Security II - Structural Operational Semantics

Stefano Calzavara

Università Ca' Foscari Venezia

April 9, 2020

Introduction

We now start our study of language-based security

- use of PL techniques to prove application security
- popular and rich research area, with many success stories

Fundamental questions:

1. syntax: what is a program?
2. semantics: what can a program do?
3. security: when is a program secure?

In this lecture, we will focus on the first two points.

Syntax of IMP

Three main syntactic categories: arithmetic expressions (a), boolean expressions (b) and commands (c)

$$
\begin{array}{lcl}
a & ::= & n \mid x \mid a + a \mid a - a \mid a * a \\
b & ::= & \textbf{true} \mid \textbf{false} \mid a \leq a \mid b \wedge b \mid b \vee b \mid \neg b \\
c & ::= & \textbf{skip} \mid x := a \mid c; c \mid \textbf{if } b \textbf{ then } c \textbf{ else } c \mid \textbf{while } b \textbf{ do } c
\end{array}
$$

Here, $n \in \mathbb{Z}$ ranges over integers and $x \in \mathcal{V}$ ranges over variables.

Example

$x := 3; \textbf{if } \neg(x \leq 5) \textbf{ then } y := x \textbf{ else } y := 0$

Configurations

Since IMP models imperative programs, its semantics depends upon and affects the state of memory

- a memory is a total function $\mu : \mathcal{V} \to \mathbb{Z}$ assigning a value (integer) to each variable
- we write $\mu[x \mapsto n]$ for the memory obtained from μ by rebinding the variable x to the value n
- a configuration is a pair $\langle c, \mu \rangle$

Programs start in an initial configuration and compute until termination.

Small-Step Semantics

A small-step semantics specifies the operation of a program c one step at a time:

- rules of the form $\langle c, \mu \rangle \rightarrow \langle c', \mu' \rangle$
- the rules are applied until we eventually hit a configuration of the form $\langle \textbf{skip}, \mu'' \rangle$ for some μ''
- if this is not possible, e.g., in the case of a non-terminating while loop, the computation goes on forever

We need auxiliary rules for arithmetic and boolean expressions as well.

Small-Step Semantics of Arithmetic Expressions

For arithmetic expressions, we use rules of the form $\langle a, \mu \rangle \to a'$

(A-Var)
$$\langle x, \mu \rangle \to \mu(x)$$

(A-Bin)
$$\langle n_1 \oplus n_2, \mu \rangle \to n_1 \oplus n_2$$

(A-Left)
$$\frac{\langle a_1, \mu \rangle \to a_1'}{\langle a_1 \oplus a_2, \mu \rangle \to a_1' \oplus a_2}$$

(A-Right)
$$\frac{\langle a_2, \mu \rangle \to a_2'}{\langle n_1 \oplus a_2, \mu \rangle \to n_1 \oplus a_2'}$$

Exercise: write down similar rules $\langle b, \mu \rangle \to b'$ for boolean expressions.

Arithmetic Expressions: Example

To exemplify, pick the memory $\mu = \{x \mapsto 5\}$ and the expression $3 * x + x$

$$\text{(A-LEFT)}\ \dfrac{\text{(A-RIGHT)}\ \dfrac{\text{(A-VAR)}\ \dfrac{}{\langle x, \mu \rangle \rightarrow 5}}{\langle 3 * x, \mu \rangle \rightarrow 3 * 5}}{\langle 3 * x + x, \mu \rangle \rightarrow 3 * 5 + x}$$

How many more steps are needed before eventually evaluating to 20?

Small-Step Semantics of Commands (1/3)

For commands, we use rules of the form $\langle c, \mu \rangle \to \langle c', \mu' \rangle$

(C-Asg1)
$$\frac{\langle a, \mu \rangle \to a'}{\langle x := a, \mu \rangle \to \langle x := a', \mu \rangle}$$

(C-Asg2)
$$\langle x := n, \mu \rangle \to \langle \textbf{skip}, \mu[x \mapsto n] \rangle$$

(C-Seq1)
$$\frac{\langle c_1, \mu \rangle \to \langle c_1', \mu' \rangle}{\langle c_1; c_2, \mu \rangle \to \langle c_1'; c_2, \mu' \rangle}$$

(C-Seq2)
$$\langle \textbf{skip}; c_2, \mu \rangle \to \langle c_2, \mu \rangle$$

Small-Step Semantics of Commands (2/3)

(C-Cond1)

$$\frac{\langle b, \mu \rangle \rightarrow b'}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \mu \rangle \rightarrow \langle \textbf{if } b' \textbf{ then } c_1 \textbf{ else } c_2, \mu \rangle}$$

(C-Cond2)

$$\langle \textbf{if true then } c_1 \textbf{ else } c_2, \mu \rangle \rightarrow \langle c_1, \mu \rangle$$

(C-Cond3)

$$\langle \textbf{if false then } c_1 \textbf{ else } c_2, \mu \rangle \rightarrow \langle c_2, \mu \rangle$$

Small-Step Semantics of Commands (3/3)

Finally, we define the semantics of while by loop unrolling

(C-While)
$$\langle \textbf{while } b \textbf{ do } c, \mu\rangle \rightarrow \langle \textbf{if } b \textbf{ then } (c; \textbf{while } b \textbf{ do } c) \textbf{ else skip}, \mu\rangle$$

Notice that this might lead to non-terminating computations!

