, \{x > 5\}) \Downarrow 3 \\ \hline (x+3, \{x > 5\}) \Downarrow 8 \end{array}}{(5, \{x > 17\}) \Downarrow 5 \quad (x := x + 3, \{x > 5\}) \Downarrow \{x > 8\}}$$
$$\frac{(5, \{x > 17\}) \Downarrow 5 \quad (x := x + 3, \{x > 5\}) \Downarrow \{x > 8\}}{(\text{let } x = 5 \text{ in } (x := x + 3), \{x > 17\}) \Downarrow ?}$$



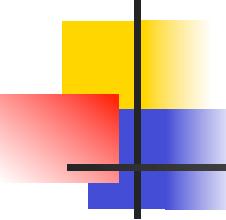
Example

$$\frac{\begin{array}{c} (x,\{x>5\}) \Downarrow 5 \quad (3,\{x>5\}) \Downarrow 3 \\ \hline (x+3,\{x>5\}) \Downarrow 8 \end{array}}{(5,\{x>17\}) \Downarrow 5 \quad (x:=x+3,\{x>5\}) \Downarrow \{x>8\}}$$
$$\frac{(5,\{x>17\}) \Downarrow 5 \quad (x:=x+3,\{x>5\}) \Downarrow \{x>8\}}{(\text{let } x = 5 \text{ in } (x:=x+3), \{x > 17\}) \Downarrow \{x>17\}}$$



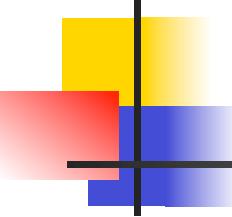
Comment

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



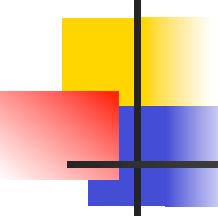
Interpretation Versus Compilation

- 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



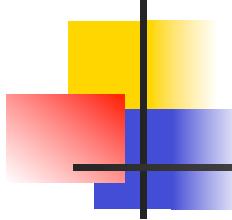
Interpreter

- 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



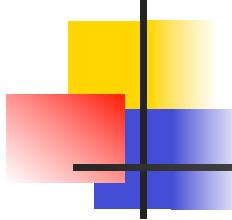
Interpreter

- 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

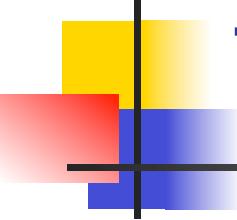
- $\text{compute_exp}(\text{Var}(v), m) = \text{look_up } v \text{ in } m$
- $\text{compute_exp}(\text{Int}(n), _) = \text{Num}(n)$
- ...
- $\text{compute_com}(\text{IfExp}(b, c_1, c_2), m) =$
 if $\text{compute_exp}(b, m) = \text{Bool}(\text{true})$
 then $\text{compute_com}(c_1, m)$
 else $\text{compute_com}(c_2, m)$



Natural Semantics Example

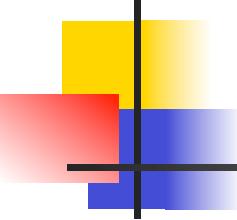
- $\text{compute_com}(\text{While}(b,c), m) =$
 if $\text{compute_exp}(b,m) = \text{Bool}(\text{false})$
 then m
 else $\text{compute_com}(\text{While}(b,c), \text{compute_com}(c,m))$

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



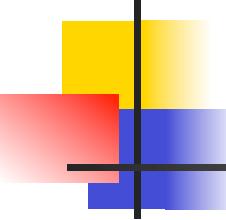
Transition Semantics

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



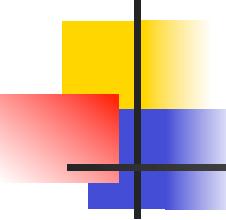
Expressions and Values

- 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



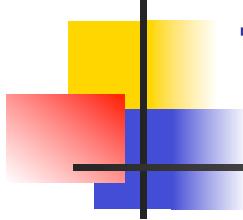
Evaluation Semantics

- 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



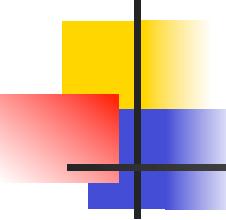
Simple Imperative Programming Language

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



Transitions for Expressions

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



Boolean Operations:

