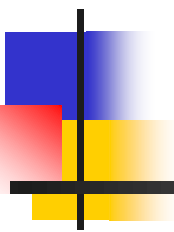


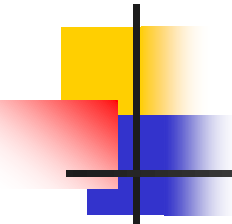
Programming Languages and Compilers (CS 421)



William Mansky

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

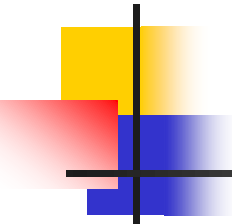
Based in part on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, Elsa Gunter, and Dennis Griffith



Natural Semantics

$$\frac{(E, m) \Downarrow v \quad (E', m) \Downarrow v' \quad v \text{ op } v' = n}{(E \text{ op } E', m) \Downarrow n}$$

where n is the specified value for $v \text{ op } v'$

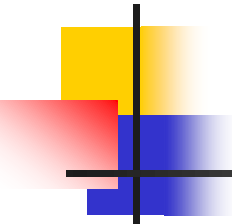


Natural Semantics

$$\frac{(E, m) \Downarrow v \quad (E', m) \Downarrow v' \quad v \text{ op } v' = n}{(E \text{ op } E', m) \Downarrow n}$$

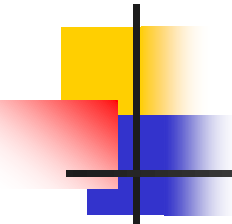
where n is the specified value for $v \text{ op } v'$

- Which comes first?
- What if evaluation fails?



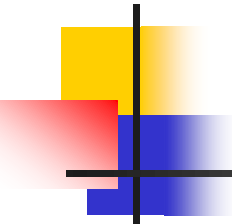
Structural Operational Semantics

- Also called “small-step” semantics
- Describes how each program construct transforms machine state by *transitions* or *steps*
- 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
- Indicates exactly one step of computation



Expressions and Values

- C, C' used for commands; E, E' for expressions; v, v' for values
- *Values* are a special class of expression we have to be able to recognize
 - E.g. 2, 3 are values, but $2 + 3$ is only an expression
- Memory only holds values
 - At least in the simplest model

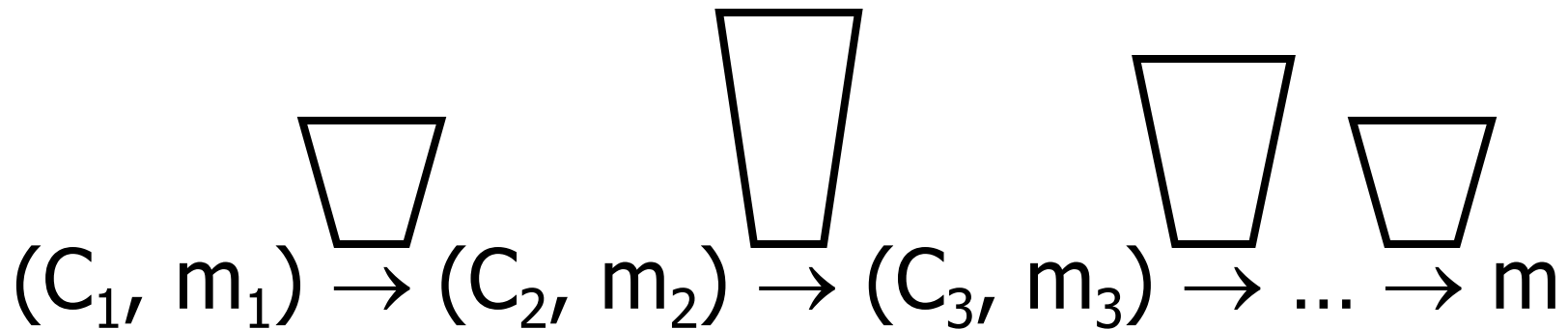


Evaluation Semantics

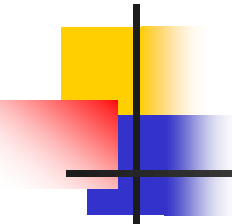
- Program successfully stops when left-hand side is a value/memory
- Evaluation fails if it has not succeeded and no step is possible
- Value/memory is the final *meaning* of original expression/command (in the given state)
- Coarse semantics: final value / memory
- More fine grained: whole sequence of steps

