

IS593: Language-based Security

2. Operational Semantics

Kihong Heo



Language = Syntax + Semantics

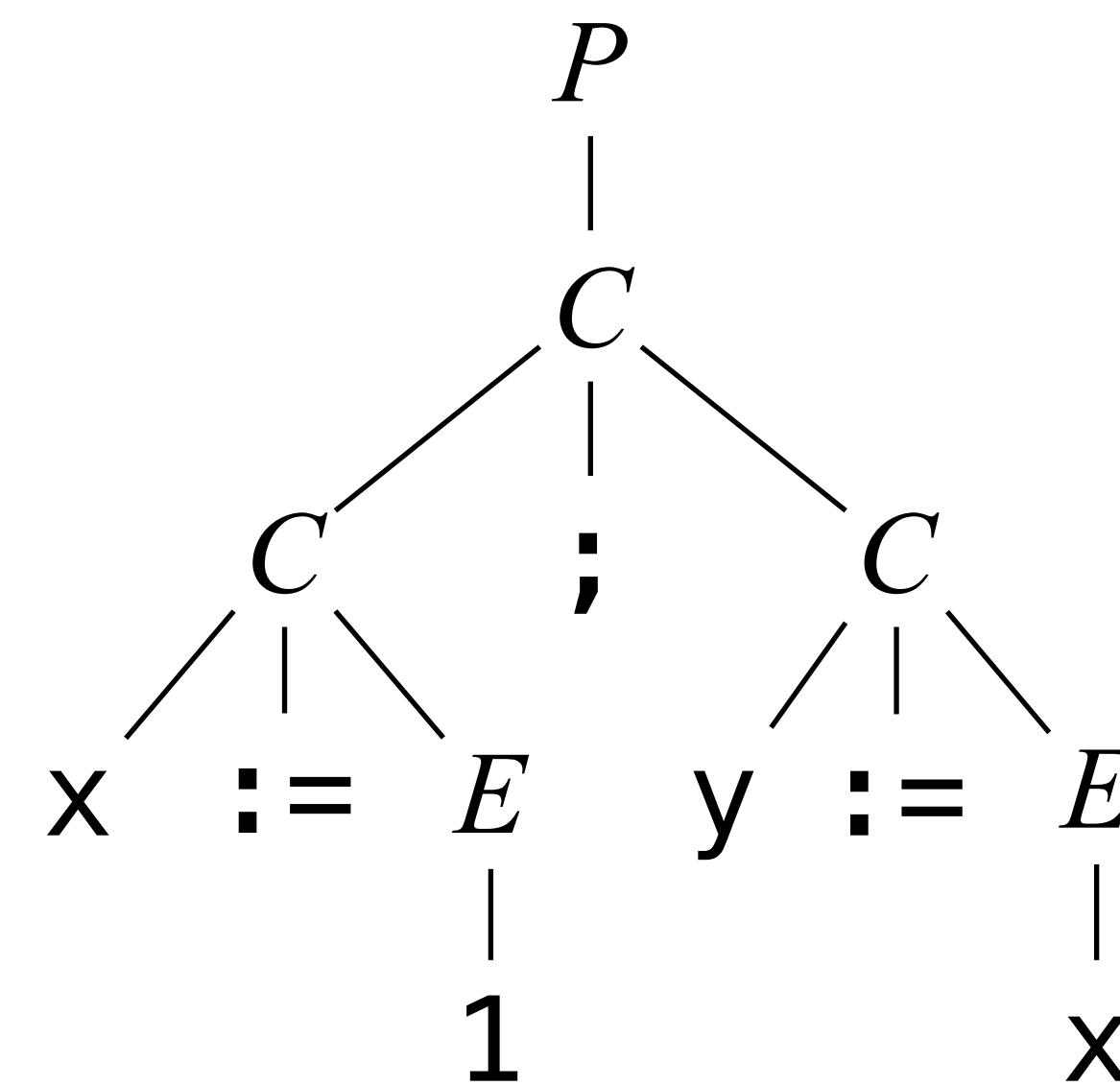
- How to design a programming language?
- How to execute/compile/analyze a program written in a certain language?



