# Systems of High Safety and Security

## Lecture 7 from 26.11.2025: Operational Semantics

Winter term 2025/26

Christoph Lüth

# Roadmap

# Semantics — what and why?

> *Semantics* (noun [uncountable]) 2. the meaning of words, phrases or systems
>
> — Oxford Learner's Dictionaries

Describes the meaning of a program in mathematical precise and unambiguous way:

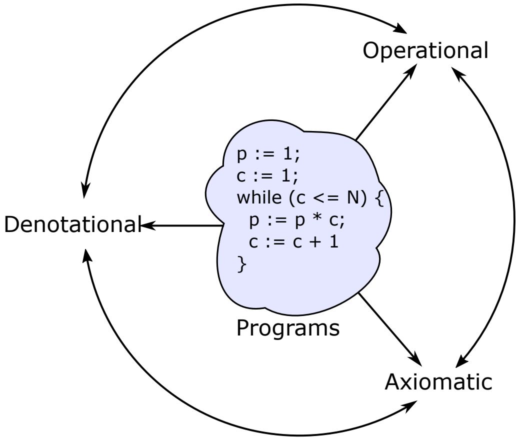
- Better compilers — independent of a particular compiler implementation.
- We will know when it should produce a result or not, and which situations to avoid.
- Lets us reason about program and compiler correctness.

# Semantics of Programming Languages

Historically, there are three ways to write down the semantics of a programming language:

- **Operational semantics** describes the meaning of programs by specifying how they **executes** on an abstract machine.
- **Denotational semantics** assigns a **meaning** to programs: a partial function on the system state.
- **Axiomatic semantics** gives a meaning to programs by giving proof rules. A prominent example of this is the Floyd-Hoare logic.

# A Tale of Three Semantics




- Each semantics is a view of the program.
- All semantics should be equivalent.
- In particular, for axiomatic semantics (Floyd-Hoare logic), rules should be correct.

# Our Wee Language

- We consider a **simple imperative language** (like C or Java).
- It has only integer types (no arrays, structs, pointers or references), no function calls, and no local variables.
- It is **Turing-complete** (we can write all programs).
- We give the programs in terms of an **abstract syntax**.

# Expressions

# Expressions

- Our simple language has the following expressions:

$$
\begin{aligned}
e ::= \mathbb{Z} \mid \mathbf{Idt} \mid \mathit{true} \mid \mathit{false} \\
\mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 / e_2 \\
\mid e_1 == e_2 \mid e_1 < e_2 \\
\mid\ !\, e \mid e_1 \,\&\&\, e_2 \mid e_1 \,||\, e_2
\end{aligned}
$$

- This **abstract** (not concrete) syntax, it lacks parentheses and precedence etc.
- Expressions (terms) are represented as **trees**.

# Structural Operational Semantics

- Defined inductively by **rules** of the form

$$\frac{\langle t', a' \rangle \to b' \qquad \phi(a', t, t')}{\langle t, a \rangle \to b}$$

- $t$ is a tree of depth 1 (one top symbol, all children are distinct variables)
- $a$ is input data (e.g. the state), $b$ return data (e.g. a value)
- Applying the rule corresponds to a **state transition** of the abstract machine

# States

- States are **finite partial maps** represented by **right-unique** relations

$$f : X \rightharpoonup A \subseteq X \times A \text{ such that } \forall x, a, b.\, (x, a) \in f \wedge (x, b) \in f \Longrightarrow a = b$$

- State for our language: $\Sigma \stackrel{def}{=} \mathbf{Idt} \rightharpoonup \mathbb{Z}$ (identifiers mapped to integers)
- Notation:
  - $\langle x \mapsto 5, y \mapsto 7, z \mapsto 10 \rangle$ für $\{(x, 5), (y, 7), (z, 10)\}$
  - $f(x)$ for the value of $x$ in $f$ (*lookup*)
  - $f(x) = \bot$ if $x$ not in $f$ (*undefined*) and $\mathrm{def}(f(x))$ für $(x, y) \in f$ (*defined*)
  - $f \setminus x$ to **remove** $x$ from $f$
  - $f[x \mapsto n]$ to **update** $f$ at $x$ with the value $n$.

