

CSED490O: Automated Software Verification

Programming Language Semantics in Maude

Kyungmin Bae

Department of Computer Science and Engineering
POSTECH

Logic vs. Programming Language

- ▶ Logic
 - ▶ **syntax**: a formal language
 - ▶ **semantics**: its interpretation
- ▶ Programming Language
 - ▶ vocabulary and grammar
 - ▶ the meaning of programs

IMP: A Very Simple Imperative Language

- ▶ Expressions
 - ▶ arithmetic expressions (over integers), and Boolean expressions
- ▶ Statements
 - ▶ (sequences of) assignment, conditional, and while statements
- ▶ Example

```
vars n, i, s;
n := 100;
i := 0;
s := 0;
while i <= n do
    s := s + i;
    i := i + 1
```

IMP: Syntax

expression $e ::= n \mid b \mid x$
 $\mid e + e \mid e / e \mid e \leq e \mid e \&\& e \mid !e$

statement $s ::= \text{skip} \mid x := e \mid s ; s$
 $\mid \text{if } e \text{ then } s \text{ else } s \mid \text{while } e \text{ do } s$

program $p ::= \text{vars } v\mid ; s$
 $v\mid ::= x \mid v\mid, v\mid$

- ▶ n is an integer, b is a Boolean value, and x is a variable identifier

Operational Semantics

- ▶ Specifies how to **execute** programs
 - ▶ given by **transition relations** on configurations
 - ▶ where **configurations** contain all information for program “states”
- ▶ E.g., a configuration of IMP is a tuple $\langle p, \sigma \rangle$
 - ▶ p : a program fragment to be executed
 - ▶ σ : values of variables (i.e., assignment $x_1 \mapsto v_1; \dots; x_n \mapsto v_n$)

Structural Operational Semantics

Structural Operational Semantics (SOS)

- ▶ Defined inductively over the structure of the syntax
 - ▶ very common (traditional) operational semantics approach
- ▶ **Big-step SOS** defines relations of configurations $C \Downarrow C'$
 - ▶ C' is obtained after the **complete evaluation** of C
- ▶ **Small-step SOS** defines relations of configurations $C \rightarrow C'$
 - ▶ C' is obtained by **one-step evaluation** of C

Big-Step SOS of IMP

- ▶ Can be defined as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ where
 - ▶ $C \Downarrow C' \iff \mathcal{R} \vdash \Downarrow(C) \longrightarrow C'$
 - ▶ (Σ, E) declares terms for IMP programs and configurations
- ▶ Two kinds of rules
 - ▶ $\Downarrow(e, \sigma) \longrightarrow v$: an expression e evaluates to the value v in state σ
 - ▶ $\Downarrow(s, \sigma) \longrightarrow \sigma'$: executing a statement s from state σ results in σ'

Big-Step SOS: Expressions

$$\Downarrow(n, \sigma) \longrightarrow n$$

$$\Downarrow(b, \sigma) \longrightarrow b$$

$$\Downarrow(x, x \mapsto v; \sigma) \longrightarrow v$$

$$\Downarrow(e_1 + e_2, \sigma) \longrightarrow n_1 + n_2 \quad \text{if } \Downarrow(e_1, \sigma) \longrightarrow n_1 \wedge \Downarrow(e_2, \sigma) \longrightarrow n_2$$

$$\Downarrow(e_1 / e_2, \sigma) \longrightarrow n_1 / n_2 \quad \text{if } \Downarrow(e_1, \sigma) \longrightarrow n_1 \wedge \Downarrow(e_2, \sigma) \longrightarrow n_2 \wedge n_2 \neq 0$$

$$\Downarrow(e_1 \leq e_2, \sigma) \longrightarrow \text{true} \quad \text{if } \Downarrow(e_1, \sigma) \longrightarrow n_1 \wedge \Downarrow(e_2, \sigma) \longrightarrow n_2 \wedge n_1 \leq n_2$$

$$\Downarrow(e_1 \leq e_2, \sigma) \longrightarrow \text{false} \quad \text{if } \Downarrow(e_1, \sigma) \longrightarrow n_1 \wedge \Downarrow(e_2, \sigma) \longrightarrow n_2 \wedge n_1 > n_2$$

$$\Downarrow(e_1 \&\& e_2, \sigma) \longrightarrow \text{and}(b_1, b_2) \quad \text{if } \Downarrow(e_1, \sigma) \longrightarrow b_1 \wedge \Downarrow(e_2, \sigma) \longrightarrow b_2$$

$$\Downarrow(! e, \sigma) \longrightarrow \text{not}(b) \quad \text{if } \Downarrow(e, \sigma) \longrightarrow b$$