Syntax

- Syntax : concern with the **grammatical structure** of programs
- Syntactic analysis = Parsing
 - constructing an *abstract syntax tree*
 - w.r.t the grammar

$x := 1; y := x$



A Simple Imperative Language

$E ::=$		arithmetic expressions
	n	integer constants
	x	variable
	$E \odot E$	binary operation
$B ::=$		boolean expression
	$\text{true} \mid \text{false}$	boolean constants
	$E \oslash E$	comparison expressions
$C ::=$		commands
	skip	command that does nothing
	$C; C$	sequence
	$x := E$	assignment
	$\text{input}(x)$	command reading of a value
	$\text{if } B \text{ then } C \text{ else } C$	conditional command
	$\text{while } B \text{ } C$	loop command

You Think You Know C?

```
struct S {  
    int i;  
    char c;  
} s;  
  
int main(void) {  
    return sizeof(s);  
}
```

- A. 4
- B. 5
- C. 8
- D. I don't know.

```
int main(void) {  
    char a = 0;  
    short int b = 0;  
    return sizeof(a) + sizeof(b);  
}
```

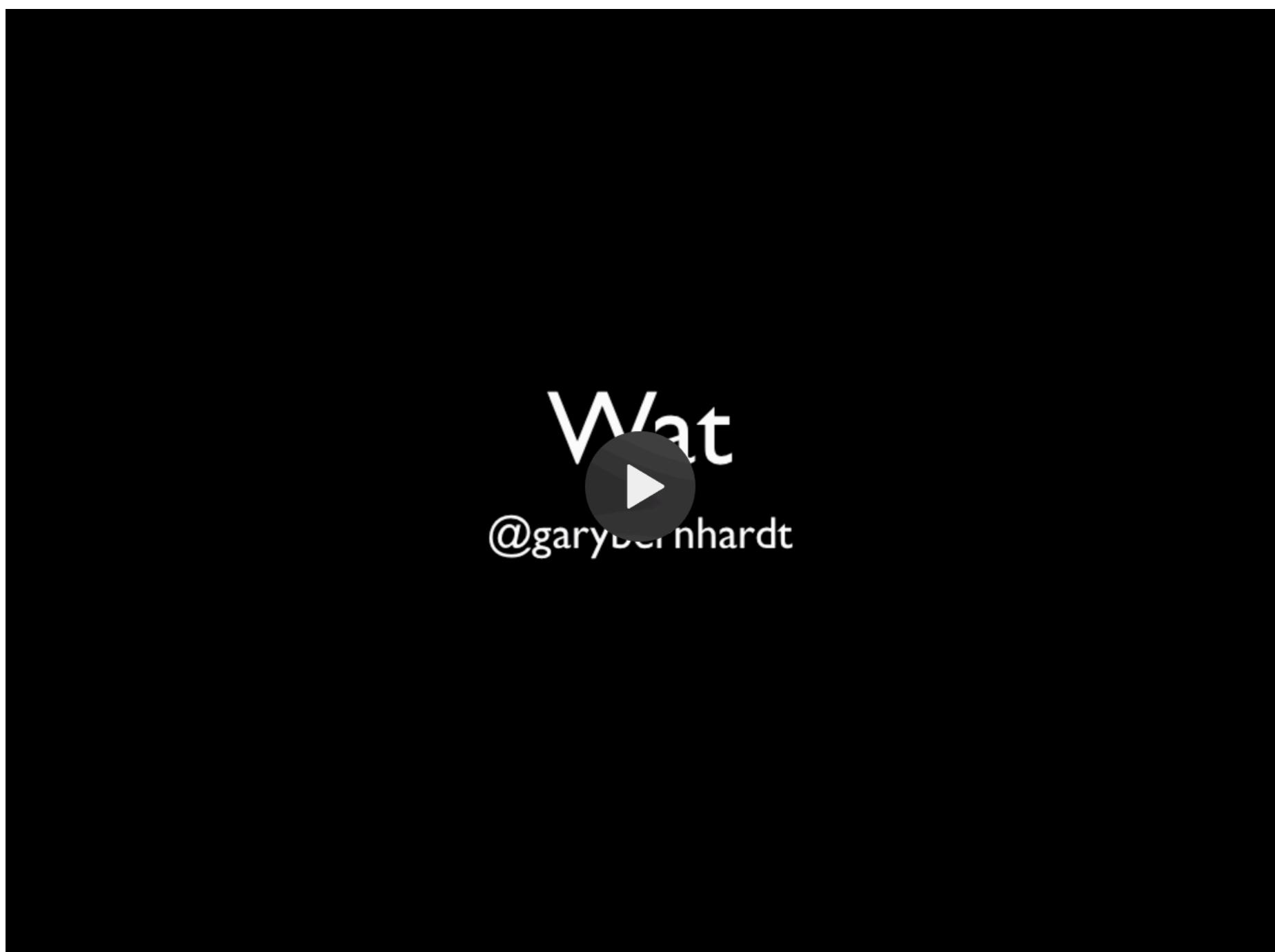
- A. 0
- B. 1
- C. 2
- D. I don't know.

```
int main(void) {  
    int i = 0;  
    return i++ + ++i;  
}
```

