Operational Semantics

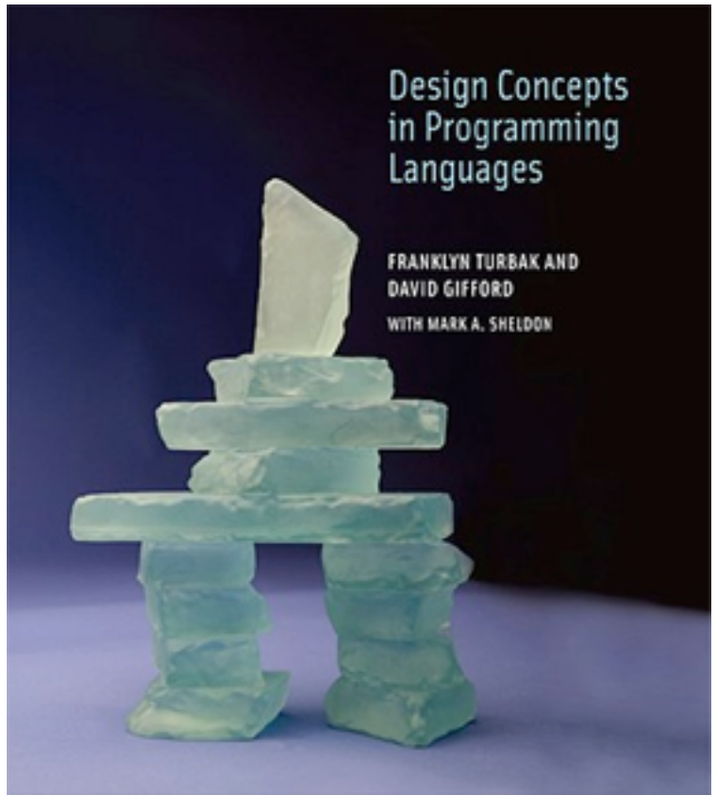
A4M36TPJ, 2015/2016

Overview

- Introduction
- Small-step Operational Semantics (SOS)
- Big-step Operational Semantics (BOS)

References

- Franklyn Turbak, David Gifford, Mark A. Sheldon: *Design Concepts in Programming Languages*, The MIT Press, 2008, ISBN: 978-0262201759



- https://edux.feld.cvut.cz/courses/A4M36TPJ

Programming Languages

1. Syntax — the *form*...

2. Semantics — the *meaning*...

3. Pragmatics — the *implementation*...

...of programming languages

Small Example

- The arithmetic language syntax:

$$E ::= N \mid E_1 + E_2$$

$$where\ N \in \{0, 1, 2, ...\}$$

Arithmetic Language Semantics

- The semantics is really simple:

$$N_1 + N_2 \Rightarrow N'$$

$$\text{where } N' \text{ is the sum}$$

$$\text{of the numerals } N_1 \text{ and } N_2$$

Note: This a term rewriting system!

Example

$$(1+2)+(3+4)$$

Solution 1:

$$(1+2)+(3+4) \Rightarrow 3+(3+4) \Rightarrow 3+7 \Rightarrow 10$$

Solution 2:

$$(1+2)+(3+4) \Rightarrow (1+2)+7 \Rightarrow 3+7 \Rightarrow 10$$

Why is it possible?

- The set of semantics rules is *confluent*.
- More on this later in 4th lecture.

How to demand left-to-right computation?

- We need to add new rules (called *inference rules*).

$$\frac{E_1 \Rightarrow E_1'}{E_1 + E_2 \Rightarrow E_1' + E_2} \tag{1}$$

$$\frac{E_2 \Rightarrow E_2'}{N + E_2 \Rightarrow N + E_2'} \tag{2}$$

Is there any other operational semantics for Arithmetic language?

- YES!
- Consider the stack semantics.
- We want to see how the computation is done on a stack machine.

Stack Semantics for Arithmetic Language

$$< s, N :: c > \Rightarrow < N :: s, c > \tag{1}$$

$$< s, E_1 + E_2 :: c > \Rightarrow < s, E_1 :: E_2 :: add :: c > \tag{2}$$

$$< N1 :: N2 :: s, add \ :: c > \Rightarrow < add :: N1 :: N2 :: s, c > \tag{3}$$

$$\begin{gathered} < add :: N_2 :: N_1 :: s, c > \Rightarrow < N' :: s, c > \\ \text{where } N' \text{ is the sum of } N_1 \text{ and } N_2 \end{gathered} \tag{4}$$

Initial and Final state: $S =< nil, E :: nil >, F =< N :: nil, nil >$

Note: This form of operational semantics is called *state-transitional semantics*.

