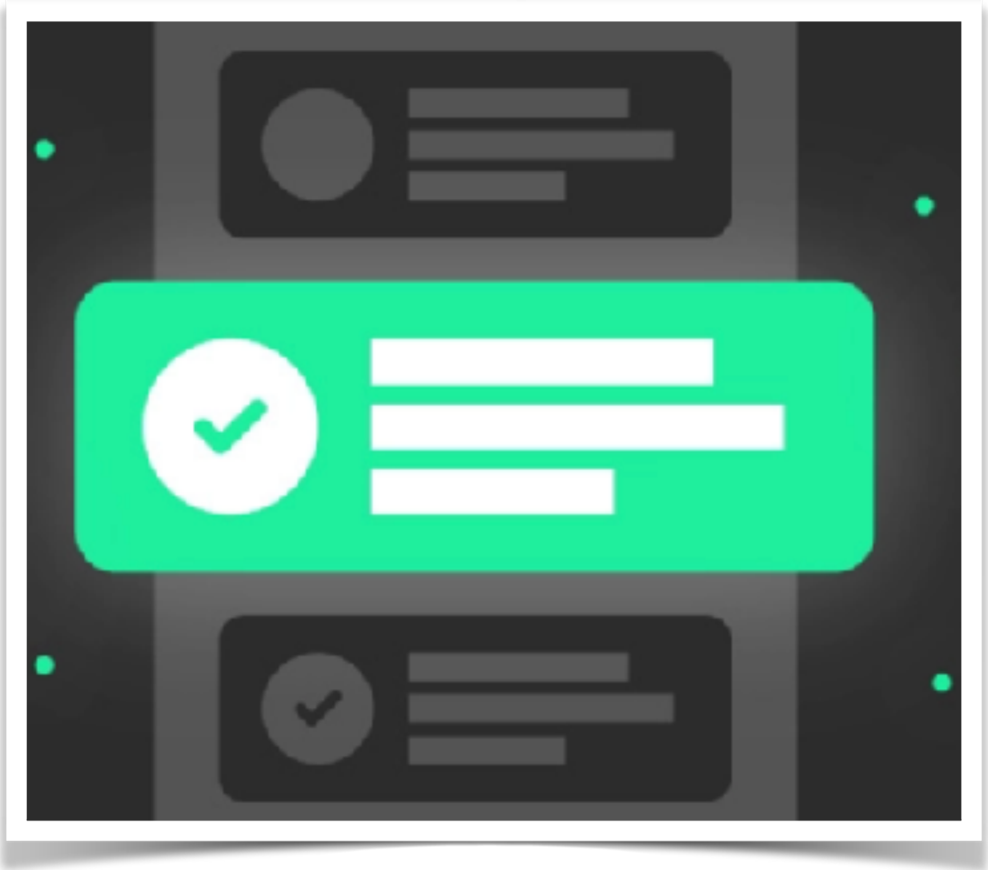


http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/start

PSC 2022/23 (375AA, 9CFU)

Principles for Software Composition

Roberto Bruni
http://www.di.unipi.it/~bruni/

02 - Preliminaries

From syntax to semantics

Programming languages

When we define a programming language, we fix its:

1. syntax	well-formed programs, exclude nonsense
2. types	reduce allowed programs, exclude common mistakes
3. pragmatics	how to use constructs and features
4. (semantics)	the meaning of (well-typed) programs

Formal syntax

The syntax of a formal language rigorously defines

1. the alphabet

   which symbols can be used

2. the grammatical structure of programs

   which sequences of symbols are valid,
   which sequences should be discarded

Standard ways for defining syntax are, e.g.
regular expressions, context-free grammars, BNF notation, syntax diagrams,…

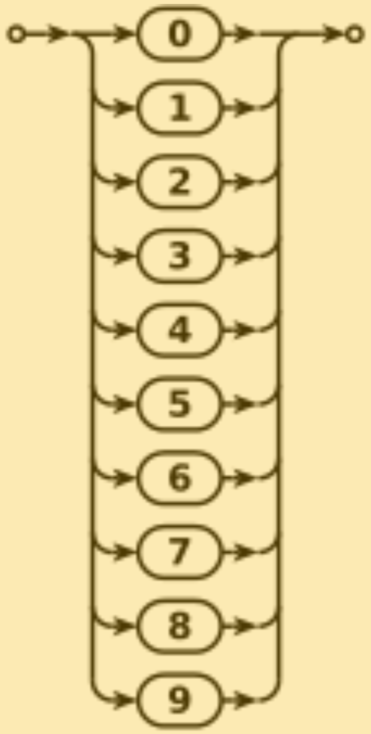
Example

A BNF grammar and its corresponding syntax diagram

```
<numeral> ::= <digit> | <numeral> <digit>
<digit>   ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

numeral:

digit:



Type systems

Type systems can be used to

1. limit the occurrence of errors
2. allow compiler optimisation
3. reduce the presence of bugs
4. discourage programming malpractices

different type systems
can be defined
for the same language!

Type systems are often presented as logic rules

Example

