Introduction to Program Analysis

2. Operational Semantics

Kihong Heo

KAIST

Language = Syntax + Semantics

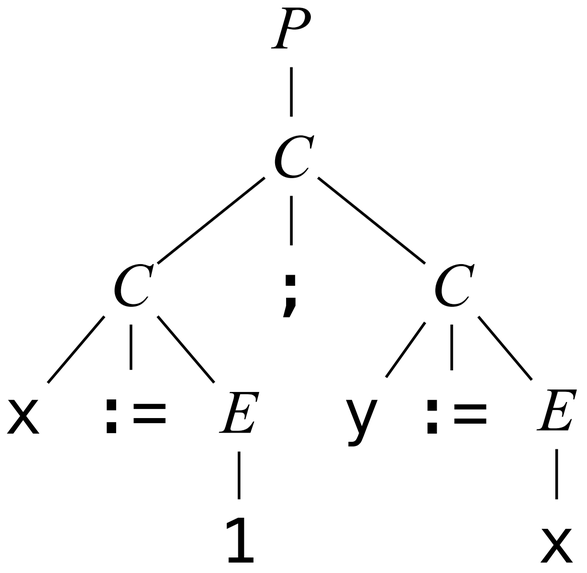
- How to design a programming language?
- How to execute/compile/analyze a program written in a certain language?

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
```

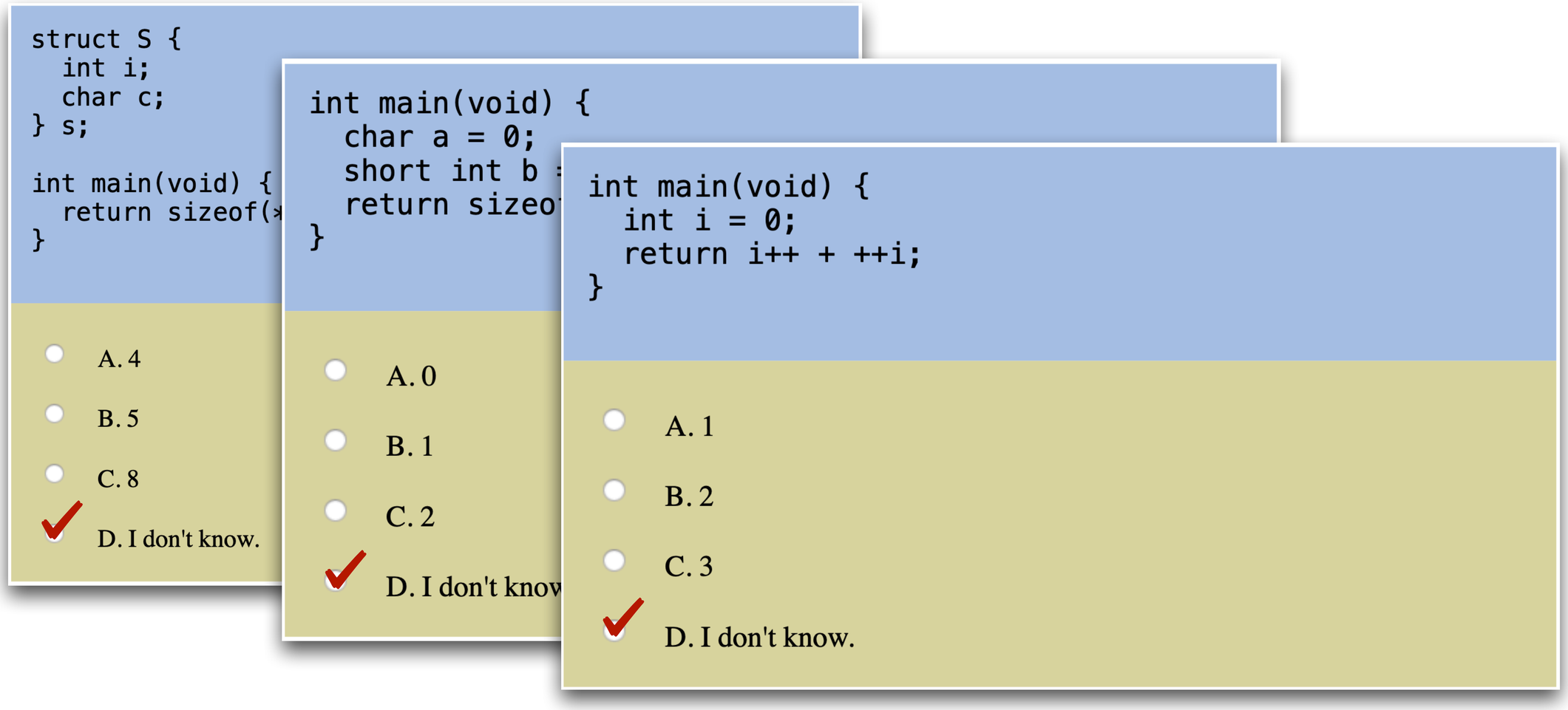
parsing



A Simple Imperative Language

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

You Think You Know C?



*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**