- A. 1
- B. 2
- C. 3
- D. I don't know.

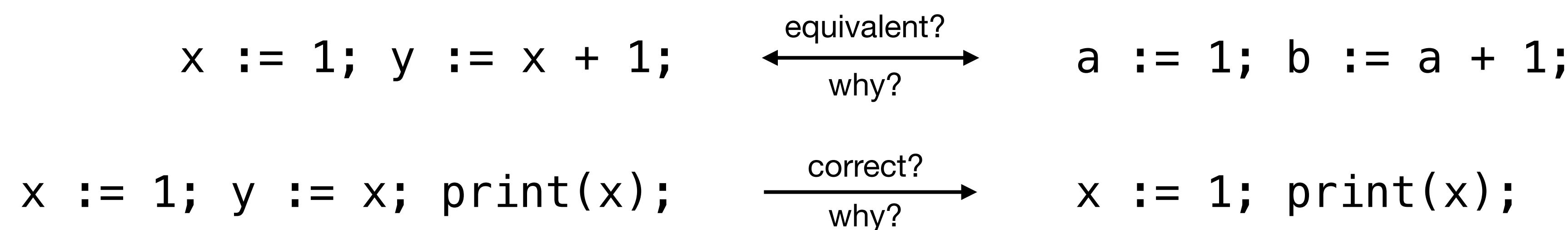
*<https://wordsandbuttons.online/so you think you know c.html>

Wat?



Semantics

- Semantics: concern with the **meaning** of grammatically correct programs
 - interpreters execute programs w.r.t. the **semantics**
 - compilers translate programs w.r.t. the **semantics**
 - analyzers analyze programs w.r.t. the **semantics**



Different Styles of Semantics

- **Operational Semantics:** “How to compute the execution result”
 - so-called transitional style
 - $3 * (2 + 1) : 3 * (2 + 1) \rightarrow 3 * 3 \rightarrow 9$
- **Denotational Semantics:** “What the execution result is”
 - so-called compositional style
 - $3 * (2 + 1) : 9$
- ...

Different approaches for different purposes and languages

Operational Semantics

- Program semantics is a series of machine state transitions
 - Big-step : how the **overall results** of executions are obtained
 - Small-step : how the **individual steps** of the computations take place
- The semantics of a program is determined by the semantics of its subcomponents or that of itself (i.e., **inductive**)

Big-step

$$\begin{array}{c} \dfrac{}{2 \times 2 \Rightarrow 4} \quad \dfrac{}{4 \times 2 \Rightarrow 8} \\ \hline \dfrac{2 \times 2 \Rightarrow 4}{2 \times 2 \times 2 \Rightarrow 8} \quad \dfrac{4 \times 2 \Rightarrow 8}{2 + 1 \Rightarrow 3} \quad \dfrac{}{8 \times 3 \Rightarrow 24} \\ \hline (2 \times 2 \times 2) \times (2 + 1) \Rightarrow 24 \end{array}$$

why?

Small-step

$$\begin{aligned} & (2 \times 2 \times 2) \times (2 + 1) \\ & \rightarrow (4 \times 2) \times (2 + 1) \\ & \rightarrow 8 \times (2 + 1) \\ & \rightarrow 8 \times 3 \\ & \rightarrow 24 \end{aligned}$$

so?

