Operational Semantics

Hongfei Fu

John Hopcroft Center for Computer Science
Shanghai Jiao Tong University

Mar. 4th, 2021

Today's Topic

Operational Semantics

- a simple imperative language as a minimal language
- a set of rules as building blocks for the semantics
- rule-based derivations as the operational semantics

Today's Topic

After the lecture, we will be able to ...

- know the logical background of operational semantics.
- know the necessary ingredients to construct operational semantics.

A Simple Imperative Language **IMP**

Textbook, Page 11 – Page 13

A Simple Imperative Language **IMP**

A Simple Imperative Language **IMP**

- data type: integers **N**
  (e.g., $0, 1, 2, \ldots, -1, -2, \ldots$)

A Simple Imperative Language **IMP**

- data type: integers $\mathbf{N}$
  (e.g., $0, 1, 2, \ldots, -1, -2, \ldots$)
- truth value: boolean values $\mathbf{T} = \{\textbf{true}, \textbf{false}\}$

A Simple Imperative Language **IMP**

- data type: integers **N**
  (e.g., $0, 1, 2, \ldots, -1, -2, \ldots$)
- truth value: boolean values $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$
- locations: **Loc** (identifiers or program variables)
  (e.g., x, y, i, j, a, b, $flag$, $\ldots$)

A Simple Imperative Language **IMP**

- data type: integers **N**
  (e.g., $0, 1, 2, \ldots, -1, -2, \ldots$)
- truth value: boolean values $\mathbf{T} = \{\textbf{true}, \textbf{false}\}$
- locations: **Loc** (identifiers or program variables)
  (e.g., x, y, i, j, a, b, $flag$, $\ldots$)
- arithmetic expressions: **Aexp**
  (e.g., $x + y$, $z - 3$, $x \times y$, $\ldots$)

A Simple Imperative Language **IMP**

- data type: integers **N**
  (e.g., $0, 1, 2, \ldots, -1, -2, \ldots$)
- truth value: boolean values $\mathbf{T} = \{\textbf{true}, \textbf{false}\}$
- locations: **Loc** (identifiers or program variables)
  (e.g., x, y, i, j, a, b, $flag, \ldots$)
- arithmetic expressions: **Aexp**
  (e.g., $x + y$, $z - 3$, $x \times y$, $\ldots$)
- boolean expressions: **Bexp**
  (e.g., $(x > 0) \wedge (y < 0)$, $(x > 0) \vee (y < 0)$, $\neg(x > y)$, $\ldots$)

A Simple Imperative Language **IMP**

- data type: integers **N**
  (e.g., $0, 1, 2, \ldots, -1, -2, \ldots$)
- truth value: boolean values $\mathbf{T} = \{\textbf{true}, \textbf{false}\}$
- locations: **Loc** (identifiers or program variables)
  (e.g., x, y, i, j, a, b, $flag$, $\ldots$)
- arithmetic expressions: **Aexp**
  (e.g., $x + y$, $z - 3$, $x \times y$, $\ldots$)
- boolean expressions: **Bexp**
  (e.g., $(x > 0) \wedge (y < 0)$, $(x > 0) \vee (y < 0)$, $\neg(x > y)$, $\ldots$)
- commands: statements **Com**
  (e.g., assignment, if branch, while loop, $\ldots$)

Program Syntax

Arithmetic Expressions **Aexp**

Arithmetic expressions are built from

- integers,
- locations (identifiers),
- arithmetic operations including $+, -, \times$.

Program Syntax

Arithmetic Expressions **Aexp**

Arithmetic expressions are built from

- integers,
- locations (identifiers),
- arithmetic operations including $+, -, \times$.

The syntax:

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

where n is any integer and X is any location.

Program Syntax

Boolean Expressions **Bexp**

Boolean expressions are built from

- truth values: **true**, **false**
- comparison: $=, \leq, \geq, <, >$
- propositional logical connectives: $\neg, \wedge, \vee$

Program Syntax

Boolean Expressions **Bexp**

Boolean expressions are built from

- truth values: **true**, **false**
- comparison: $=, \leq, \geq, <, >$
- propositional logical connectives: $\neg, \wedge, \vee$

The syntax

$$b ::= \textbf{true} \mid \textbf{false} \mid a \bowtie a' \mid \neg b \mid b \wedge b' \mid b \vee b'$$

where $\bowtie \in \{=, \leq, \geq, <, >\}$ and a, a' are arithmetic expressions.