```
x := 1; y := x + 1;
```
equivalent? / why?
```
a := 1; b := a + 1;
```

```
x := 1; y := x; print(x);
```
correct? / why?
```
x := 1; print(x);
```

Different Styles of Semantics

- **Operational Semantics**: "**How** to compute the execution result"
  - so-called transitional style
  - `3 * (2 + 1)` : **3 * (2 + 1) → 3 * 3 → 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

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

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?

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

Inference Rules

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

$$\frac{P_1 \quad \cdots \quad P_n}{P}$$ *premise(s)* / *conclusion*

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

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

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

Semantics as Proofs

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

$$\langle C, m \rangle \Rightarrow m' \qquad \langle E, m \rangle \Rightarrow n \qquad \langle B, m \rangle \Rightarrow b$$

"Execution of C from m will result in m'."

"Execution of E from m will result in n."

"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

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

why?

Semantic Domains

- Sets of semantic objects
- Memory is a mapping $\mathbb{M} = \mathbb{X} \to \mathbb{V}$
  - $\mathbb{X}$: the set of variables
  - $\mathbb{V}$: the set of integers ($\mathbb{Z}$) and booleans ($\mathbb{B}$)
- Example:

$$\langle \mathtt{x} := 7; \mathtt{y} := 3, \{\} \rangle \Rightarrow \{\mathtt{x} \mapsto 7, \mathtt{y} \mapsto 3\}$$

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

"Execution of C from m will result in m'."

Variable	Value
x	7
y	3

Big-step Operational Semantics

$\boxed{\langle E, m\rangle \Rightarrow n}$

$$\frac{}{\langle n, m\rangle \Rightarrow n}$$

$$\frac{m(x) = n}{\langle x, m\rangle \Rightarrow n}$$

$$\frac{\langle E_1, m\rangle \Rightarrow n_1 \qquad \langle E_2, m\rangle \Rightarrow n_2 \qquad n = n_1 \odot n_2}{\langle E_1 \odot E_2, m\rangle \Rightarrow n}$$

$\boxed{\langle B, m\rangle \Rightarrow b}$

$$\frac{}{\langle \mathsf{true}, m\rangle \Rightarrow \mathsf{true}}$$

$$\frac{}{\langle \mathsf{false}, m\rangle \Rightarrow \mathsf{false}}$$

$$\frac{\langle E_1, m\rangle \Rightarrow n_1 \qquad \langle E_2, m\rangle \Rightarrow n_2 \qquad b = n_1 \oslash n_2}{\langle E_1 \oslash E_2, m\rangle \Rightarrow b}$$

$\boxed{\langle C, m\rangle \Rightarrow m'}$

$$\frac{}{\langle \texttt{skip}, m\rangle \Rightarrow m}$$

$$\frac{\langle C_1, m\rangle \Rightarrow m_1 \qquad \langle C_2, m_1\rangle \Rightarrow m_2}{\langle C_1;C_2, m\rangle \Rightarrow m_2}$$

$$\frac{\langle E, m\rangle \Rightarrow n}{\langle x := E, m\rangle \Rightarrow m\{x \mapsto n\}}$$

$$\frac{}{\langle \texttt{input(x)}, m\rangle \Rightarrow m\{x \mapsto n\}}$$

$$\frac{\langle B, m\rangle \Rightarrow \mathsf{true} \qquad \langle C_1, m\rangle \Rightarrow m_1}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, m\rangle \Rightarrow m_1}$$

$$\frac{\langle B, m\rangle \Rightarrow \mathsf{false} \qquad \langle C_2, m\rangle \Rightarrow m_2}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, m\rangle \Rightarrow m_2}$$

$$\frac{\langle B, m\rangle \Rightarrow \mathsf{true} \qquad \langle C, m\rangle \Rightarrow m_1 \qquad \langle \texttt{while } B\ C, m_1\rangle \Rightarrow m_2}{\langle \texttt{while } B\ C, m\rangle \Rightarrow m_2}$$

$$\frac{\langle B, m\rangle \Rightarrow \mathsf{false}}{\langle \texttt{while } B\ C, m\rangle \Rightarrow m}$$

Exercise

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

Exercise

- `x := 1; if x > 0 then y := 1 else y := -1`

Exercise

- `x := 0; 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