SOS Evaluation

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

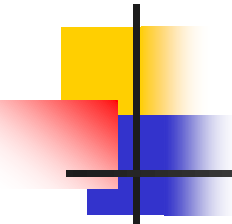


- We use $\rightarrow^*$ for the *transitive closure* of $\rightarrow$, the smallest transitive relation containing $\rightarrow$ (i.e., “some number of steps”)



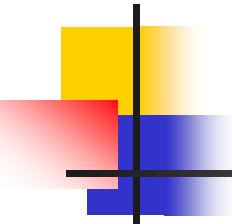
Simple Imperative Programming Language

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



Rules for Expressions

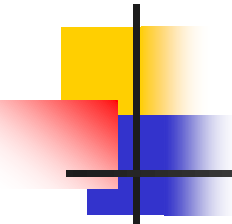
- Numerals are values
- true and false are values
- Identifiers: $(I, m) \rightarrow (m(I), m)$



Boolean Operations

■ Operators: (short-circuit)

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

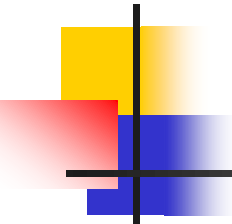


Relations

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

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

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

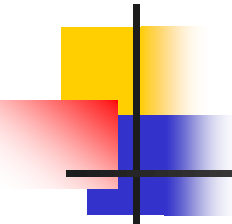


Arithmetic Expressions

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

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

$(v \text{ op } v', m) \rightarrow (N, m)$ where N is the value of $v \text{ op } v'$



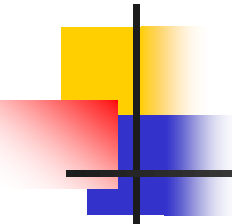
Commands

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

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

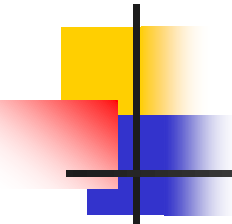
$$(I := v, m) \rightarrow m[I \leftarrow v]$$

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



Notes on Commands

- skip means done evaluating
- When evaluating an assignment, evaluate the expression first
- Once the expression being assigned is a value, update the memory with the new value
- When evaluating a sequence, work on the first command in the sequence first
- If the first command evaluates to a new memory (i.e. completes), evaluate remainder with new memory

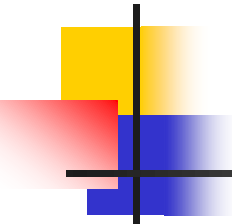


If Then Else Command

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

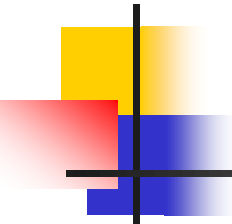
$(\text{if false then } C \text{ else } C' \text{ fi}, m) \rightarrow (C', m)$

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



Notes on If-Then-Else

- If the boolean guard is not a value, then start by evaluating it first
- If the boolean guard is true, then evaluate the first branch
- If it is false, evaluate the second branch

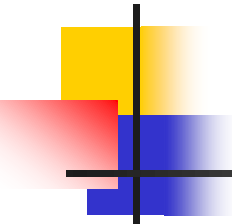


While Command

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

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

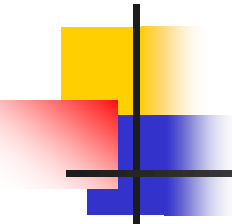
- Test the boolean guard, in the true case do the body and then try the while loop again, in the false case stop



Example Evaluation

- First step:

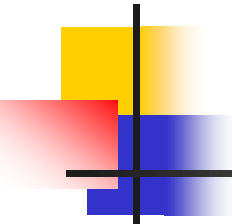
(if $x > 5$ then $y := 2 + 3$ else $y := 3 + 4$ fi,
 $\{x \rightarrow 7\}$)
 $\rightarrow ?$



Example Evaluation

- First step:

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



Example Evaluation

- First step:

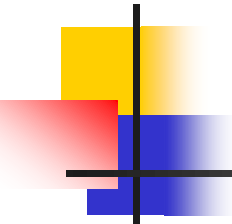
$$\frac{(x, \{x \rightarrow 7\}) \rightarrow (7, \{x \rightarrow 7\})}{(x > 5, \{x \rightarrow 7\}) \rightarrow ?}$$

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

Example Evaluation

- First step:

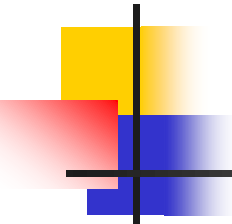
$$\frac{\frac{(x, \{x \rightarrow 7\}) \rightarrow (7, \{x \rightarrow 7\})}{(x > 5, \{x \rightarrow 7\}) \rightarrow (7 > 5, \{x \rightarrow 7\})}}{(if\ x > 5\ then\ y := 2 + 3\ else\ y := 3 + 4\ fi,\ \{x \rightarrow 7\}) \rightarrow ?}$$