Program Syntax

Commands **Com**

- assignment statements
- sequential composition
- if branches
- while loops

Program Syntax

Commands **Com**

The syntax of commands:

$$
\begin{aligned}
c ::= \ & \textbf{skip} \\
\mid \ & X := a \\
\mid \ & c_0;\ c_1 \\
\mid \ & \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 \\
\mid \ & \textbf{while } b \textbf{ do } c'
\end{aligned}
$$

Program Syntax

A Variant of the Euclidean Algorithm

$$\begin{array}{l}\textbf{while}\ \ \neg(M = N)\ \ \textbf{do}\\ \quad \textbf{if}\ \ M \leq N\\ \qquad \textbf{then}\ \ N := N - M\\ \qquad \textbf{else}\ \ M := M - N\end{array}$$

Program Syntax

IMP allows

- integer type,
- assignment,
- sequential composition,
- conditional branch,
- while loop.

Program Syntax

IMP does not allow

- data structures,
- floating numbers,
- recursion,
- pointers,
- ...

The Operational Semantics of **IMP**

Textbook, Page 15 – 20

Operational Semantics

Overview

- rules for arithmetic/boolean expressions
- rules for commands (statements)
- derivations for the final operational semantics

Operational Semantics

States

- A state is a function $\sigma : \mathbf{Loc} \to \mathbf{N}$.
- The set of states is denote by Σ.

Operational Semantics

States

- A state is a function $\sigma : \mathbf{Loc} \to \mathbf{N}$.
- The set of states is denote by Σ.

Intuition

A state specifies values held by locations.

Operational Semantics

Our Goal

A relation $\mathcal{R} \subseteq (\textbf{Com} \times \Sigma) \times \Sigma$ such that $(c, \sigma)\, \mathcal{R}\, \sigma'$ iff

when executing c with initial state σ, c terminates and we eventually get σ' after the execution.

Operational Semantics

Our Goal

A relation $\mathcal{R} \subseteq (\mathbf{Com} \times \Sigma) \times \Sigma$ such that $(c, \sigma)\,\mathcal{R}\,\sigma'$ iff

when executing c with initial state σ, c terminates and we eventually get σ' after the execution.

We often write $(c, \sigma) \to \sigma'$ instead of $(c, \sigma)\,\mathcal{R}\,\sigma'$.

Operational Semantics

Question

How can we construct such a relation?

Operational Semantics

Question

How can we construct such a relation?

The Methodology

- from rules to derivations
- from arithmetic expressions to commands

Arithmetic Expressions

Configurations (for **Aexp**)

A configuration is a pair $\langle a, \sigma \rangle$ where $a \in \mathbf{Aexp}$ and $\sigma \in \Sigma$.

Arithmetic Expressions

Configurations (for **Aexp**)

A configuration is a pair $\langle a, \sigma \rangle$ where $a \in \mathbf{Aexp}$ and $\sigma \in \Sigma$.

Sub-goal

a relation for $\langle a, \sigma \rangle \to n$:

an arithmetic expression a is evaluated to an integer n when locations in a are substituted by their values from σ.

Arithmetic Expressions

Question

How can we define "$\langle a, \sigma \rangle \to n$" rigorously?

Arithmetic Expressions

Question

How can we define "$\langle a, \sigma \rangle \to n$" rigorously?

Principles

- The definition should be syntactical.
- The definition should be correct.

Arithmetic Expressions

The Intuition

How can we evaluate $a_0 + a_1$ under a state σ ?

Arithmetic Expressions

The Intuition

How can we evaluate $a_0 + a_1$ under a state σ ?

Arithmetic Expressions

The Intuition

How can we evaluate $a_0 + a_1$ under a state σ ?

- first evaluate a_0, a_1 correctly: $\langle a_0, \sigma \rangle \to n_0$, $\langle a_1, \sigma \rangle \to n_1$

Arithmetic Expressions

The Intuition

How can we evaluate $a_0 + a_1$ under a state σ ?

- first evaluate a_0, a_1 correctly: $\langle a_0, \sigma \rangle \to n_0$, $\langle a_1, \sigma \rangle \to n_1$
- then evaluate $a_0 + a_1$ correctly: $\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1$

Arithmetic Expressions

The Intuition

How can we evaluate $a_0 + a_1$ under a state σ ?

- first evaluate a_0, a_1 correctly: $\langle a_0, \sigma \rangle \to n_0$, $\langle a_1, \sigma \rangle \to n_1$
- then evaluate $a_0 + a_1$ correctly: $\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1$

Implementation: rules and derivations!