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

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

$$
\begin{aligned}
[\![E]\!] : \mathbb{M} &\to \mathbb{Z} \\
[\![n]\!](m) &= n \\
[\![x]\!](m) &= m(x) \\
[\![E_1 \odot E_2]\!](m) &= [\![E_1]\!](m) \odot [\![E_2]\!](m)
\end{aligned}
$$

$$
\begin{aligned}
[\![B]\!] : \mathbb{M} &\to \mathbb{B} \\
[\![\mathsf{true}]\!](m) &= \mathsf{true} \\
[\![\mathsf{false}]\!](m) &= \mathsf{false} \\
[\![E_1 \circledcirc E_2]\!](m) &= [\![E_1]\!](m) \circledcirc [\![E_2]\!](m)
\end{aligned}
$$

$$\frac{\langle C_1, m\rangle \to \langle C_1', m'\rangle}{\langle C_1;\ C_2, m\rangle \to \langle C_1';\ C_2, m'\rangle}$$

$$\frac{}{\langle \texttt{skip};\ C_2, m\rangle \to \langle C_2, m\rangle}$$

$$\frac{[\![E]\!](m) = n}{\langle x := E, m\rangle \to \langle \texttt{skip}, m\{x \mapsto n\}\rangle}$$

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

$$\frac{[\![B]\!](m) = \mathsf{true}}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, m\rangle \to \langle C_1, m\rangle}$$

$$\frac{[\![B]\!](m) = \mathsf{false}}{\langle \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, m\rangle \to \langle C_2, m\rangle}$$

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

Exercise

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

Exercise

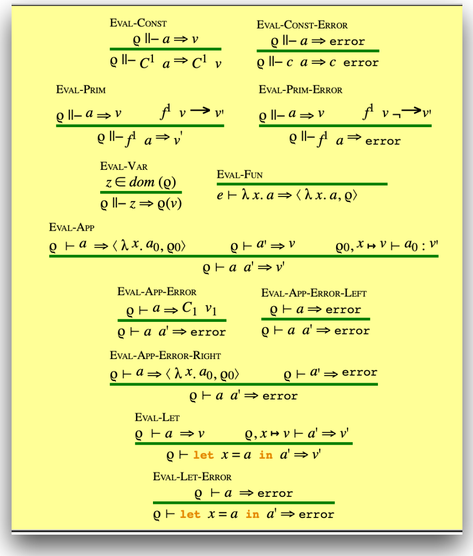
- `x := 1; if x > 0 then y := 1 else y := -1`

Exercise

- `x := 0; while (x < 10) (x := x + 1)`

Have Learned So Far

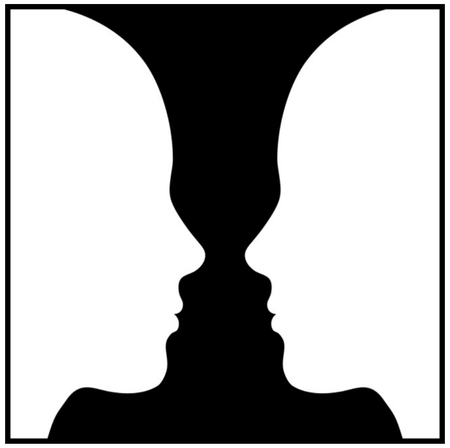
- Formal ways to describe programming language semantics
  - operational: inductive definition of machine state transitions
- Principles of programming systems (interpreters, compilers, analyzers, etc)



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 (some parts of) 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-lever (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' 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 $\langle \mathbb{L}, \texttt{next} \rangle$ where
  - $\mathbb{L}$ is a set of program labels
  - `next` control-flow relation
- Each program label l is associated with a command denoted by $\texttt{cmd}(l)$

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

Semantic Domains

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

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

Small-step Operation Semantics

$$
\begin{aligned}
[\![E]\!] : \mathbb{M} &\to \mathbb{V} \\
[\![n]\!](m) &= n \\
[\![\mathsf{true}]\!](m) &= \mathsf{true} \\
[\![\mathsf{false}]\!](m) &= \mathsf{false} \\
[\![x]\!](m) &= m(x) \\
[\![E_1 \oplus E_2]\!](m) &= [\![E_1]\!](m) \oplus [\![E_2]\!](m) \\
[\![E_1 \otimes E_2]\!](m) &= [\![E_1]\!](m) \otimes [\![E_2]\!](m)
\end{aligned}
$$

$$
\frac{\mathsf{cmd}(l) = \text{“}x := E\text{”} \qquad n = [\![E]\!](m) \qquad l' = \texttt{next}(l)}{\langle l, m \rangle \hookrightarrow \langle l', m\{x \mapsto n\} \rangle}
$$

$$
\frac{\mathsf{cmd}(l) = \text{“}\texttt{br}\ E\ l_1\ l_2\text{”} \qquad [\![E]\!](m) = \mathsf{true}}{\langle l, m \rangle \hookrightarrow \langle l_1, m \rangle}
$$

$$
\frac{\mathsf{cmd}(l) = \text{“}\texttt{br}\ E\ l_1\ l_2\text{”} \qquad [\![E]\!](m) = \mathsf{false}}{\langle l, m \rangle \hookrightarrow \langle l_2, m \rangle}
$$

$$
\frac{\mathsf{cmd}(l) = \text{“}\texttt{goto}\ l'\text{”}}{\langle l, m \rangle \hookrightarrow \langle l', m \rangle}
$$

$$
\frac{\mathsf{cmd}(l) = \text{“}x := \texttt{input}()\text{”} \qquad l' = \texttt{next}(l)}{\langle l, m \rangle \hookrightarrow \langle l', m\{x \mapsto n\} \rangle}
$$

$$
\frac{\mathsf{cmd}(l) = \text{“}\texttt{print}(x)\text{”} \qquad l' = \texttt{next}(l)}{\langle l, m \rangle \hookrightarrow \langle l', m \rangle}
$$

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**