Big-Step SOS: Statements and Programs

$$\Downarrow(\text{skip}, \sigma) \longrightarrow \sigma$$

$$\Downarrow(x := e, x \mapsto v; \sigma) \longrightarrow x \mapsto v'; \sigma \quad \text{if } \Downarrow(e, \sigma) \longrightarrow v'$$

$$\Downarrow(s_1; s_2, \sigma) \longrightarrow \sigma_2$$

$$\quad \text{if } \Downarrow(s_1, \sigma) \longrightarrow \sigma_1 \wedge \Downarrow(s_2, \sigma_1) \longrightarrow \sigma_2$$

$$\Downarrow(\text{if } e \text{ then } s_1 \text{ else } s_2, \sigma) \longrightarrow \sigma'$$

$$\quad \text{if } \Downarrow(e, \sigma) \longrightarrow \text{true} \wedge \Downarrow(s_1, \sigma) \longrightarrow \sigma'$$

$$\Downarrow(\text{if } e \text{ then } s_1 \text{ else } s_2, \sigma) \longrightarrow \sigma'$$

$$\quad \text{if } \Downarrow(e, \sigma) \longrightarrow \text{false} \wedge \Downarrow(s_2, \sigma) \longrightarrow \sigma'$$

$$\Downarrow(\text{while } e \text{ do } s, \sigma) \longrightarrow \sigma$$

$$\quad \text{if } \Downarrow(e, \sigma) \longrightarrow \text{false}$$

$$\Downarrow(\text{while } e \text{ do } s, \sigma) \longrightarrow \sigma'$$

$$\quad \text{if } \Downarrow(e, \sigma) \longrightarrow \text{true} \wedge$$

$$\Downarrow(s; \text{while } e \text{ do } s, \sigma) \longrightarrow \sigma'$$

$$\Downarrow(\text{vars } x_1, \dots, x_n; s, \emptyset) \longrightarrow \sigma$$

$$\quad \text{if } \Downarrow(s, x_1 \mapsto 0; \dots; x_n \mapsto 0) \longrightarrow \sigma$$

Determinism

Lemma 1

- ▶ If $\Downarrow(e, \sigma) \rightarrow v_1$ and $\Downarrow(e, \sigma) \rightarrow v_2$, then $v_1 = v_2$.
- ▶ If $\Downarrow(s, \sigma) \rightarrow \sigma_1$ and $\Downarrow(s, \sigma) \rightarrow \sigma_2$, then $\sigma_1 = \sigma_2$.
- ▶ If $\Downarrow(p, \emptyset) \rightarrow \sigma_1$ and $\Downarrow(p, \emptyset) \rightarrow \sigma_2$, then $\sigma_1 = \sigma_2$.

Small-Step SOS of IMP

- ▶ Can be defined as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ where
 - ▶ $C \rightarrow C' \iff \mathcal{R} \vdash \langle C \rangle \longrightarrow \langle C' \rangle$
 - ▶ (Σ, E) declares terms for IMP programs and configurations

Small-Step SOS: Expressions (1)

$$\langle x, x \mapsto v ; \sigma \rangle \longrightarrow \langle v, x \mapsto v ; \sigma \rangle$$

$$\langle e_1 + e_2, \sigma \rangle \longrightarrow \langle e'_1 + e_2, \sigma \rangle \quad \text{if } \langle e_1, \sigma \rangle \longrightarrow \langle e'_1, \sigma \rangle$$

$$\langle e_1 + e_2, \sigma \rangle \longrightarrow \langle e_1 + e'_2, \sigma \rangle \quad \text{if } \langle e_2, \sigma \rangle \longrightarrow \langle e'_2, \sigma \rangle$$

$$\langle n_1 + n_2, \sigma \rangle \longrightarrow \langle n, \sigma \rangle \quad \text{if } n \text{ is the sum of } n_1 \text{ and } n_2$$

$$\langle e_1 / e_2, \sigma \rangle \longrightarrow \langle e'_1 / e_2, \sigma \rangle \quad \text{if } \langle e_1, \sigma \rangle \longrightarrow \langle e'_1, \sigma \rangle$$

$$\langle e_1 / e_2, \sigma \rangle \longrightarrow \langle e_1 / e'_2, \sigma \rangle \quad \text{if } \langle e_2, \sigma \rangle \longrightarrow \langle e'_2, \sigma \rangle$$

$$\langle n_1 / n_2, \sigma \rangle \longrightarrow \langle n, \sigma \rangle \quad \text{if } n_2 \neq 0 \wedge n \text{ is the quotient of } n_1 \text{ by } n_2$$