The Rule for Addition

$$\frac{\langle a_0, \sigma \rangle \to n_0, \ \langle a_1, \sigma \rangle \to n_1}{\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1}$$

- if $\langle a_0, \sigma \rangle \to n_0$ and $\langle a_1, \sigma \rangle \to n_1$, then $\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1$;

Arithmetic Expressions

The Rule for Addition

$$\frac{\langle a_0, \sigma \rangle \to n_0, \ \langle a_1, \sigma \rangle \to n_1}{\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1}$$

- if $\langle a_0, \sigma \rangle \to n_0$ and $\langle a_1, \sigma \rangle \to n_1$, then $\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1$;
- premise: $\langle a_0, \sigma \rangle \to n_0$ and $\langle a_1, \sigma \rangle \to n_1$;
- conclusion: $\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1$;

Arithmetic Expressions

How to build rules?

- Establish rules for each arithmetic operation.
  (e.g., addition, subtraction, multiplication)
- Prove correctness for each rule.
  (i.e., proving that the premise implies the conclusion)

Arithmetic Expressions

Numbers and Locations

$$\frac{}{\langle n, \sigma \rangle \to n} \qquad \frac{}{\langle X, \sigma \rangle \to \sigma(X)}$$

- axioms: rules without premise
- metavariables: n, X, σ

Arithmetic Expressions

Arithmetic Operations

$$\frac{\langle a_0, \sigma \rangle \to n_0, \ \langle a_1, \sigma \rangle \to n_1}{\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1} \qquad \frac{\langle a_0, \sigma \rangle \to n_0, \ \langle a_1, \sigma \rangle \to n_1}{\langle a_0 - a_1, \sigma \rangle \to n_0 - n_1}$$

$$\frac{\langle a_0, \sigma \rangle \to n_0, \ \langle a_1, \sigma \rangle \to n_1}{\langle a_0 \times a_1, \sigma \rangle \to n_0 \cdot n_1}$$

- $n_0, n_1, a_0, a_1, \sigma$: metavariables

Arithmetic Expressions

Rule Instances

A rule instance is obtained from substituting metavariables by concrete elements.

Arithmetic Expressions

Rule Instances

A rule instance is obtained from substituting metavariables by concrete elements.

Examples

$$\frac{}{\langle 5, \sigma \rangle \to 5} \qquad \frac{}{\langle X, \{X \mapsto 4, Y \mapsto 5\} \rangle \to 4}$$

$$\frac{\langle a_0, \sigma \rangle \to 2, \ \langle a_1, \sigma \rangle \to 3}{\langle a_0 \times a_1, \sigma \rangle \to 6} (a_i\text{'s are concrete arithmetic expressions})$$

Arithmetic Expressions

Question

How can we organize rules for compound arithmetic expressions?

Derivation Trees

Arithmetic Expressions

Derivation Trees

- $\sigma(X) = 1, \sigma(Y) = -1$

Arithmetic Expressions

Derivation Trees

- $\sigma(X) = 1, \sigma(Y) = -1$
- the evaluation of $\langle (X + 5) - (Y \times 2), \sigma \rangle$:

Arithmetic Expressions

Derivation Trees

- $\sigma(X) = 1, \sigma(Y) = -1$
- the evaluation of $\langle (X + 5) - (Y \times 2), \sigma \rangle$:

$$\dfrac{\dfrac{\dfrac{}{\langle X, \sigma \rangle \to 1} \quad \dfrac{}{\langle 5, \sigma \rangle \to 5}}{\langle X + 5, \sigma \rangle \to 6} \quad \dfrac{\dfrac{}{\langle Y, \sigma \rangle \to -1} \quad \dfrac{}{\langle 2, \sigma \rangle \to 2}}{\langle Y \times 2, \sigma \rangle \to -2}}{\langle (X + 5) - (Y \times 2), \sigma \rangle \to 8}$$

Arithmetic Expressions

Derivation Trees

- $\sigma(X) = 1, \sigma(Y) = -1$
- the evaluation of $\langle (X + 5) - (Y \times 2), \sigma \rangle$:

$$\dfrac{\dfrac{\dfrac{}{\langle X, \sigma \rangle \to 1} \quad \dfrac{}{\langle 5, \sigma \rangle \to 5}}{\langle X + 5, \sigma \rangle \to 6} \qquad \dfrac{\dfrac{}{\langle Y, \sigma \rangle \to -1} \quad \dfrac{}{\langle 2, \sigma \rangle \to 2}}{\langle Y \times 2, \sigma \rangle \to -2}}{\langle (X + 5) - (Y \times 2), \sigma \rangle \to 8}$$

- conclusion: $\langle (X + 5) - (Y \times 2), \sigma \rangle \to 8$

Arithmetic Expressions

Derivation Tree

A derivation tree (derivation) is a finite tree such that every parent-children substructure in the tree is a rule instance.