```
…
bool b := true;
while (b <= (b && i)) do {
    i := i-1;
}
```

Benefits of formalisation

Standardisation of the language

programmers write syntactically correct programs
implementors write correct parsers

Formal analysis of language properties

ambiguity, expressiveness,
recognizability, comparability

Automatic implementation of compiler's front-end

yacc, Bison, xtext, …

Exercise

Take the alphabet $A \triangleq \{\,(\,,\,)\,\}$

Define the grammar for strings of balanced parentheses

$\mathsf{S} ::= \quad ?$

Pragmatics

Programmers should understand the code they type

Every language manual also contains

1. natural language descriptions of the various constructs
2. sample code fragments and usage patterns
3. examples of malpractices

We call them pragmatics

1. how to exploit the various features
2. how compilers should be designed
3. which auxiliary tools are available

Is it enough?

Natural language descriptions should be

1. as much precise as possible
2. understandable
3. unambiguous but not pedantic

Still …

1. it is difficult (nearly impossible) to cover all cases
2. many points will remain open to different interpretations
3. inconsistencies can arise
4. good practices do not eliminate problems (hide them)

Some issues

How to prove conformance to some specification?

How to prove absence of problems?

How to produce reliable code?

How to prove vendors' compliance?

How to prove correctness of an implementation?

How to define the correct outcomes of test cases?

How to early detect ambiguities, anomalies, inconsistencies?

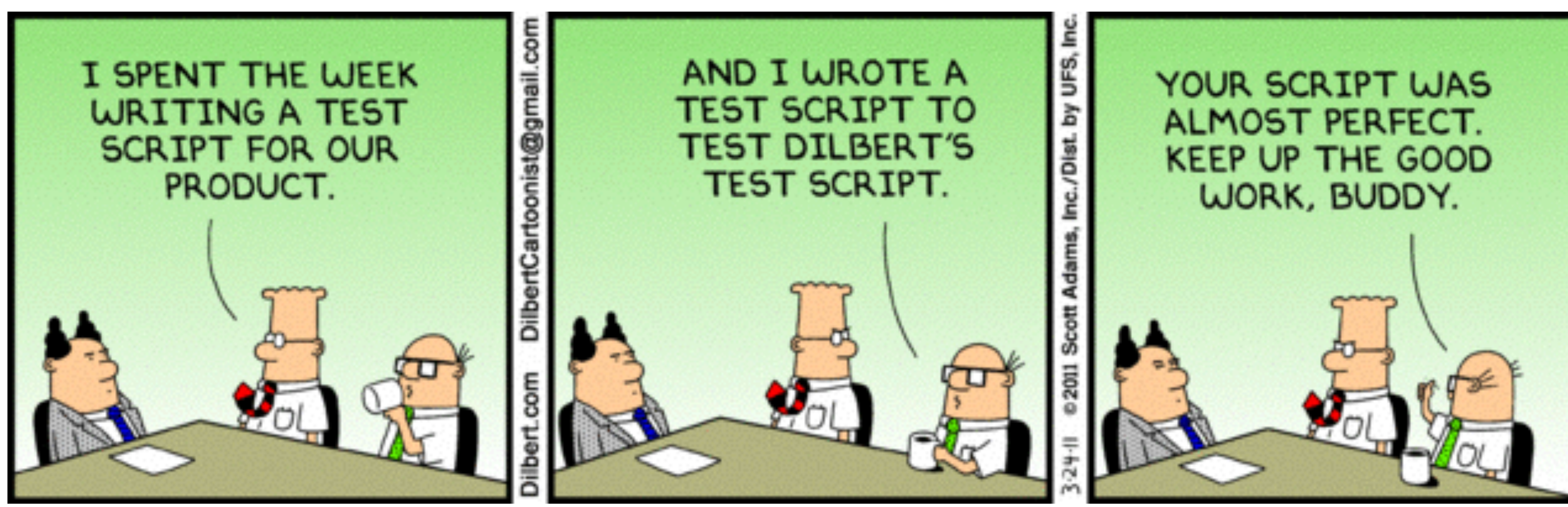
How to expose weaknesses?

…

Best practices

Code reviews

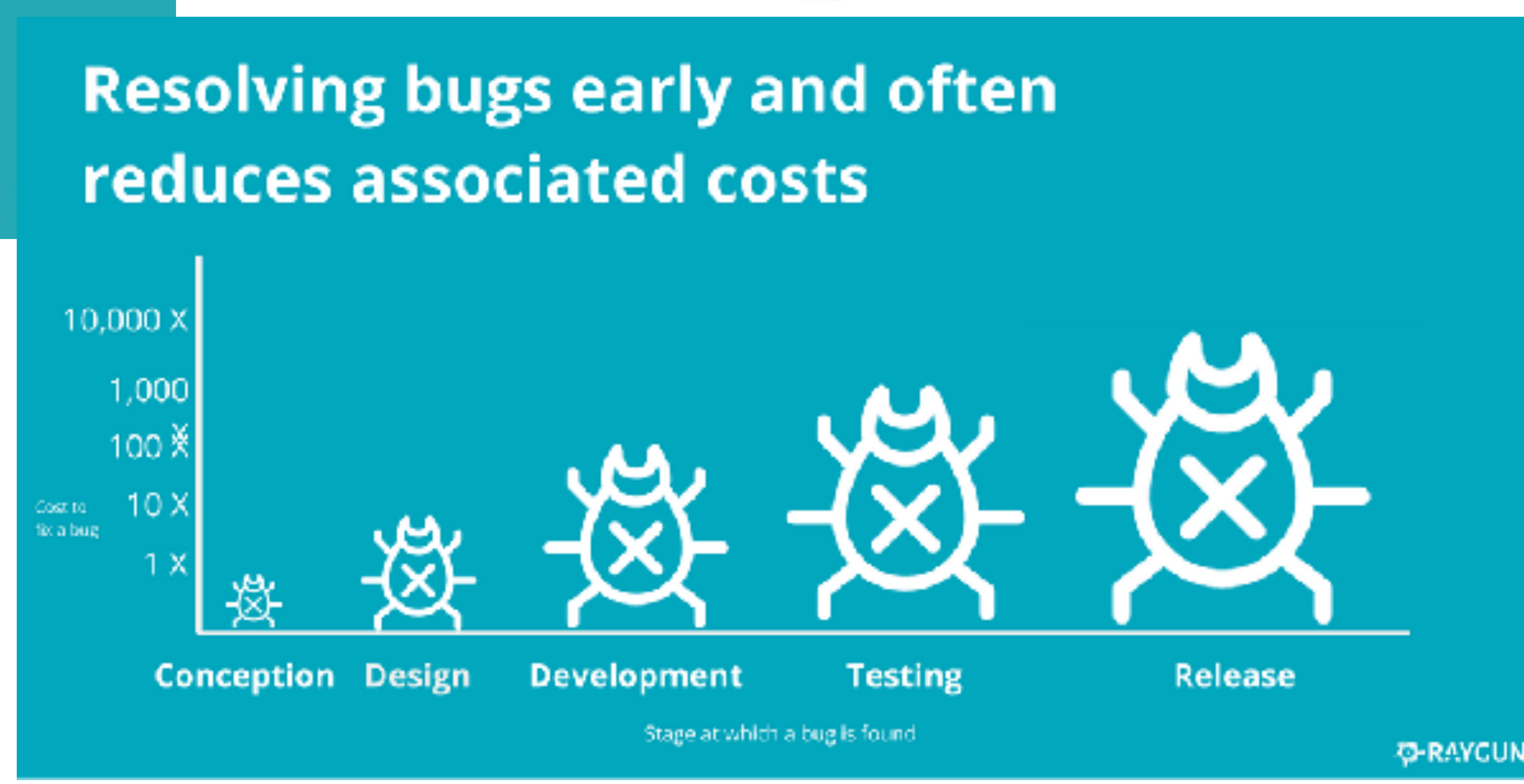
Test-driven development



Still…


Cost of software fails in 2017
606 fails
from 314 companies
US$1.7 trillion
in financial losses
3.6 billion
people affected
268 years
lost to downtime


IT WORKS
on my machine


Resolving bugs early and often reduces associated costs
10,000 X
1,000
100 X
10 X
1 X
Conception
Design
Development
Testing
Release
Stage at which a bug is found
RAYGUN

Semantics

The word ***semantics*** was introduced in 1900 as

*the study of **how words change their meanings*** (M. Bréal)

Ironically its meaning has now changed to

the study of the attachment between the sentences of a language (written, spoken or formal) and their meanings

In Computer Science, it is concerned with

*the study of the **meaning of (well-typed) programs***