Example Evaluation

- First step:

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



Example Evaluation

- Second Step:

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

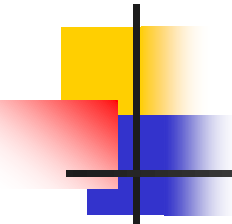
Example Evaluation

- Second Step:

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

- Third Step:

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



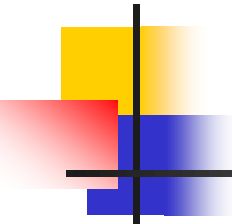
Example Evaluation

- 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:

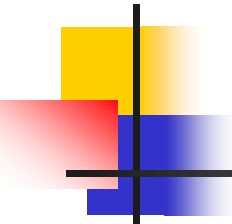
(if $x > 5$ then $y := 2 + 3$ else $y := 3 + 4$ fi,
 $\{x \rightarrow 7\}$)

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

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

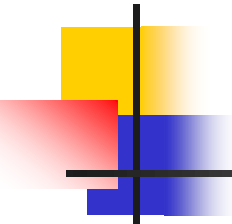
$\rightarrow (y := 2 + 3, \{x \rightarrow 7\})$

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



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 memory, but so far evaluating expressions doesn't alter the environment
- We will use substitution here instead
- **Notation:** $E[E'/I]$ means replace all free occurrences of I by E' in E



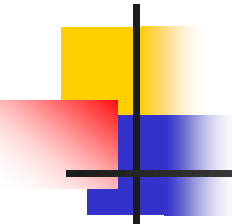
Call-by-value (Eager Evaluation)

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

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

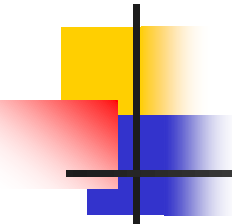
$$((\text{fun } I \rightarrow E) v, m) \rightarrow (E[v/I], m)$$

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



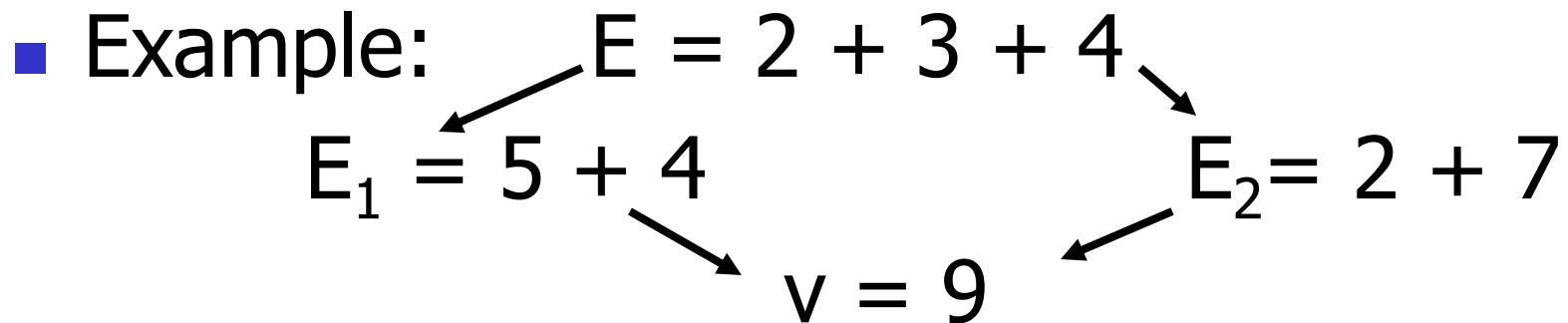
Call-by-name (Lazy Evaluation)

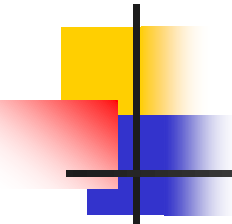
- $(\text{let } I = E \text{ in } E', m) \rightarrow (E'[E / I], m)$
- $((\text{fun } I \rightarrow E') E, m) \rightarrow (E'[E / I], m)$
- Does it make a difference? It can, depending on the language
- More on this later



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**





Does It always Hold?

- No. Languages with side-effects tend not be Church-Rosser
- Alonzo Church and Barkley Rosser proved in 1936 that the λ -calculus does have it
- Benefit of Church-Rosser: can check equality of terms by evaluating them (although evaluation might not terminate)