Big-step Semantics

- What does this notation mean?



$$\frac{\frac{2 \times 2 \Rightarrow 4}{2 \times 2 \times 2 \Rightarrow 8} \quad \frac{4 \times 2 \Rightarrow 8}{2 + 1 \Rightarrow 3}}{(2 \times 2 \times 2) \times (2 + 1) \Rightarrow 24} \quad \frac{8 \times 3 \Rightarrow 24}{}$$

Inference Rules

- Methods to **derive new facts from known facts** (i.e., derivable = provable)
- Notation of inference rule:

$$\frac{P_1 \dots P_n}{P} \quad \begin{matrix} \text{premise(s)} \\ \text{conclusion} \end{matrix}$$

$$\frac{\textit{human}(x) \implies \textit{mortal}(x) \quad \textit{human}(s)}{\textit{mortal}(s)}$$

- Inference rule with zero premise is called an **axiom**
- Examples: inference rules of propositional logic

$$\frac{}{\text{true}} \quad \frac{A \quad B}{A \wedge B} \quad \frac{A \wedge B}{A} \quad \frac{A \wedge B}{B} \quad \frac{A}{A \vee B} \quad \frac{B}{A \vee B} \quad \frac{A \quad A \implies B}{B} \quad \dots$$

Semantics as Proofs

- Program semantics: **proofs** using a set of **inference rules**

$$\langle C, m \rangle \Rightarrow m'$$

“Execution of C from m will result in m’.”

$$\langle E, m \rangle \Rightarrow n$$

“Execution of E from m will result in n.”

$$\langle B, m \rangle \Rightarrow b$$

“Execution of B from m will result in b.”

- Define the meaning of syntactic objects C, E, B using semantic objects m, m', n, b
- Proof** of how the **overall results** of executions are obtained

$$\frac{\frac{2 \times 2 \Rightarrow 4 \quad 4 \times 2 \Rightarrow 8}{2 \times 2 \times 2 \Rightarrow 8} \quad \frac{2 + 1 \Rightarrow 3}{}}{(2 \times 2 \times 2) \times (2 + 1) \Rightarrow 24} \quad \frac{}{8 \times 3 \Rightarrow 24}$$



Semantic Domains

- Sets of semantic objects
- Memory is a mapping $\mathbb{M} = \mathbb{X} \rightarrow \mathbb{V}$
 - \mathbb{X} : the set of variables
 - \mathbb{V} : the set of integers (\mathbb{Z}) and booleans (\mathbb{B})
- Example: $\langle x := 7; y := 3, \{\} \rangle \Rightarrow \{x \mapsto 7, y \mapsto 3\}$

$\langle C, m \rangle \Rightarrow m'$
"Execution of C from m will result in m'."

Variable	Value
x	7
y	3

Big-step Operational Semantics

$$\begin{array}{c}
 \boxed{\langle E, m \rangle \Rightarrow n} \\
 \dfrac{}{\langle n, m \rangle \Rightarrow n} \\
 \dfrac{m(x) = n}{\langle x, m \rangle \Rightarrow n} \\
 \hline
 \dfrac{\langle E_1, m \rangle \Rightarrow n_1 \quad \langle E_2, m \rangle \Rightarrow n_2 \quad n = n_1 \odot n_2}{\langle E_1 \odot E_2, m \rangle \Rightarrow n}
 \end{array}$$

$$\begin{array}{c}
 \boxed{\langle C, m \rangle \Rightarrow m'} \\
 \dfrac{}{\langle \text{skip}, m \rangle \Rightarrow m} \\
 \dfrac{\langle C_1, m \rangle \Rightarrow m_1 \quad \langle C_2, m_1 \rangle \Rightarrow m_2}{\langle C_1; C_2, m \rangle \Rightarrow \langle C'_1; C_2, m_2 \rangle} \\
 \dfrac{\langle E, m \rangle \Rightarrow n}{\langle x := E, m \rangle \Rightarrow m\{x \mapsto n\}} \\
 \dfrac{}{\langle \text{input}(x), m \rangle \Rightarrow m\{x \mapsto n\}}
 \end{array}$$