# Rules of the Operational Semantics

- An expression $e$ with a state $\sigma$ evaluates to an integer $n \in \mathbb{Z}$ or a boolean $b \in \mathbb{B}$:

$$e ::= \mathbb{Z} \mid \mathbf{Idt} \mid \mathit{true} \mid \mathit{false} \mid \ldots \qquad \langle e, \sigma \rangle \rightarrow_{Exp} n \mid b$$

- **Rules:**

$$\frac{i \in \mathbb{Z}}{\langle i, \sigma \rangle \rightarrow_{Exp} i} \qquad \frac{b \in \mathbb{B}}{\langle b, \sigma \rangle \rightarrow_{Exp} b} \qquad \frac{x \in \mathbf{Idt}, x \in \mathrm{dom}(\sigma), \sigma(x) = v}{\langle x, \sigma \rangle \rightarrow_{Exp} v}$$

# Operational Semantics: Arithmetic Expressions

- Expressions:

$$e ::= \ldots \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 / e_2 \mid \ldots \qquad \langle e, \sigma \rangle \rightarrow_{Exp} n$$

- **Rules:**

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} n_1 \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} n_2 \qquad n_i \in \mathbb{Z}}{\langle e_1 + e_2, \sigma \rangle \rightarrow_{Exp} n_1 + n_2}$$

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} n_1 \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} n_2 \qquad n_i \in \mathbb{Z}}{\langle e_1 - e_2, \sigma \rangle \rightarrow_{Exp} n_1 - n_2}$$

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} n_1 \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} n_2 \qquad n_i \in \mathbb{Z}}{\langle e_1 * e_2, \sigma \rangle \rightarrow_{Exp} n_1 \cdot n_2}$$

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} n_1 \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} n_2 \qquad n_i \in \mathbb{Z}, n_2 \neq 0}{\langle e_1 / e_2, \sigma \rangle \rightarrow_{Exp} n_1 \div n_2}$$

# Operational Semantics: Predicates

- Expressions:

$$e ::= \ldots \mid e_1 == e_2 \mid e_1 < e_2 \mid \ldots \qquad \langle e, \sigma \rangle \rightarrow_{Exp} b$$

- **Rules:**

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} n_1 \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} n_2 \qquad n_i \in \mathbb{Z}, n_1 = n_2}{\langle e_1 == e_2, \sigma \rangle \rightarrow_{Exp} \mathit{true}}$$

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} n_1 \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} n_2 \qquad n_i \in \mathbb{Z}, n_1 \neq n_2}{\langle e_1 == e_2, \sigma \rangle \rightarrow_{Exp} \mathit{false}}$$

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} n_1 \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} n_2 \qquad n_i \in \mathbb{Z}, n_1 < n_2}{\langle e_1 < e_2, \sigma \rangle \rightarrow_{Exp} \mathit{true}}$$

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} n_1 \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} n_2 \qquad n_i \in \mathbb{Z}, n_1 \geq n_2}{\langle e_1 < e_2, \sigma \rangle \rightarrow_{Exp} \mathit{false}}$$

# Operational Semantics: Connectives

- Expressions:

$$e ::= \ldots \mid !\, e \mid e_1 \,\&\&\, e_2 \mid e_1 \,||\, e_2 \qquad \langle e, \sigma \rangle \rightarrow_{Exp} b$$

- **Rules:**

$$\frac{\langle e, \sigma \rangle \rightarrow_{Exp} \mathit{true}}{\langle !\, e, \sigma \rangle \rightarrow_{Exp} \mathit{false}} \qquad \frac{\langle e, \sigma \rangle \rightarrow_{Exp} \mathit{false}}{\langle !\, e, \sigma \rangle \rightarrow_{Exp} \mathit{true}}$$