- Operators: (short-circuit)

$$(\text{false} \ \& \ B, m) \rightarrow (\text{false}, m) \quad (B, m) \rightarrow (B'', m)$$

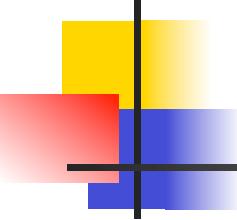
$$(\text{true} \ \& \ B, m) \rightarrow (B, m) \quad \overline{(B \ \& \ B', m) \rightarrow (B'' \ \& \ B', m)}$$

$$(\text{true or } B, m) \rightarrow (\text{true}, m) \quad (B, m) \rightarrow (B'', m)$$

$$(\text{false or } B, m) \rightarrow (B, m) \quad \overline{(B \text{ or } B', m) \rightarrow (B'' \text{ or } B', m)}$$

$$(\text{not true}, m) \rightarrow (\text{false}, m) \quad (B, m) \rightarrow (B', m)$$

$$(\text{not false}, m) \rightarrow (\text{true}, m) \quad \overline{(\text{not } B, m) \rightarrow (\text{not } B', m)}$$

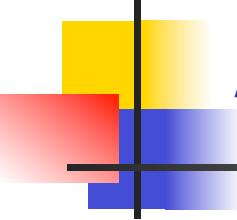


Relations

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

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

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

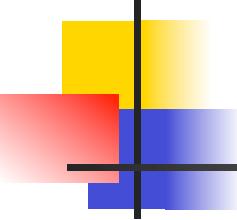


Arithmetic Expressions

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

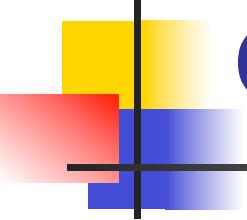
$$\frac{(E, m) \rightarrow (E', m)}{(V \text{ op } E, m) \rightarrow (V \text{ op } E', m)}$$

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



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



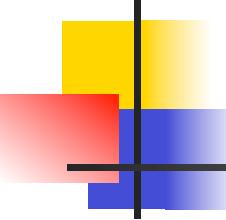
Commands

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

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

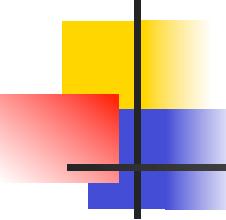
$$(I ::= V, m) \rightarrow m[I \leftarrow V]$$

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



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.

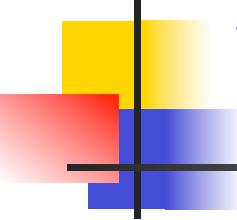


If Then Else Command

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

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

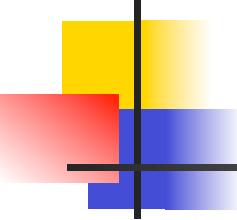
$$\frac{(B, m) \rightarrow (B', m)}{\begin{aligned} &(\text{if } B \text{ then } C \text{ else } C' \text{ fi, } m) \\ \rightarrow & (\text{if } B' \text{ then } C \text{ else } C' \text{ fi, } m) \end{aligned}}$$



While Command

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

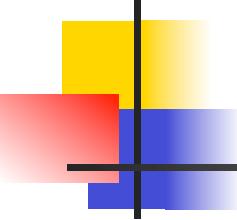
In English: Expand a While into a test of the boolean guard, with the true case being to do the body and then try the while loop again, and the false case being to stop.