$$\begin{array}{c}
 \boxed{\langle B, m \rangle \Rightarrow b} \\
 \dfrac{}{\langle \text{true}, m \rangle \Rightarrow \text{true}} \\
 \dfrac{}{\langle \text{false}, m \rangle \Rightarrow \text{false}} \\
 \hline
 \dfrac{\langle E_1, m \rangle \Rightarrow n_1 \quad \langle E_2, m \rangle \Rightarrow n_2 \quad b = n_1 \oslash n_2}{\langle E_1 \oslash E_2, m \rangle \Rightarrow b}
 \end{array}
 \quad
 \begin{array}{c}
 \dfrac{\langle B, m \rangle \Rightarrow \text{true} \quad \langle C_1, m \rangle \Rightarrow m_1}{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, m \rangle \Rightarrow m_1} \\
 \dfrac{\langle B, m \rangle \Rightarrow \text{false} \quad \langle C_2, m \rangle \Rightarrow m_2}{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, m \rangle \Rightarrow m_2} \\
 \dfrac{\langle B, m \rangle \Rightarrow \text{true} \quad \langle C, m \rangle \Rightarrow m_1 \quad \langle \text{while } B \text{ } C, m_1 \rangle \Rightarrow m_2}{\langle \text{while } B \text{ } C, m \rangle \Rightarrow m_2} \\
 \dfrac{\langle B, m \rangle \Rightarrow \text{false}}{\langle \text{while } B \text{ } C, m \rangle \Rightarrow m}
 \end{array}$$

Exercise

- $x := 1; y := x + 1$

Exercise

- $x := 1; \text{ if } x > 0 \text{ then } y := 1 \text{ else } y := -1$

Exercise

- $x := 0; \text{while } (x < 10) \ (x := x + 1)$

Operational Semantics (Cont'd)

- Program semantics is a series of machine state transitions
 - Big-step : how the **overall results** of executions are obtained
 - Small-step : how the **individual steps** of the computations take place
- The semantics of a program is determined by the semantics of its subcomponents or that of itself (i.e., **inductive**)

Big-step

$$\begin{array}{c} \dfrac{2 \times 2 \Rightarrow 4}{2 \times 2 \times 2 \Rightarrow 8} \quad \dfrac{4 \times 2 \Rightarrow 8}{2 + 1 \Rightarrow 3} \\ \hline (2 \times 2 \times 2) \times (2 + 1) \Rightarrow 24 \end{array}$$

why?

Small-step

$$\begin{aligned} & (2 \times 2 \times 2) \times (2 + 1) \\ & \rightarrow (4 \times 2) \times (2 + 1) \\ & \rightarrow 8 \times (2 + 1) \\ & \rightarrow 8 \times 3 \\ & \rightarrow 24 \end{aligned}$$

so?

Semantics as State Transitions

- The semantics is specified by a transition system
 - \mathbb{S} : the set of states
 - $(\rightarrow) \subseteq \mathbb{S} \times \mathbb{S}$: the transition relation that describes computation steps
- **Footprint** of how the **individual steps** of the computations take place

$$\begin{aligned}(2 \times 2 \times 2) \times (2 + 1) \\ \rightarrow (4 \times 2) \times (2 + 1) \\ \rightarrow 8 \times (2 + 1) \\ \rightarrow 8 \times 3 \\ \rightarrow 24\end{aligned}$$


so?

States, Transitions, and Semantic Functions

- A state is a pair of command and memory $\langle C, m \rangle$
- A transition transfers a given state to the next one

$$\langle C, m \rangle \rightarrow \langle C', m' \rangle$$

*“Execution of C from m
will result in C' and m'.”*

- Semantics of expressions is defined as function

$$[E] : \mathbb{M} \rightarrow \mathbb{Z}$$

*Arithmetic expression E
as a function*

$$[B] : \mathbb{M} \rightarrow \mathbb{B}$$

*Boolean expression B
as a function*

Semantics of Expressions

$$\llbracket E \rrbracket : \mathbb{M} \rightarrow \mathbb{Z}$$

$$\llbracket n \rrbracket(m) = n$$

$$\llbracket x \rrbracket(m) = m(x)$$

$$\llbracket E_1 \odot E_2 \rrbracket(m) = \llbracket E_1 \rrbracket(m) \odot \llbracket E_2 \rrbracket(m)$$