Arithmetic Expressions

Derivation Tree

A derivation tree (derivation) is a finite tree such that every parent-children substructure in the tree is a rule instance.

Definitions

Arithmetic Expressions

Derivation Tree

A derivation tree (derivation) is a finite tree such that every parent-children substructure in the tree is a rule instance.

Definitions

- definition: $\langle a, \sigma \rangle \to n$ iff there is a derivation tree with conclusion $\langle a, \sigma \rangle \to n$.

Arithmetic Expressions

Derivation Tree

A derivation tree (derivation) is a finite tree such that every parent-children substructure in the tree is a rule instance.

Definitions

- definition: $\langle a, \sigma \rangle \to n$ iff there is a derivation tree with conclusion $\langle a, \sigma \rangle \to n$.
- property: $\forall a. \forall \sigma. \exists n. \langle a, \sigma \rangle \to n$

Arithmetic Expressions

Derivation Tree

A derivation tree (derivation) is a finite tree such that every parent-children substructure in the tree is a rule instance.

Definitions

- definition: $\langle a, \sigma \rangle \to n$ iff there is a derivation tree with conclusion $\langle a, \sigma \rangle \to n$.
- property: $\forall a. \forall \sigma. \exists n. \langle a, \sigma \rangle \to n$
- equivalence: $a \sim a'$ iff $\forall n. \forall \sigma. (\langle a, \sigma \rangle \to n \Leftrightarrow \langle a', \sigma \rangle \to n)$

Arithmetic Expressions

Derivation Tree

A derivation tree (derivation) is a finite tree such that every parent-children substructure in the tree is a rule instance.

Definitions

- definition: $\langle a, \sigma \rangle \to n$ iff there is a derivation tree with conclusion $\langle a, \sigma \rangle \to n$.
- property: $\forall a. \forall \sigma. \exists n. \langle a, \sigma \rangle \to n$
- equivalence: $a \sim a'$ iff $\forall n. \forall \sigma. (\langle a, \sigma \rangle \to n \Leftrightarrow \langle a', \sigma \rangle \to n)$
- big-step semantics: internal computation is omitted.

Arithmetic Expressions

Derivation Tree

A derivation tree (derivation) is a finite tree such that every parent-children substructure in the tree is a rule instance.

Definitions

- definition: $\langle a, \sigma \rangle \to n$ iff there is a derivation tree with conclusion $\langle a, \sigma \rangle \to n$.
- property: $\forall a. \forall \sigma. \exists n. \langle a, \sigma \rangle \to n$
- equivalence: $a \sim a'$ iff $\forall n. \forall \sigma. (\langle a, \sigma \rangle \to n \Leftrightarrow \langle a', \sigma \rangle \to n)$
- big-step semantics: internal computation is omitted.
- missing rigor: derivation trees

Boolean Expressions

Truth Values

$$\frac{}{\langle \textbf{true}, \sigma \rangle \to \textbf{true}} \qquad \frac{}{\langle \textbf{false}, \sigma \rangle \to \textbf{false}}$$

Boolean Expressions

Comparison

$$\frac{\langle a_0, \sigma\rangle \to n_0,\ \langle a_1, \sigma\rangle \to n_1}{\langle a_0 = a_1, \sigma\rangle \to \textbf{true}} \text{ if } n_0 = n_1$$

$$\frac{\langle a_0, \sigma\rangle \to n_0,\ \langle a_1, \sigma\rangle \to n_1}{\langle a_0 = a_1, \sigma\rangle \to \textbf{false}} \text{ if } n_0 \neq n_1$$

Boolean Expressions

Comparison

$$\frac{\langle a_0, \sigma \rangle \to n_0,\ \langle a_1, \sigma \rangle \to n_1}{\langle a_0 \leq a_1, \sigma \rangle \to \textbf{true}} \text{ if } n_0 \leq n_1$$

$$\frac{\langle a_0, \sigma \rangle \to n_0,\ \langle a_1, \sigma \rangle \to n_1}{\langle a_0 \leq a_1, \sigma \rangle \to \textbf{false}} \text{ if } n_0 > n_1$$