Formal semantics assigns rigorous non-ambiguous meaning:
it tells programmers the meaning of the code they type
(at some level of abstraction)

Someone will always say

(my) implementation is the semantics of the language

correct by definition

machine-independent?
portability?
how long it will last?
useful abstraction for other programmers?
how to reason on it?
what about competitors?
…

Exercise

1. DO NOT ask questions

2. open your favourite text editor

3. implement as fast as possible the following primitive, by writing the corresponding code in your favourite imperative programming language

repeat *c* **until** *b*

4. DO NOT show your solution to others

5. send your code to bruni@di.unipi.it with subject “**I did it**”

Benefits of formalisation

Standardisation of the reference model of the language

official, machine-independent
a mental model for programmers
a benchmark for implementors

Formal analysis of language properties

subtleties, expressiveness, type safety,
program compliance, subject reduction

Automatic implementation of compiler's back-end

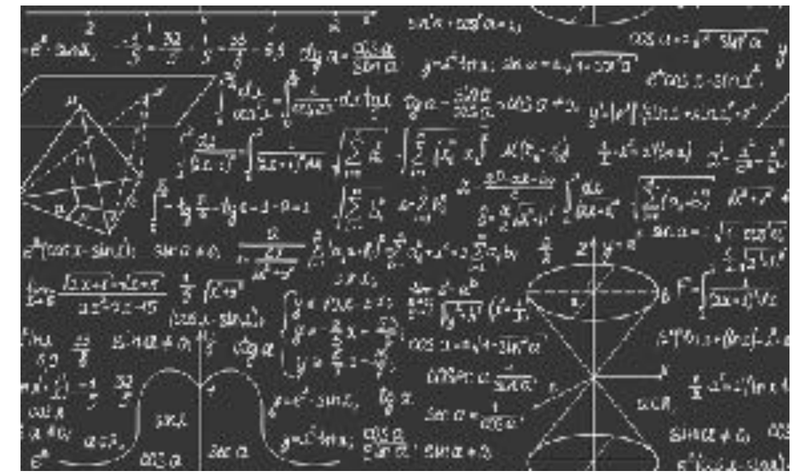
prototypical interpreter for experimentation

Still…

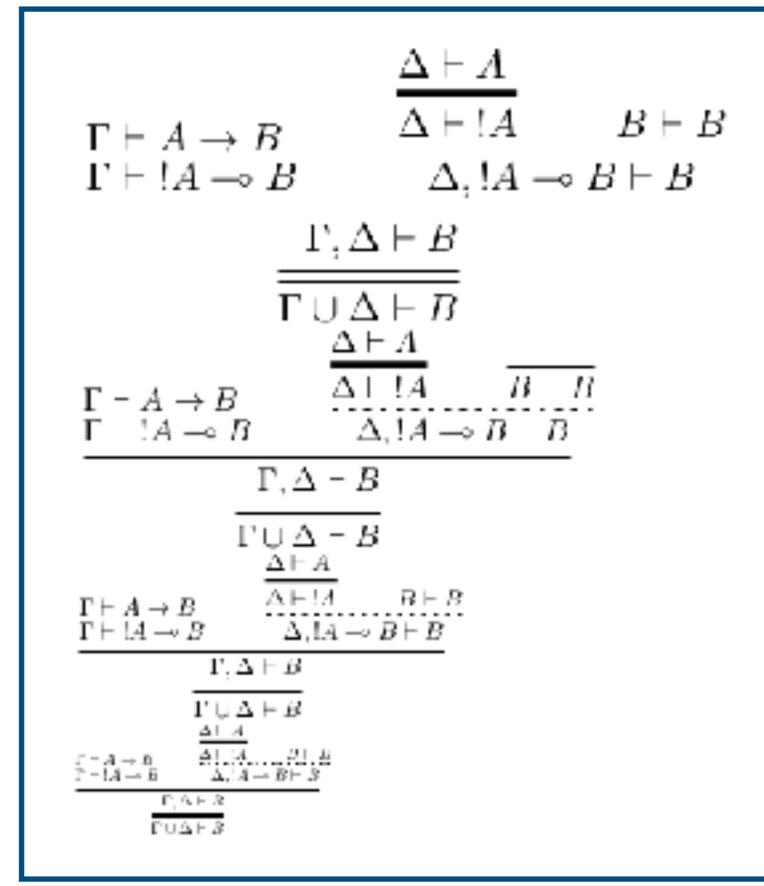
semantics is harder to formalize than syntax!

different methods

heavy math and logic involved



don't make me waste my time, I want to code



and then...

SOFTWARE BUGS IN HISTORY

The Ariane 5 Disaster

SOFTWARE BUGS IN HISTORY

Mars Climate Orbiter Disassembly

SOFTWARE BUGS IN HISTORY

Therac-25

SOFTWARE BUGS IN HISTORY

Losing $460m in 45 minutes

and then...

Heathrow Airport has apologised for disruption after the west London hub was hit by "technical issues".

One passenger said the situation was "utter chaos" after a problem with the airport's IT system saw staff called in to help passengers get to gates on the second day of the half-term weekend.

Heathrow Airport @HeathrowAirport · 16 feb 2020

Today's technical issue has now been resolved and Heathrow's systems are returning to normal. We apologise for the inconvenience caused. Our teams will continue to monitor our systems and be on hand to provide assistance to passengers as we work to resume our regular operations.

29 36 97

Risposte

[redacted] · 16 feb 2020

In risposta a @HeathrowAirport

Someone turned the system off and back on again?

1

Semantics

Different approaches

Roughly, semantics definition methods fall into three groups