Example Evaluation

- First step:

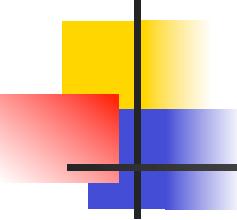
$$\begin{array}{c} (\text{if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi,} \\ \quad \{x \rightarrow 7\}) \\ \quad \rightarrow ? \end{array}$$



Example Evaluation

- First step:

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

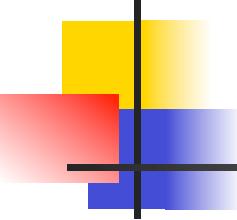


Example Evaluation

- First step:

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

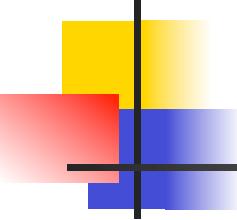
$$\begin{aligned} & (\text{if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi,} \\ & \quad \{x \rightarrow 7\}) \\ & \quad \rightarrow ? \end{aligned}$$



Example Evaluation

- First step:

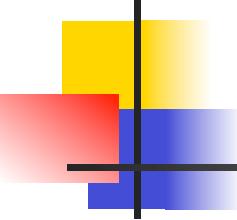
$$\frac{\frac{(x, \{x \rightarrow 7\}) \rightarrow (7, \{x \rightarrow 7\})}{(x > 5, \{x \rightarrow 7\}) \rightarrow (7 > 5, \{x \rightarrow 7\})}}{(\text{if } x > 5 \text{ then } y := 2 + 3 \text{ else } y := 3 + 4 \text{ fi}, \\ \{x \rightarrow 7\})} \rightarrow ?$$



Example Evaluation

- First step:

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



Example Evaluation

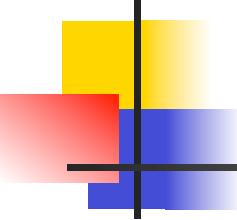
- Second Step:

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

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

- Third Step:

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



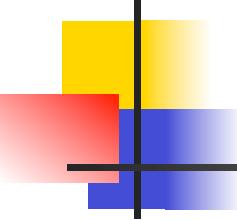
Example Evaluation

- 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:

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



Example Evaluation

- Bottom Line:

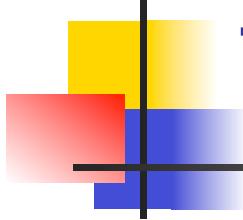
(if $x > 5$ then $y := 2 + 3$ else $y := 3 + 4$ fi,
 $\{x \rightarrow 7\}$)

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

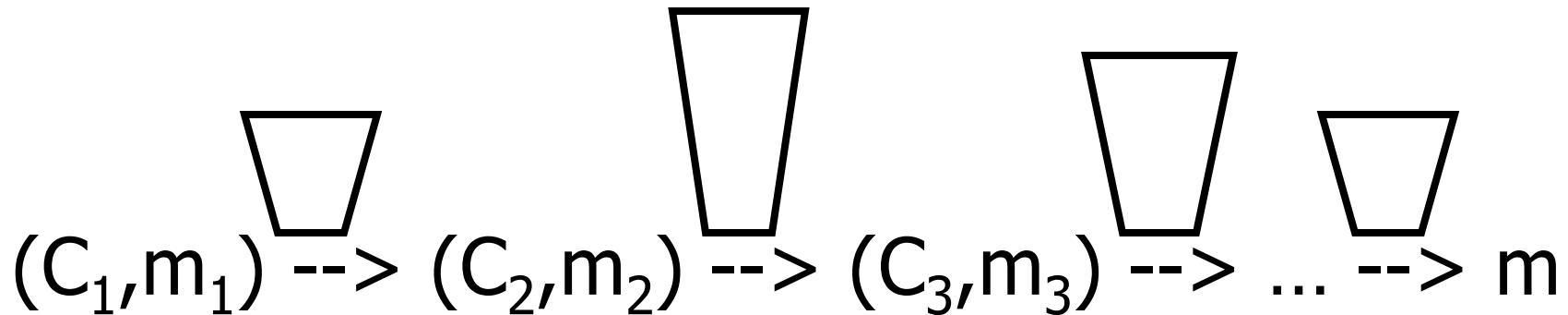
--> ($y := 2 + 3$, $\{x \rightarrow 7\}$)