Example

$$\begin{array}{rcl} \langle \textbf{while true do skip}, \mu\rangle & \rightarrow & \langle \textbf{if true then (skip; while true do skip) else skip}, \mu\rangle \\ & \rightarrow & \langle \textbf{skip; while true do skip}, \mu\rangle \\ & \rightarrow & \langle \textbf{while true do skip}, \mu\rangle \end{array}$$

Commands: Example

Evaluate $x := 3 + y; z := x$ in the memory $\mu = \{x \mapsto 0, y \mapsto 2, z \mapsto 0\}$

$$
\begin{aligned}
\langle x := 3 + y; z := x, \mu \rangle & \rightarrow \langle x := 3 + 2; z := x, \mu \rangle \\
& \rightarrow \langle x := 5; z := x, \mu \rangle \\
& \rightarrow \langle \mathbf{skip}; z := x, \{x \mapsto 5, y \mapsto 2, z \mapsto 0\} \rangle \\
& \rightarrow \langle z := x, \{x \mapsto 5, y \mapsto 2, z \mapsto 0\} \rangle \\
& \rightarrow \langle z := 5, \{x \mapsto 5, y \mapsto 2, z \mapsto 0\} \rangle \\
& \rightarrow \langle \mathbf{skip}, \{x \mapsto 5, y \mapsto 2, z \mapsto 5\} \rangle
\end{aligned}
$$

Big-Step Semantics

A big-step semantics specifies the operation of a program c in terms of the final result of its computation:

- rules of the form $\langle c, \mu \rangle \Downarrow \mu'$
- the rules are applied to build a proof tree, which directly yields the final memory μ'
- if this is not possible, e.g., in the case of a non-terminating while loop, no proof tree can be built!

We need auxiliary rules for arithmetic and boolean expressions as well.

Big-Step Semantics of Arithmetic Expressions

For arithmetic expressions, we use rules of the form $\langle a, \mu \rangle \Downarrow n$

$$\text{(A-INT)} \quad \langle n, \mu \rangle \Downarrow n$$

$$\text{(A-VAR)} \quad \langle x, \mu \rangle \Downarrow \mu(x)$$

$$\text{(A-BIN)} \quad \frac{\langle a_1, \mu \rangle \Downarrow n_1 \qquad \langle a_2, \mu \rangle \Downarrow n_2}{\langle a_1 \oplus a_2, \mu \rangle \Downarrow n_1 \oplus n_2}$$

Exercise: write down similar rules $\langle b, \mu \rangle \Downarrow b'$ for boolean expressions.

Big-Step Semantics of Commands (1/2)

For commands, we use rules of the form $\langle c, \mu \rangle \Downarrow \mu'$

$$\text{(C-Skip)} \quad \langle \textbf{skip}, \mu \rangle \Downarrow \mu$$

$$\text{(C-Asg)} \quad \frac{\langle a, \mu \rangle \Downarrow n}{\langle x := a, \mu \rangle \Downarrow \mu[x \mapsto n]}$$

$$\text{(C-Seq)} \quad \frac{\langle c_1, \mu \rangle \Downarrow \mu_1 \qquad \langle c_2, \mu_1 \rangle \Downarrow \mu_2}{\langle c_1; c_2, \mu \rangle \Downarrow \mu_2}$$

$$\text{(C-Cond1)} \quad \frac{\langle b, \mu \rangle \Downarrow \textbf{true} \qquad \langle c_1, \mu \rangle \Downarrow \mu_1}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \mu \rangle \Downarrow \mu_1}$$

$$\text{(C-Cond2)} \quad \frac{\langle b, \mu \rangle \Downarrow \textbf{false} \qquad \langle c_2, \mu \rangle \Downarrow \mu_2}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \mu \rangle \Downarrow \mu_2}$$

Big-Step Semantics of Commands (2/2)

Finally, we define the semantics of while by induction

$$\frac{\langle b, \mu \rangle \Downarrow \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \mu \rangle \Downarrow \mu}$$

$$\frac{\langle b, \mu \rangle \Downarrow \textbf{true} \qquad \langle c, \mu \rangle \Downarrow \mu' \qquad \langle \textbf{while } b \textbf{ do } c, \mu' \rangle \Downarrow \mu''}{\langle \textbf{while } b \textbf{ do } c, \mu \rangle \Downarrow \mu''}$$

Exercise: try to evaluate $\langle \textbf{while true do skip}, \mu \rangle$. What happens?