1. Operational	it is of interest **how** the effect of the computation is achieved
2. Denotational	only the **effect** is of interest, not how it is obtained
3. Axiomatic	the focus is on valid **assertions** about the computation

Operational semantics

as opposed to a physical existing device

Idea: define some kind of abstract machine and describe the meaning of a program in terms of the steps or instructions that this machine executes to perform the task

the emphasis is on **states** and **state transformations**

Rationale:
explain computations

$$s_0 \to s_1 \to s_2 \to \cdots \to s_n \to r$$

Founding fathers

['70] small-step: semantics of LISP by John McCarthy (1960) and of Algol 68 (1975)

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge, Mass.

April 1960

['80] SOS approach: Gordon Plotkin introduced the structural (syntax-oriented and inductively defined) operational semantics in 1981 (one of the most cited technical reports in computer science, published in a journal only more than 20 years later).

$$\frac{\langle e_0, \sigma \rangle \longrightarrow \langle e_0', \sigma \rangle}{\langle e_0 + e_1, \sigma \rangle \longrightarrow \langle e_0' + e_1, \sigma \rangle}$$

['90] big-step: Gilles Kahn introduced the natural semantics in 1987, where the result is computed in a single step

$$s_0 \rightarrow r$$

Overview

Nowadays the transition relation is typically defined inductively, by axioms and inference rules according to the syntax of the program (SOS style).

Advantages:
immediate translation to Horn clauses in logic programming;
prototype Prolog interpreter (almost) for free;
strong connections to the syntax of the language;
rules for different constructs are neatly separated;
useful to detect underspecified behaviours;
involved mathematics is usually not much complicated;
SOS descriptions are easy to read, even for non-specialists;
could appear in any manual (but usually it won’t)

Denotational semantics

Idea: the meaning of a program is some mathematical object (e.g. a function from input to output) and the steps taken to calculate the result are unimportant

$$[\![\cdot]\!] : \mathsf{Programs} \rightarrow \mathsf{Domains}$$

Rationale: functions are independent of their means of computation and hence are simpler than the step-by-step sequence of operations of operational semantics

Founding fathers

['60/'70]: Christopher Strachey and Dana Scott

OUTLINE OF A MATHEMATICAL
THEORY OF COMPUTATION

by

Dana Scott
Princeton University