$$\frac{\langle C_1, m \rangle \rightarrow \langle C'_1, m' \rangle}{\langle C_1; C_2, m \rangle \rightarrow \langle C'_1; C_2, m' \rangle}$$

$$\frac{}{\langle \text{skip}; C_2, m \rangle \rightarrow \langle C_2, m \rangle}$$

$$\frac{\llbracket E \rrbracket(m) = n}{\langle x := E, m \rangle \rightarrow \langle \text{skip}, m\{x \mapsto n\} \rangle}$$

$$\frac{}{\langle \text{input}(x), m \rangle \rightarrow \langle \text{skip}, m\{x \mapsto n\} \rangle}$$

$$\llbracket B \rrbracket : \mathbb{M} \rightarrow \mathbb{B}$$

$$\llbracket \text{true} \rrbracket(m) = \text{true}$$

$$\llbracket \text{false} \rrbracket(m) = \text{false}$$

$$\llbracket E_1 \oslash E_2 \rrbracket(m) = \llbracket E_1 \rrbracket(m) \oslash \llbracket E_2 \rrbracket(m)$$

$$\frac{\llbracket B \rrbracket(m) = \text{true}}{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, m \rangle \rightarrow \langle C_1, m \rangle}$$

$$\frac{\llbracket B \rrbracket(m) = \text{false}}{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, m \rangle \rightarrow \langle C_2, m \rangle}$$

$$\frac{}{\langle \text{while } B \text{ } C, m \rangle \rightarrow \langle \text{if } B \text{ then } (\text{C } \text{ while } B \text{ } C \text{ else skip}, m) \rangle}$$

Exercise

- $x := 1; y := x + 1$

Exercise

- $x := 1; \text{ if } x > 0 \text{ then } y := 1 \text{ else } y := -1$

Exercise

- $x := 0; \text{while } (x < 10) (x := x + 1)$

Have Learned So Far

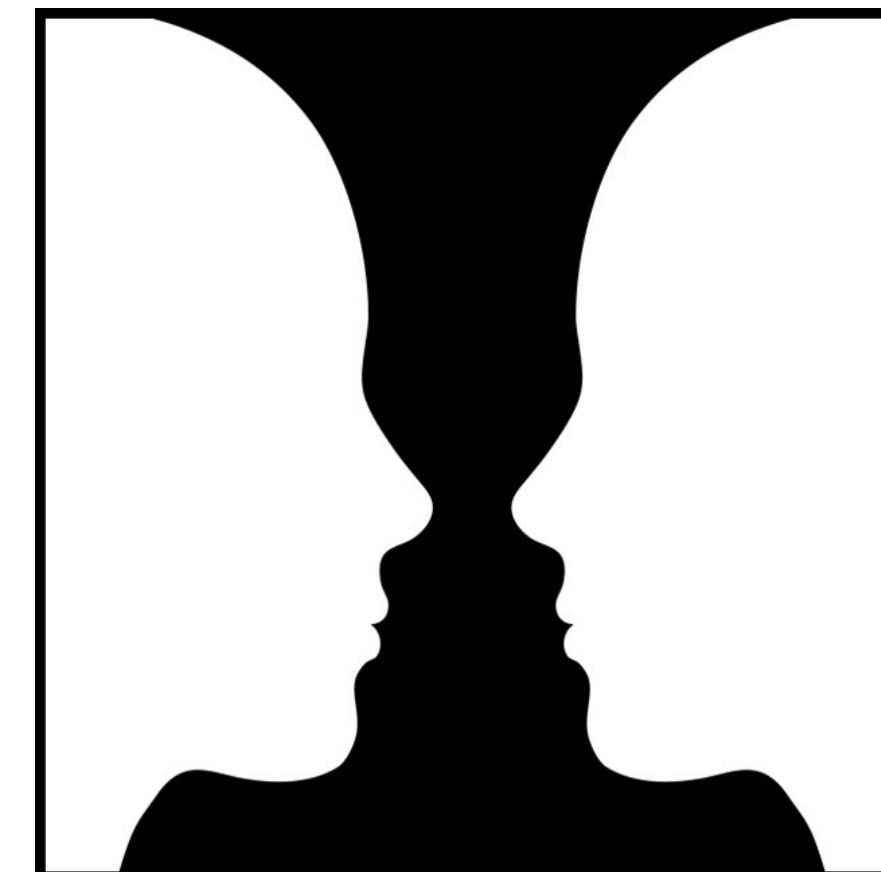
- Formal ways to describe programming language semantics
 - operational: inductive definition of machine state transitions
- Principles of programming systems (interpreters, compilers, analyzers, etc)