Boolean Expressions

Negation

$$\frac{\langle b, \sigma \rangle \to \textbf{true}}{\langle \neg b, \sigma \rangle \to \textbf{false}} \qquad \frac{\langle b, \sigma \rangle \to \textbf{false}}{\langle \neg b, \sigma \rangle \to \textbf{true}}$$

Boolean Expressions

Disjunction and Conjunction

$$\frac{\langle b_0, \sigma \rangle \to t_0, \ \langle b_1, \sigma \rangle \to t_1}{\langle b_0 \wedge b_1, \sigma \rangle \to t_0 \wedge t_1}$$

$$\frac{\langle b_0, \sigma \rangle \to t_0, \ \langle b_1, \sigma \rangle \to t_1}{\langle b_0 \vee b_1, \sigma \rangle \to t_0 \vee t_1}$$

Boolean Expressions

Definition

- definition: $\langle b, \sigma \rangle \to t$ iff there is a derivation tree with conclusion $\langle b, \sigma \rangle \to t$.
- property: $\forall b. \forall \sigma. \exists t. \langle b, \sigma \rangle \to t$
- equivalence: $b \sim b'$ iff $\forall t \in \{\textbf{true}, \textbf{false}\}. \forall \sigma. (\langle b, \sigma \rangle \to t \Leftrightarrow \langle b', \sigma \rangle \to t)$
- big-step semantics: Internal computation is omitted.
- missing rigor: derivation trees

Commands

Skip

$$\frac{}{\langle \textbf{skip}, \sigma \rangle \to \sigma}$$

Commands

Substitution over States

- σ: a state
- m: an integer
- X: a location (program variable)

We define $\sigma\left[m/X\right]$ by

$$\sigma\left[m/X\right](Y) := \begin{cases} m & \text{if } Y = X \\ \sigma(Y) & \text{otherwise} \end{cases}$$

Commands

Assignment Statements

$$\frac{\langle a, \sigma \rangle \to m}{\langle X := a, \sigma \rangle \to \sigma\,[m/X]}$$

Commands

Sequential Composition

$$\frac{\langle c_0, \sigma \rangle \to \sigma'', \ \langle c_1, \sigma'' \rangle \to \sigma'}{\langle c_0; \ c_1, \sigma \rangle \to \sigma'}$$

Commands

Conditional Branches

$$\frac{\langle b, \sigma \rangle \to \textbf{true}, \ \langle c_0, \sigma \rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma \rangle \to \sigma'}$$

$$\frac{\langle b, \sigma \rangle \to \textbf{false}, \ \langle c_1, \sigma \rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma \rangle \to \sigma'}$$

Commands

While Loops

$$\frac{\langle b, \sigma \rangle \to \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \sigma}$$

$$\frac{\langle b, \sigma \rangle \to \textbf{true},\ \langle c, \sigma \rangle \to \sigma'',\ \langle \textbf{while } b \textbf{ do } c, \sigma'' \rangle \to \sigma'}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \sigma'}$$

Commands

Definitions

- definition: $\langle c, \sigma \rangle \to \sigma'$ iff there is a derivation tree with conclusion $\langle c, \sigma \rangle \to \sigma'$.
- equivalence: $c \sim c'$ iff $\forall \sigma, \sigma' . (\langle c, \sigma \rangle \to \sigma' \Leftrightarrow \langle c', \sigma \rangle \to \sigma')$
- big-step semantics: Internal computation is omitted.
- missing rigor: derivation trees

Commands

An Example

$$\begin{array}{l}\textbf{while } \neg(M = N) \textbf{ do} \\ \quad \textbf{if } M \leq N \\ \qquad \textbf{then } N := N - M \\ \qquad \textbf{else } M := M - N\end{array}$$

Commands

Another Example

while true do skip

Commands

Question

What if there is no σ' such that $\langle c, \sigma \rangle \to \sigma'$?

Summary

- a simple imperative language **IMP**
- a first look at operational semantics
- rules and derivations

Questions?

Homework

Let X, Y be locations (i.e., program variables). Let the state σ be given by $\sigma(X) = 3$ and $\sigma(Y) = 5$. Solve the following problems through derivation trees.

(a) For $a = X - 1$, determine the integer n such that $\langle a, \sigma \rangle \to n$.

(b) For $b = Y - X \leq 2$, determine the truth value t such that $\langle b, \sigma \rangle \to t$.

(c) For $c =$ **if** $Y - X \leq 2$ **then** $Y := X - 1$ **else skip** , determine the state σ' such that $\langle c, \sigma \rangle \to \sigma'$.