Small-Step SOS: Expressions (2)

$$\langle e_1 \leq e_2, \sigma \rangle \longrightarrow \langle e'_1 \leq e_2, \sigma \rangle \quad \text{if } \langle e_1, \sigma \rangle \longrightarrow \langle e'_1, \sigma \rangle$$

$$\langle e_1 \leq e_2, \sigma \rangle \longrightarrow \langle e_1 \leq e'_2, \sigma \rangle \quad \text{if } \langle e_2, \sigma \rangle \longrightarrow \langle e'_2, \sigma \rangle$$

$$\langle n_1 \leq n_2, \sigma \rangle \longrightarrow \langle \text{true}, \sigma \rangle \quad \text{if } n_1 \leq n_2$$

$$\langle n_1 \leq n_2, \sigma \rangle \longrightarrow \langle \text{false}, \sigma \rangle \quad \text{if } n_1 > n_2$$

$$\langle e_1 \&\& e_2, \sigma \rangle \longrightarrow \langle e'_1 \&\& e_2, \sigma \rangle \quad \text{if } \langle e_1, \sigma \rangle \longrightarrow \langle e'_1, \sigma \rangle$$

$$\langle \text{true} \&\& e_2, \sigma \rangle \longrightarrow \langle e_2, \sigma \rangle$$

$$\langle \text{false} \&\& e_2, \sigma \rangle \longrightarrow \langle \text{false}, \sigma \rangle$$

$$\langle ! e, \sigma \rangle \longrightarrow \langle ! e', \sigma \rangle \quad \text{if } \langle e, \sigma \rangle \longrightarrow \langle e', \sigma \rangle$$

$$\langle ! b, \sigma \rangle \longrightarrow \langle \text{not}(b), \sigma \rangle$$

Small-Step SOS: Statements and Programs

$$\langle x := e, \sigma \rangle \longrightarrow \langle x := e', \sigma \rangle \quad \text{if } \langle e, \sigma \rangle \longrightarrow \langle e', \sigma \rangle$$

$$\langle x := v', x \mapsto v ; \sigma \rangle \longrightarrow \langle \text{skip}, x \mapsto v' ; \sigma \rangle$$

$$\langle s_1; s_2, \sigma \rangle \longrightarrow \langle s'_1; s_2, \sigma' \rangle \quad \text{if } \langle s_1, \sigma \rangle \longrightarrow \langle s'_1, \sigma' \rangle$$

$$\langle \text{skip}; s_2, \sigma \rangle \longrightarrow \langle s_2, \sigma \rangle$$

$$\langle \text{if } e \text{ then } s_1 \text{ else } s_2, \sigma \rangle \longrightarrow \langle \text{if } e' \text{ then } s_1 \text{ else } s_2, \sigma \rangle \quad \text{if } \langle e, \sigma \rangle \longrightarrow \langle e', \sigma \rangle$$

$$\langle \text{if } \text{true} \text{ then } s_1 \text{ else } s_2, \sigma \rangle \longrightarrow \langle s_1, \sigma \rangle$$

$$\langle \text{if } \text{false} \text{ then } s_1 \text{ else } s_2, \sigma \rangle \longrightarrow \langle s_2, \sigma \rangle$$

$$\langle \text{while } e \text{ do } s, \sigma \rangle \longrightarrow \langle \text{if } e \text{ then } (s ; \text{while } e \text{ do } s) \text{ else skip}, \sigma \rangle$$

$$\langle \text{vars } x_1, \dots, x_n; s, \emptyset \rangle \longrightarrow \langle s, x_1 \mapsto 0 ; \dots ; x_n \mapsto 0 \rangle$$

Relating Big-Step and Small-Step SOS Definitions

Proposition 1

For any IMP program p , $(p, \emptyset) \Downarrow \sigma \iff \langle p, \emptyset \rangle \rightarrow \langle \text{skip}, \sigma \rangle$.