$$\begin{array}{c} \text{EVAL-CONST} \quad \text{EVAL-CONST-ERROR} \\ \frac{}{\varrho \Vdash a \Rightarrow v} \qquad \frac{}{\varrho \Vdash a \Rightarrow \text{error}} \\ \text{EVAL-PRIM} \quad \text{EVAL-PRIM-ERROR} \\ \frac{\varrho \Vdash a \Rightarrow v \quad f^1 v \rightarrow v'}{\varrho \Vdash f^1 a \Rightarrow v'} \qquad \frac{\varrho \Vdash a \Rightarrow v \quad f^1 v \rightarrow v'}{\varrho \Vdash f^1 a \Rightarrow \text{error}} \\ \text{EVAL-VAR} \quad \text{EVAL-FUN} \\ \frac{z \in \text{dom } (\varrho)}{\varrho \Vdash z \Rightarrow \varrho(z)} \qquad \frac{e \vdash \lambda x. a \Rightarrow \langle \lambda x. a, \varrho \rangle}{\varrho \Vdash e \Rightarrow \langle \lambda x. a, \varrho \rangle} \\ \text{EVAL-APP} \\ \frac{\varrho \vdash a \Rightarrow \langle \lambda x. a_0, \varrho_0 \rangle \quad \varrho \vdash a' \Rightarrow v \quad \varrho_0, x \mapsto v \vdash a_0 : v'}{\varrho \vdash a' \Rightarrow v'} \\ \text{EVAL-APP-ERROR} \quad \text{EVAL-APP-ERROR-LEFT} \\ \frac{\varrho \vdash a \Rightarrow C_1 v_1}{\varrho \vdash a' \Rightarrow \text{error}} \qquad \frac{\varrho \vdash a \Rightarrow \text{error}}{\varrho \vdash a' \Rightarrow \text{error}} \\ \text{EVAL-APP-ERROR-RIGHT} \\ \frac{\varrho \vdash a \Rightarrow \langle \lambda x. a_0, \varrho_0 \rangle \quad \varrho \vdash a' \Rightarrow \text{error}}{\varrho \vdash a' \Rightarrow \text{error}} \\ \text{EVAL-LET} \\ \frac{\varrho \vdash a \Rightarrow v \quad \varrho, x \mapsto v \vdash a' \Rightarrow v'}{\varrho \vdash \text{let } x = a \text{ in } a' \Rightarrow v'} \\ \text{EVAL-LET-ERROR} \\ \frac{}{\varrho \vdash a \Rightarrow \text{error}} \\ \frac{}{\varrho \vdash \text{let } x = a \text{ in } a' \Rightarrow \text{error}} \end{array}$$

Operational semantics of Core ML (the core of OCaml)
<https://caml.inria.fr/pub/docs/u3-ocaml/ocaml-ml.html>

Dark-side of Programs

- So far, we have discussed programs whose semantics is formally defined
- What happens if the semantics is not formally defined?