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} \mathit{false}}{\langle e_1 \,\&\&\, e_2, \sigma \rangle \rightarrow_{Exp} \mathit{false}} \qquad \frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} \mathit{true} \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} t}{\langle e_1 \,\&\&\, e_2, \sigma \rangle \rightarrow_{Exp} t}$$

$$\frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} \mathit{true}}{\langle e_1 \,||\, e_2, \sigma \rangle \rightarrow_{Exp} \mathit{true}} \qquad \frac{\langle e_1, \sigma \rangle \rightarrow_{Exp} \mathit{false} \qquad \langle e_2, \sigma \rangle \rightarrow_{Exp} t}{\langle e_1 \,||\, e_2, \sigma \rangle \rightarrow_{Exp} t}$$

# Questions

- Does evaluation always terminate?
- When not?
- Why not?
- Order of operands?
- Strictness?

# Statements

# Simple Statements

- **Core language**:
  - Assignment
  - Sequencing and empty statement — **sequences** of expressions
  - Case distinction
  - Iteration (while)
- Makes it **Turing-equivalent**
- Some languages view expressions (with side effects) as statements — and assignments as expressions

# Abstract Syntax

- Statements:

$$
\begin{aligned}
c ::= &\ \mathbf{Idt} := \mathbf{Exp} \\
&\mid c_1 ; c_2 \\
&\mid \mathbf{nil} \\
&\mid \mathbf{if}\ (e)\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \\
&\mid \mathbf{while}\ (e)\ c
\end{aligned}
$$

- Operational semantics: $\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$

# Operational Semantics: Statements

- Rules:

$$\frac{\langle e, \sigma \rangle \rightarrow_{Exp} n \qquad n \in \mathbb{Z}}{\langle x := e, \sigma \rangle \rightarrow_{Stmt} \sigma[x \mapsto n]} \qquad \frac{}{\langle \mathbf{nil}, \sigma \rangle \rightarrow_{Stmt} \sigma}$$

$$\frac{\langle c_1, \sigma \rangle \rightarrow_{Stmt} \sigma' \qquad \langle c_2, \sigma' \rangle \rightarrow_{Stmt} \sigma''}{\langle c_1; c_2, \sigma \rangle \rightarrow_{Stmt} \sigma''}$$

- Example: $\sigma \stackrel{def}{=} \langle\rangle$

```
x := 6;
y := 4+ x;
```

# Operational Semantics: Statements

- Rules:

$$\frac{\langle b, \sigma\rangle \rightarrow_{Exp} \mathit{true} \qquad \langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma'}{\langle \textbf{if } (b) \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'} \qquad \frac{\langle b, \sigma\rangle \rightarrow_{Exp} \mathit{false} \qquad \langle c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'}{\langle \textbf{if } (b) \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'}$$

- Example: $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 10\rangle$

```
if (x != 0) {
  y := y/x;
  } else {
  y := 0;
  }
```

# Operational Semantics: Statements

- Rules:

$$\frac{\langle b, \sigma \rangle \rightarrow_{Exp} \mathit{false}}{\langle \textbf{while}\ (b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma}$$

$$\frac{\langle b, \sigma \rangle \rightarrow_{Exp} \mathit{true} \qquad \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \qquad \langle \textbf{while}\ (b)\ c, \sigma' \rangle \rightarrow_{Stmt} \sigma''}{\langle \textbf{while}\ (b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma''}$$

- Example: $\sigma \stackrel{def}{=} \langle x \mapsto 3 \rangle$

```
f := 1;
while (x > 0) {
  f := f* x;
  x := x-1;
}
```

# Conclusions

- **Operational semantics** describe the stepwise **evaluation** of programs by structured derivation rules.
- It gives a precise notion of **program execution** (small-step semantics), but not so much about the **meaning** of the program (big-step semantics).
- It can be used to write and in particular **verify compilers** — see next lecture.