Discussion: Big-Step vs. Small-Step SOS

- ▶ Advantages of Big-step SOS
 - ▶ easier to understand and implement
 - ▶ easier to prove properties about programs
- ▶ Advantages of Small-step SOS
 - ▶ “**executable**” and thus easier to trace
 - ▶ well support non-determinism

Disadvantages of Both Big-Step and Small-Step SOS (1)

- ▶ Not **modular**
 - ▶ adding new features requires modifying definitions of unrelated features
- ▶ Example: add a variable increment expression to IMP

`e ::= ... | ++ x`

Disadvantages of Both Big-Step and Small-Step SOS (2)

- ▶ Hard to define control-intensive language features
 - ▶ exit (or halt), return, goto, ...
- ▶ Example: add a halt statement to IMP

s ::= ... | halt

Disadvantages of Both Big-Step and Small-Step SOS (3)

- ▶ Hard to define concurrency features
 - ▶ multiple threads/processes, synchronization, ...
- ▶ Big-step SOS
 - ▶ not suitable for defining any meaningful concurrent language
- ▶ Small-step SOS
 - ▶ only interleaving concurrency (no sideways or nested concurrency)

K Framework

K Framework

- ▶ A rewriting-based semantic definitional framework
 - ▶ a modular way to define concurrent programming languages
- ▶ Many programming languages are (completely) specified in K
 - ▶ C, Java, Javascript, Verilog, Python, Ethereum VM (Blockchain), ...
- ▶ There are dedicated tools for K (<http://www.kframework.org>)
 - ▶ but K-based semantics can also be specified in Maude

Key Ideas of K

- ▶ Computations as algebraic structures
- ▶ Configurations as (nested) algebraic structures
- ▶ Rewrite rules on configurations for (concurrent) executions

Evaluation Context

- ▶ Consider a small-step evaluation of the expression

$$1 + (x * 4)$$

- ▶ The **position** of the evaluation can be expressed using a **hole**:

$$1 + (\square * 4)$$

- ▶ Often combined with SOS approaches for modularity

Computations

- ▶ A sequence of computational tasks (to be processed sequentially)

$$\kappa_1 \curvearrowright \kappa_2 \curvearrowright \kappa_3 \curvearrowright \cdots \curvearrowright \kappa_n$$

- ▶ A computational task (called a K label) includes
 - ▶ (fragments of) programs, evaluation contexts, ...
- ▶ Example

$$x \curvearrowright \square + 4 \curvearrowright 1 + \square$$

Heating and Cooling Rules

- ▶ Heating rules decompose a task into a sequence of computations

$$p \rightarrow p' \curvearrowright C$$

- ▶ Cooling rules integrates a (completed) task with the next task

$$p \curvearrowright C \rightarrow C'$$

- ▶ Heating and cooling rules are typically paired and often written

$$t \rightleftharpoons t'$$

- ▶ Declared using equations

- ▶ heating/cooling rules specify structural properties of computations

Heating and Cooling Rules for IMP

$$e_1 + e_2 \Rightarrow e_1 \curvearrowright \square + e_2$$

$$v_1 + e_2 \Rightarrow e_2 \curvearrowright v_1 + \square$$

$$e_1 / e_2 \Rightarrow e_1 \curvearrowright \square / e_2$$

$$v_1 / e_2 \Rightarrow e_2 \curvearrowright v_1 / \square$$

$$e_1 \leq e_2 \Rightarrow e_1 \curvearrowright \square \leq e_2$$

$$v_1 \leq e_2 \Rightarrow e_2 \curvearrowright v_1 \leq \square$$

$$e_1 \&\& e_2 \Rightarrow e_1 \curvearrowright \square \&\& e_2$$

$$!e \Rightarrow e \curvearrowright !\square$$

$$x := e \Rightarrow e \curvearrowright x := \square$$

$$\text{if } e \text{ then } s_1 \text{ else } s_2 \Rightarrow e \curvearrowright \text{if } \square \text{ then } s_1 \text{ else } s_2$$

K Configurations

- ▶ All information for program “states”
 - ▶ including computations, variable environments, threads, etc.
- ▶ A (nested) multiset of configuration items
 - ▶ represented as algebraic terms (using equational theories)

K Configurations for IMP

- ▶ Configurations items
 - ▶ $k : K \rightarrow \text{ConfigItem}$: a computation (of sort K)
 - ▶ $\text{env} : \text{Map}\{\text{Id}, \text{Value}\} \rightarrow \text{ConfigItem}$: an assignment
- ▶ Example:

$$k(x \curvearrowright \square + 4 \curvearrowright 1 + \square \curvearrowright y := \square) \text{ env}(x \mapsto 1; y \mapsto 0)$$