Compositionally principle: the semantics takes the form of a function that assigns an element of some mathematical domain to each individual construct in such a way that

the meaning of a composite construct does not depend on the particular form of the constituent constructs, but only on their meanings

Overview

Advantages:
mathematically elegant;
useful to detect underspecified behaviours;
can be used to derive prototype implementations;
has served as inspiration for many programming languages;
difficult to apply to concurrent, interactive systems

Axiomatic semantics

Idea: describe the constructs in a programming language by providing logical axioms that are satisfied by these constructs

Rationale: prove the correctness of a program with respect to a given specification

$$\vdash \{P\}\ c\ \{Q\}$$

Founding fathers

['60]: Robert W. Floyd (1967) and Tony Hoare (1969)

Robert W. Floyd

ASSIGNING MEANINGS TO PROGRAMS[1]

An Axiomatic Basis for Computer Programming

C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

Hoare logic: a statement is accompanied by a precondition (the state before the execution) and a postcondition (after the execution)

> *the meaning of a program is a logical proposition that states some property of the output whenever some properties of the input are met*

Overview

Advantages:
emphasis on proof correctness from the very start;
strikingly elegant proof systems;
can be used to prove absence of bugs;
difficult to apply to concurrent, interactive systems

Make love not war

Different semantics are often seen in opposition one each other, but this should not be the case!
We would gain much more from their combination!

This course

We focus on operational and denotational semantics

We will present the fundamental ideas and methods behind these approaches and stress their relationship, by proving some relevant correspondence theorems.

$s_0 \to r$

$[\![\cdot]\!]$: Programs $\to$ Domains

A taste of semantics methods

A simple language

Informal syntax of numerical expressions

- any numeral N is an expression;
- if E_1 and E_2 are expressions, then $E_1 \oplus E_2$ is an expression;
- if E_1 and E_2 are expressions, then $E_1 \otimes E_2$ is an expression.

N numerals vs numbers n

syntax for writing numbers

5

1 0 1

five

$5 \in \mathbb{N}$ mathematical objects, concepts

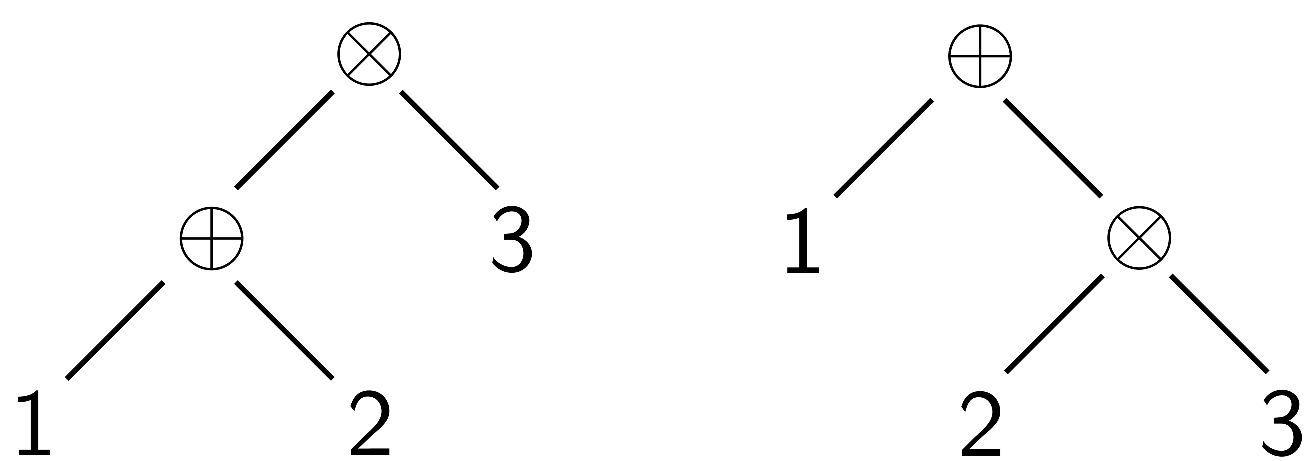
Formal syntax

$$E ::= N \mid E \oplus E \mid E \otimes E$$