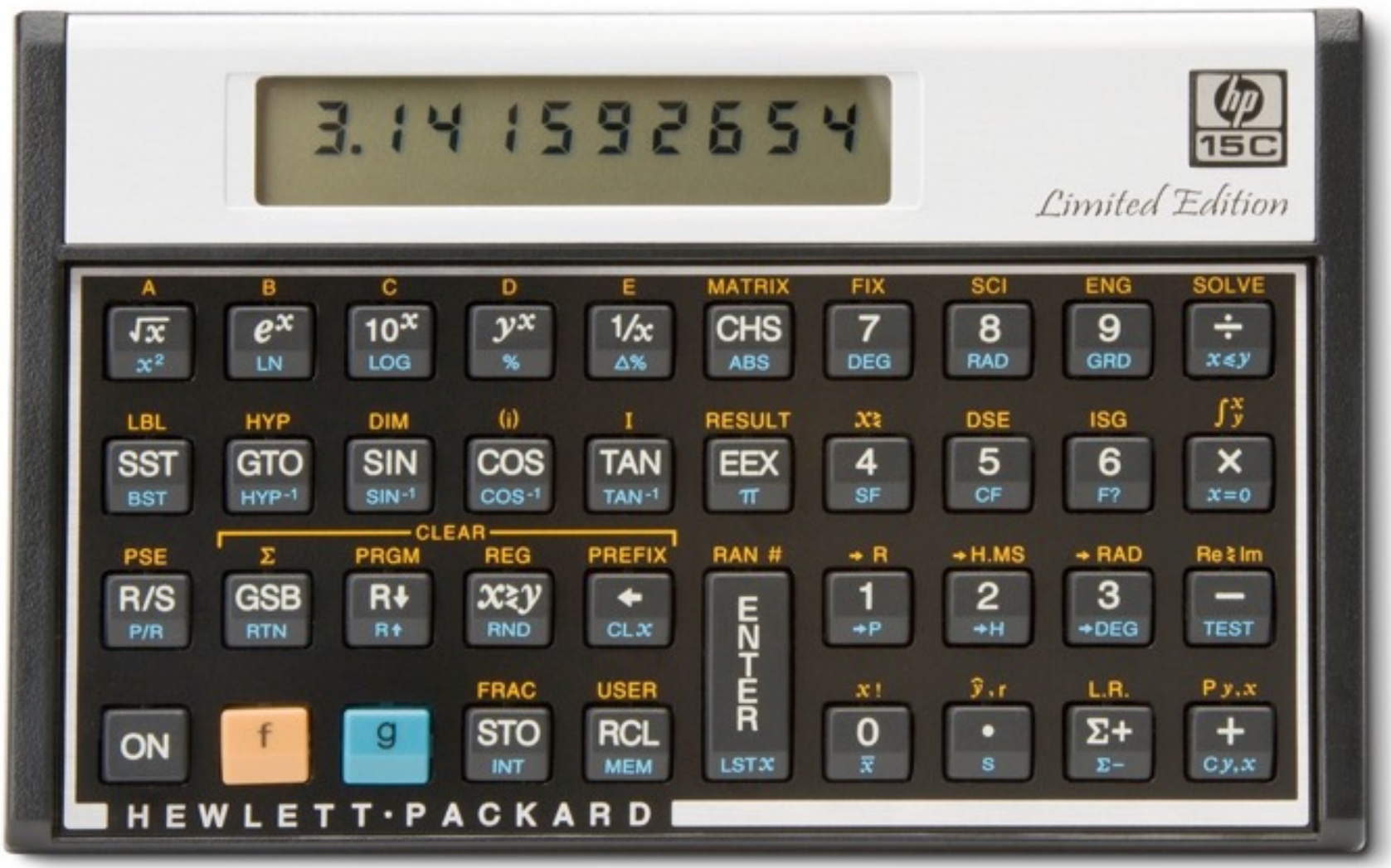
Abstract Machine

1. Code component
2. State component

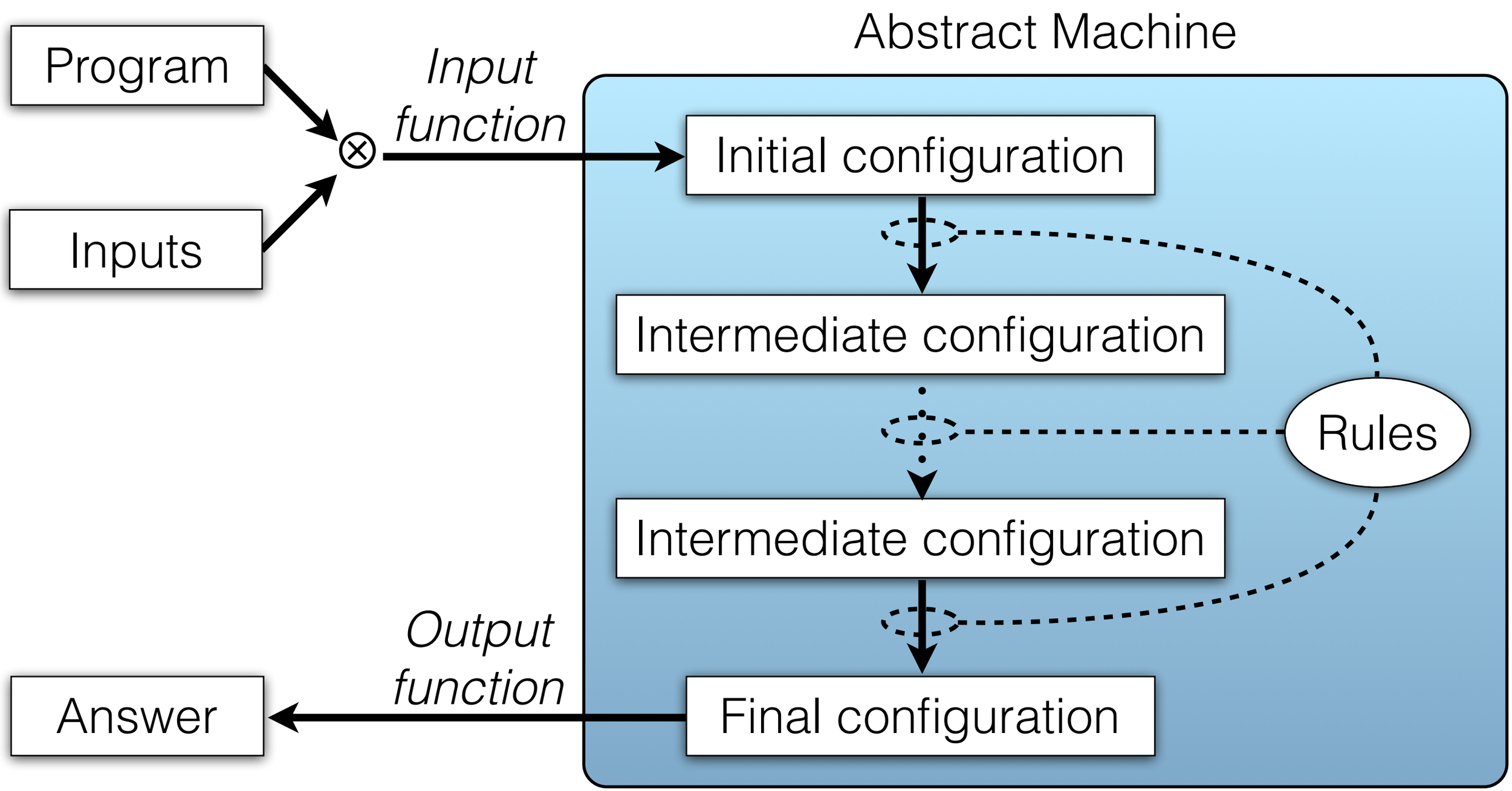
Operational Execution

- <u>Program</u> ⊗ <u>Inputs</u> → Input function
- Initial configuration
- *(Intermediate configurations...)*
- Final configuration
- Output function → <u>Answer</u>

Configuration



The OS "game board"



Example Language - Syntax

$$
\begin{aligned}
Expr ::= \; & Num \; | \\
& \triangle Expr \; | \\
& Expr \odot Expr
\end{aligned}
$$

Num is a predefined set of integer numbers (a.k.a. Z).

Expressions

$$Expr = \{\ldots, 0, 1, 2, \ldots, \triangle 0, \triangle 1, \triangle 2, \ldots, 0 \odot 0, 0 \odot 1, 0 \odot 2, \ldots, (\triangle 0) \odot 0, (\triangle 0) \odot 1, (\triangle 0) \odot 2, 0 \odot (\triangle 0), 0 \odot (\triangle 1), 0 \odot (\triangle 2), \triangle(0 \odot 0), \triangle(0 \odot 1), \triangle(0 \odot 2), \ldots\}$$

Small-step Operational Semantics

$n, n' \in Num$

$$\frac{}{\triangle n \Rightarrow -n}$$

$$\frac{}{n \odot n' \Rightarrow n + n'}$$

$e, e_1, e_2, e' \in Expr$

$$\frac{e \Rightarrow e'}{\triangle e \Rightarrow \triangle e'}$$

$$\frac{e_1 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e' \odot e_2}$$

$$\frac{e_2 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e_1 \odot e'}$$