Unspecified / Undefined Behavior

- **Permissive**: multiple implementations are permitted (e.g., evaluation order)
- **Erroneous**: potentially unsafe program execution
(e.g., integer overflow, null-dereference, buffer-overrun, etc)
- For more details, see <https://en.cppreference.com/w/c/language/behavior>

In the Wild

- What about lower-level (less structured) programming languages?
 - jmp with/without if and while
 - e.g., C, LLVM, x86, etc
- Small-step operation semantics can be a better choice
 - similar to machine's behavior

```
x = 0;  
while (x < 10) {  
    x++;  
}
```

vs

```
x = 0;  
l1: br (x < 10) l2 l3;  
l2: x = x + 1;  
    goto l1;  
l3:
```

The SmaLLVM Language



- A program is a tuple $\langle \mathbb{L}, \text{next} \rangle$ where
 - \mathbb{L} is a set of program labels
 - next control-flow relation
- Each program label l is associated with a command denoted by $\text{cmd}(l)$

L	$::=$	l	program label
E	$::=$	n	integer
		true false	boolean
		x	variable
		$E \oplus E$	arithmetic operation
		$E \otimes E$	comparison operation
\oplus	$::=$	$+ - \times /$	
\otimes	$::=$	$< \leq > \geq == !=$	
C	$::=$	$x := E$	assignment
		br $E L L$	conditional jump
		goto L	unconditional jump
		$x := \text{input}()$	input
		print(x)	print

Semantic Domains

- The semantics is specified by a transition system
 - \mathbb{S} : the set of states
 - $(\rightarrow) \subseteq \mathbb{S} \times \mathbb{S}$: the transition relation that describes computation steps

$$\begin{array}{llll}\langle l, m \rangle & \in & \mathbb{S} & = \mathbb{L} \times \mathbb{M} \\ l & \in & \mathbb{L} & \text{program label} \\ m & \in & \mathbb{M} & = \mathbb{X} \rightarrow \mathbb{V} \\ x & \in & \mathbb{X} & \text{variable} \\ & & \mathbb{V} & = \mathbb{Z} + \mathbb{B} \\ n & \in & \mathbb{Z} & \text{integer} \\ b & \in & \mathbb{B} & \text{boolean}\end{array}$$

Small-step Operation Semantics

$$\begin{array}{l} \llbracket E \rrbracket : \mathbb{M} \rightarrow \mathbb{V} \\ \llbracket n \rrbracket(m) = n \\ \llbracket \text{true} \rrbracket(m) = \text{true} \\ \llbracket \text{false} \rrbracket(m) = \text{false} \\ \llbracket x \rrbracket(m) = m(x) \\ \llbracket E_1 \oplus E_2 \rrbracket(m) = \llbracket E_1 \rrbracket(m) \oplus \llbracket E_2 \rrbracket(m) \\ \llbracket E_1 \oslash E_2 \rrbracket(m) = \llbracket E_1 \rrbracket(m) \oslash \llbracket E_2 \rrbracket(m) \end{array}$$

$$\frac{\text{cmd}(l) = "x := E" \quad n = \llbracket E \rrbracket(m) \quad l' = \text{next}(l)}{\langle l, m \rangle \hookrightarrow \langle l', m\{x \mapsto n\} \rangle}$$

$$\frac{\text{cmd}(l) = "\text{br } E \ l_1 \ l_2" \quad \llbracket E \rrbracket(m) = \text{true}}{\langle l, m \rangle \hookrightarrow \langle l_1, m \rangle}$$

$$\frac{\text{cmd}(l) = "\text{br } E \ l_1 \ l_2" \quad \llbracket E \rrbracket(m) = \text{false}}{\langle l, m \rangle \hookrightarrow \langle l_2, m \rangle}$$

$$\frac{\text{cmd}(l) = "\text{goto } l'"}{\langle l, m \rangle \hookrightarrow \langle l', m \rangle}$$

$$\frac{\text{cmd}(l) = "x := \text{input}()" \quad l' = \text{next}(l)}{\langle l, m \rangle \hookrightarrow \langle l', m\{x \mapsto n\} \rangle}$$

$$\frac{\text{cmd}(l) = "\text{print}(x)" \quad l' = \text{next}(l)}{\langle l, m \rangle \hookrightarrow \langle l', m \rangle}$$

Summary

- Operational semantics tells how to execute programs
 - Big-step concerns how the **overall results** are obtained
 - Small-step concerns how the **individual steps** take place
- Operational semantics is an inductive semantic description method
 - Big-step: semantics as **inference rules**
 - Small-step: semantics as **state transitions**