$\oplus 3 \otimes 4$ not a well-formed numerical expression

a string

$1 \oplus 2 \otimes 3$



two abstract syntax trees

we use brackets

$1 \oplus (2 \otimes 3)$

or fix operators precedence to solve ambiguities

Assign meaning

$$E ::= N \mid E \oplus E \mid E \otimes E$$

this is just syntax!

is N necessarily a number?

is $\oplus$ necessarily the arithmetic sum?

is $\otimes$ necessarily the arithmetic product?

maybe we are speaking about
matrices with addition and multiplication

or sets with union and intersection

or strings with concatenation and least common prefix

or trees with branching and merging

Informal semantics

Informal semantics of numerical expressions

- a numeral N evaluates to its corresponding number n;
- to evaluate an expression of the form $E_1 \oplus E_2$ we evaluate E_1 and E_2 and sum their values;
- to evaluate an expression of the form $E_1 \otimes E_2$ we evaluate E_1 and E_2 and multiply their values.

Three rules are enough to determine the value of any well-formed expression, no matter how large

Note that we are not telling the order in which arguments are evaluated: is it important?

Pragmatics

We can provide some examples

- 2 evaluates to 2;
- $(1 \oplus 2) \otimes 3$ evaluates to 9;
- $(1 \oplus 2) \otimes (3 \oplus 4)$ evaluates to 21

Small-step semantics

Runtime numerical expressions

$$\mathsf{E} ::= n \mid \mathsf{N} \mid \mathsf{E} \oplus \mathsf{E} \mid \mathsf{E} \otimes \mathsf{E}$$

the state of the abstract machine can mix intermediate results with expressions

a step	$\mathsf{E}_0 \to \mathsf{E}_1$
an evaluation	$\mathsf{E}_0 \to \mathsf{E}_1 \to \mathsf{E}_2 \to \cdots \to \mathsf{E}_k \to n$
also written	$\mathsf{E}_0 \to^* n$
we also expect	$n \not\to$

How to define the transition relation?

Inference rules

Inference rules $$(rule\ name)\frac{premises}{conclusion}side\ condition$$

If the premises and the side condition are met then the conclusion can be drawn

The conclusion is a single judgement

The premises consist of one, none or more judgements

The side condition is a logical predicate

The rule name is just a convenient label

SOS rules

$$(num)\,\frac{}{\mathsf{N} \to n}$$

to be completed together

$$(sum)\,\frac{}{n_0 \oplus n_1 \to n}\; n = n_0 + n_1$$

$$(sumL)\,\frac{\mathsf{E}_0 \to \mathsf{E}_0'}{\mathsf{E}_0 \oplus \mathsf{E}_1 \to \mathsf{E}_0' \oplus \mathsf{E}_1}$$

$$(sumR)\,\frac{\mathsf{E}_1 \to \mathsf{E}_1'}{\mathsf{E}_0 \oplus \mathsf{E}_1 \to \mathsf{E}_0 \oplus \mathsf{E}_1'}$$