K Rules

- ▶ Specify (current) execution steps on K configurations. E.g.:

$$\begin{aligned} & k(x \curvearrowright K) \ env(x \mapsto v; ENV) \\ \longrightarrow & k(v \curvearrowright K) \ env(x \mapsto v; ENV) \end{aligned}$$

- ▶ Often written as

$$k(\underline{x} \curvearrowright K) \ env(x \mapsto v; ENV)$$

K Rules for IMP: Expressions

$k(\frac{x \curvearrowright K}{v} \ env(x \mapsto v; ENV))$

$n_1 + n_2 \longrightarrow n$ **if** n is the sum of n_1 and n_2

$n_1 / n_2 \longrightarrow n$ **if** $n_2 \neq 0 \wedge n$ is the quotient of n_1 by n_2

$n_1 \leq n_2 \longrightarrow true$ **if** $n_1 \leq n_2$

$n_1 \leq n_2 \longrightarrow false$ **if** $n_1 > n_2$

$true \&\& e_2 \longrightarrow e_2$

$false \&\& e_2 \longrightarrow false$

$! b \longrightarrow not(b)$

K Rules for IMP: Statements and Programs

$$k(\underline{\text{skip}} \curvearrowright K)$$

$$k(x := v \curvearrowright K) \text{ env}(x \mapsto \frac{v'}{v}; ENV)$$

$$s_1; s_2 \longrightarrow s_1 \curvearrowright s_2$$

if *true* then s_1 else $s_2 \longrightarrow s_1$

if *false* then s_1 else $s_2 \longrightarrow s_2$

while e do $s \longrightarrow$ if e then $(s ; \text{while } e \text{ do } s)$ else skip

$$k(\underline{\text{vars } x_1, \dots, x_n; s}) \text{ env}(\frac{}{x_1 \mapsto 0; \dots; x_n \mapsto 0}, \emptyset)$$

IMP++: A Concurrent Extension of IMP

expression $e ::= \dots \mid ++x \mid \text{read}$

statement $s ::= \dots \mid \text{halt} \mid \text{spawn } s \mid \text{print } e$

- ▶ How can the K semantics of IMP can be extended into IMP++?

Heating and Cooling Rules for IMP++

$$\text{print } e \iff e \curvearrowright \text{print } \square$$

K Configurations for IMP++

- ▶ New configurations items
 - ▶ $\textcolor{red}{in} : \text{List}\{\text{Value}\} \rightarrow \text{ConfigItem}$: an input stream
 - ▶ $\textcolor{red}{out} : \text{List}\{\text{Value}\} \rightarrow \text{ConfigItem}$: an output stream
 - ▶ multiple k items for different threads
- ▶ Example: two threads

$$k(x := x + y) \ k(x := x - y) \ \text{env}(x \mapsto 1 ; y \mapsto 0)$$

Additional K Rules for IMP++

$$k\left(\frac{++x \curvearrowright K}{v+1} \text{ env}(x \mapsto \frac{v}{v+1}; ENV)\right)$$

$$k\left(\frac{read \curvearrowright K}{v} \text{ in}(\underline{v} \text{ ISTREAM})\right)$$

$$k\left(\frac{halt \curvearrowright K}{\cdot}\right)$$

$$k\left(\frac{spawn \ s \curvearrowright K}{\cdot} \frac{\cdot}{k(s)}\right)$$

$$\frac{k(\cdot)}{\cdot}$$

$$k\left(\frac{print \ v \curvearrowright K}{\cdot} \text{ out(OISTREAM } \frac{nil}{v}\text{)}\right)$$

- ▶ No existing rules need to be changed!

Example: The Thread Game

```
vars c ;
c := 1 ;
spawn(while true do c := c + c) ;
spawn(while true do c := c + c)
```

Reference

- ▶ G. Rosu and T. Serbanuta. [An overview of the K semantic framework](#). Journal of Logic and Algebraic Programming 79(6), 397-434, 2010
 - ▶ <https://www.sciencedirect.com/science/article/pii/S1567832610000160>
- ▶ K framework
 - ▶ <http://www.kframework.org> (we do not use K tools in this class)
- ▶ Next topic: model checking

Thank you!