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

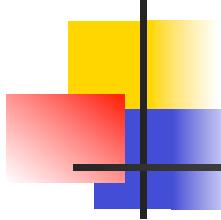


Transition Semantics Evaluation

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

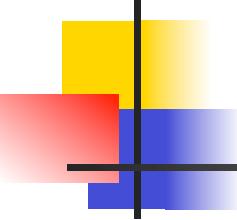


- Let \dashrightarrow^* be the transitive closure of \dashrightarrow
- Ie, the smallest transitive relation containing \dashrightarrow



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



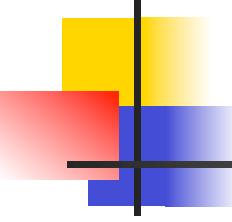
Call-by-value (Eager Evaluation)

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

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

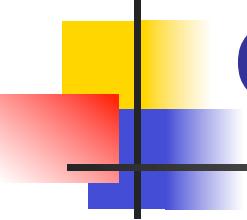
$$((\text{fun } I \rightarrow E) V, m) \rightarrow (E[V/I], m)$$

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



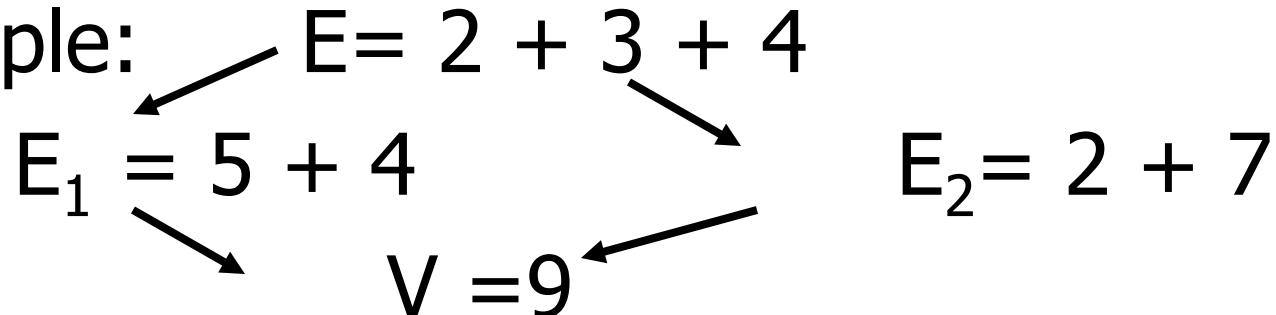
Call-by-name (Lazy Evaluation)

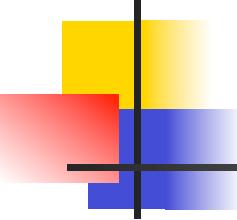
- $(\text{let } I = E \text{ in } E', m) \rightarrow (E' [E / I], m)$
- $((\text{fun } I \rightarrow E') E, m) \rightarrow (E' [E / I], m)$
- Question: Does it make a difference?
- It can depend on the language



Church-Rosser Property

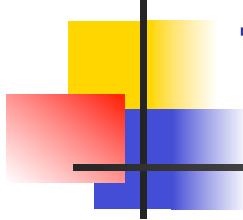
- Church-Rosser Property: If $E \rightarrow^* E_1$ and $E \rightarrow^* E_2$, if there exists a value V such that $E_1 \rightarrow^* V$, then $E_2 \rightarrow^* V$
- Also called **confluence** or **diamond property**
- Example:





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)



Transition Semantics for λ -Calculus

- Application (version 1)

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

- Application (version 2)

$$(\lambda x . E) V \rightarrow E[V/x]$$

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