$$(prod)\,\frac{}{}$$

$$(prodL)\,\frac{}{}$$

$$(prodR)\,\frac{}{}$$

Some derivations

$$(prodL)\dfrac{(sumL)\dfrac{(num)\dfrac{}{1 \to 1}}{(1 \oplus 2) \to (1 \oplus 2)}}{(1 \oplus 2) \otimes (3 \oplus 4) \to (1 \oplus 2) \otimes (3 \oplus 4)}$$

$$(prodL)\dfrac{(sumR)\dfrac{(num)\dfrac{}{2 \to 2}}{(1 \oplus 2) \to (1 \oplus 2)}}{(1 \oplus 2) \otimes (3 \oplus 4) \to (1 \oplus 2) \otimes (3 \oplus 4)}$$

$$(prodL)\dfrac{(sum)\dfrac{}{(1 \oplus 2) \to 3}\ 3 = 1 + 2}{(1 \oplus 2) \otimes (3 \oplus 4) \to 3 \otimes (3 \oplus 4)}$$

A computation

$$
\begin{aligned}
(1 \oplus 2) \otimes (3 \oplus 4) &\rightarrow (1 \oplus 2) \otimes (3 \oplus 4) \\
&\rightarrow (1 \oplus 2) \otimes (3 \oplus 4) \\
&\rightarrow 3 \otimes (3 \oplus 4) \\
&\rightarrow 3 \otimes (3 \oplus 4) \\
&\rightarrow 3 \otimes (3 \oplus 4) \\
&\rightarrow 3 \otimes 7 \\
&\rightarrow 21 \\
&\not\rightarrow
\end{aligned}
$$

$$(1 \oplus 2) \otimes (3 \oplus 4) \rightarrow^{*} 21$$

Another computation

$$
\begin{aligned}
(1 \oplus 2) \otimes (3 \oplus 4) &\to (1 \oplus 2) \otimes (3 \oplus 4) \\
&\to (1 \oplus 2) \otimes (3 \oplus 4) \\
&\to (1 \oplus 2) \otimes (3 \oplus 4) \\
&\to (1 \oplus 2) \otimes 7 \\
&\to (1 \oplus 2) \otimes 7 \\
&\to 3 \otimes 7 \\
&\to 21 \\
&\not\to
\end{aligned}
$$

$$(1 \oplus 2) \otimes (3 \oplus 4) \to^{*} 21$$

Confluence?

We have seen that there are many different evaluation sequences (non-determinism)

Are we guaranteed they all lead to the same outcome?

We can change the inference rules to impose some specific evaluation strategy (determinism)

For example, we can impose a left-to-right evaluation of arguments by changing rules (sumR) and (prodR)

Evaluation strategies

$$(num)\frac{}{\mathsf{N} \to n}$$

to be completed together

$$(sum)\frac{}{n_0 \oplus n_1 \to n}\ n = n_0 + n_1$$

$$(sumL)\frac{\mathsf{E}_0 \to \mathsf{E}_0'}{\mathsf{E}_0 \oplus \mathsf{E}_1 \to \mathsf{E}_0' \oplus \mathsf{E}_1}$$

$$(sumR)\frac{\mathsf{E}_1 \to \mathsf{E}_1'}{n_0 \oplus \mathsf{E}_1 \to n_0 \oplus \mathsf{E}_1'}$$

$$(prod)\frac{}{}$$

$$(prodL)\frac{}{}$$

$$(prodR)\frac{}{}$$

A computation

$$
\begin{aligned}
(1 \oplus 2) \otimes (3 \oplus 4) &\to (1 \oplus 2) \otimes (3 \oplus 4) \\
&\to (1 \oplus 2) \otimes (3 \oplus 4) \\
&\to 3 \otimes (3 \oplus 4) \\
&\to 3 \otimes (3 \oplus 4) \\
&\to 3 \otimes (3 \oplus 4) \\
&\to 3 \otimes 7 \\
&\to 21 \\
&\not\to
\end{aligned}
$$

it is the only possible computation