Commands: Example

Evaluate $x := 3 + y; z := x$ in the memory $\mu = \{x \mapsto 0, y \mapsto 2, z \mapsto 0\}$

$$\dfrac{\dfrac{\dfrac{\langle 3, \mu \rangle \Downarrow 3 \qquad \langle y, \mu \rangle \Downarrow 2}{\langle 3 + y, \mu \rangle \Downarrow 5}}{\langle x := 3 + y, \mu \rangle \Downarrow \mu[x \mapsto 5]} \qquad \dfrac{\langle x, \mu[x \mapsto 5] \rangle \Downarrow 5}{\langle z := x, \mu[x \mapsto 5] \rangle \Downarrow \{x \mapsto 5, y \mapsto 2, z \mapsto 5\}}}{\langle x := 3 + y; z := x, \mu \rangle \Downarrow \{x \mapsto 5, y \mapsto 2, z \mapsto 5\}}$$

Small-Step vs Big-Step Semantics

The following theorem links the two presented semantics

Theorem

For all c, μ, μ' we have $\langle c, \mu \rangle \rightarrow^* \langle \textbf{skip}, \mu' \rangle$ if and only if $\langle c, \mu \rangle \Downarrow \mu'$.

Notice that the theorem says nothing about diverging computations: the connection only holds true for terminating behaviours!

Small-Step vs Big-Step Semantics

Small-step semantics:

+ can model complex language features, like concurrency, divergence...
- lot of rules and work to prove properties

Big-step semantics:

+ very natural specification, similar to a recursive interpreter
+ easier to prove properties, since we have less rules
- all programs without final configurations (infinite loops, errors, stuck configurations) look the same

Extending IMP

Assume we change the syntax of IMP as follows:

$$
\begin{array}{lcl}
v & ::= & n \mid \mathbf{true} \mid \mathbf{false} \\
e & ::= & v \mid x \mid e \oplus e \mid e \leq e \mid e \wedge e \mid e \vee e \mid \neg e \\
c & ::= & \mathbf{skip} \mid x := e \mid c; c \mid \mathbf{if}\ e\ \mathbf{then}\ c\ \mathbf{else}\ c \mid \mathbf{while}\ e\ \mathbf{do}\ c
\end{array}
$$

This allows us to write programs that we could not write before!

Example

$x := \mathbf{true}; \mathbf{if}\ (x \vee \mathbf{false})\ \mathbf{then}\ y := 2\ \mathbf{else}\ y := 5$

Typed IMP

But we can also write programs that we don't like!

Example

$x := 4; \textbf{if } (x \lor \textbf{false}) \textbf{ then } y := 2 \textbf{ else } y := 5$

To solve these issues, we can use a type system

- we let $\mathcal{T} = \{\textbf{int}, \textbf{bool}\}$ stand for the set of types
- we let $\Gamma : \mathcal{V} \to \mathcal{T}$ represent a typing environment mapping variables to their expected type
- we define type rules to define acceptable programs

Type Rules for Expressions

For expressions, we use rules of the form $\Gamma \vdash e : t$

$$\Gamma \vdash n : \textbf{int} \qquad \Gamma \vdash \textbf{true} : \textbf{bool} \qquad \Gamma \vdash \textbf{false} : \textbf{bool} \qquad \Gamma \vdash x : \Gamma(x)$$

$$\frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 \oplus e_2 : \textbf{int}} \qquad \frac{\Gamma \vdash e_1 : \textbf{bool} \qquad \Gamma \vdash e_2 : \textbf{bool}}{\Gamma \vdash e_1 \land e_2 : \textbf{bool}}$$

$$\frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 \leq e_2 : \textbf{bool}}$$

Type Rules for Commands

For commands, we use rules of the form $\Gamma \vdash c$

$$\Gamma \vdash \textbf{skip} \qquad \frac{\Gamma \vdash x : t \qquad \Gamma \vdash e : t}{\Gamma \vdash x := e} \qquad \frac{\Gamma \vdash c_1 \qquad \Gamma \vdash c_2}{\Gamma \vdash c_1; c_2}$$

$$\frac{\Gamma \vdash e : \textbf{bool} \qquad \Gamma \vdash c_1 \qquad \Gamma \vdash c_2}{\Gamma \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2} \qquad \frac{\Gamma \vdash e : \textbf{bool} \qquad \Gamma \vdash c}{\Gamma \vdash \textbf{while } e \textbf{ do } c}$$

Typing Example

We show that $x := 4; \textbf{if } (x \vee \textbf{false}) \textbf{ then } y := 2 \textbf{ else } y := 5$ is ill-typed in the environment $\Gamma = \{x \mapsto \textbf{int}, y \mapsto \textbf{int}\}$

$$\dfrac{\dfrac{\Gamma \vdash x : \textbf{int} \qquad \Gamma \vdash 4 : \textbf{int}}{\Gamma \vdash x := 4} \qquad \dfrac{\dfrac{\Gamma \vdash x : \textbf{bool} \qquad \Gamma \vdash \textbf{false} : \textbf{bool}}{\Gamma \vdash x \vee \textbf{false} : \textbf{bool}} \qquad \dots \qquad \dots}{\Gamma \vdash \textbf{if } (x \vee \textbf{false}) \textbf{ then } y := 2 \textbf{ else } y := 5}}{\Gamma \vdash x := 4; \textbf{if } (x \vee \textbf{false}) \textbf{ then } y := 2 \textbf{ else } y := 5}$$

Observe in particular that x must be given two different types in the derivation, which is not possible in our type system.