$$(1 \oplus 2) \otimes (3 \oplus 4) \to^* 21$$

Big-step semantics

a step $\mathsf{E}_0 \longrightarrow n$

represents a whole computation!

How to define the transition relation?

Usually simpler than small-step rules

Can correspond to an efficient interpreter

SOS rules

$$(num)\,\frac{}{\mathsf{N} \longrightarrow n}$$

$$(sum)\,\frac{\mathsf{E}_0 \longrightarrow n_0 \quad \mathsf{E}_1 \longrightarrow n_1}{\mathsf{E}_0 \oplus \mathsf{E}_1 \longrightarrow n}\; n = n_0 + n_1$$

$$(prod)\,\frac{}{}$$

to be completed together

A derivation

$$
(prod)\,\dfrac{(sum)\,\dfrac{(num)\,\dfrac{}{1 \longrightarrow 1} \quad (num)\,\dfrac{}{2 \longrightarrow 2}}{(1 \oplus 2) \longrightarrow 3} \qquad (sum)\,\dfrac{(num)\,\dfrac{}{3 \longrightarrow 3} \quad (num)\,\dfrac{}{4 \longrightarrow 4}}{(3 \oplus 4) \longrightarrow 7}}{(1 \oplus 2) \otimes (3 \oplus 4) \longrightarrow 21}
$$

cannot express non-terminating computations

(derivations are possible only for terminating programs)

Denotational semantics

domain + interpretation function

$\mathbb{N}$ $\qquad\qquad$ $\mathcal{E}[\![\cdot]\!] : Exp \to \mathbb{N}$

the choice of the domain has immediate consequences: anyone knows already that expressions are **deterministic** and **normalising**

every expression has at most one answer

every expression has an answer

a term (a piece of syntax)

$\mathcal{E}[\![\ E\ ]\!]$

its denotation (a semantic object)

different syntactic categories may require different domains!

Structural induction

$$\mathcal{E}[\![\mathsf{N}]\!] = n$$

$$\mathcal{E}[\![\mathsf{E}_0 \oplus \mathsf{E}_1]\!] = \mathcal{E}[\![\mathsf{E}_0]\!] + \mathcal{E}[\![\mathsf{E}_1]\!]$$

$$\mathcal{E}[\![\mathsf{E}_0 \otimes \mathsf{E}_1]\!] = \mathcal{E}[\![\mathsf{E}_0]\!] \cdot \mathcal{E}[\![\mathsf{E}_1]\!]$$

compositionality principle

the meaning of a composite construct does not depend on the particular form of the constituent constructs, but only on their meanings

An evaluation

$$
\begin{aligned}
\mathcal{E}[\![(\mathsf{1} \oplus \mathsf{2}) \otimes (\mathsf{3} \oplus \mathsf{4})]\!] &= \mathcal{E}[\![\mathsf{1} \oplus \mathsf{2}]\!] \cdot \mathcal{E}[\![\mathsf{3} \oplus \mathsf{4}]\!] \\
&= (\mathcal{E}[\![\mathsf{1}]\!] + \mathcal{E}[\![\mathsf{2}]\!]) \cdot (\mathcal{E}[\![\mathsf{3}]\!] + \mathcal{E}[\![\mathsf{4}]\!]) \\
&= (1 + 2) \cdot (3 + 4) \\
&= 21
\end{aligned}
$$

Comparison

$\mathsf{E} \to^* n$ $\mathsf{E} \longrightarrow n$ $\mathcal{E}[\![\mathsf{E}]\!] = n$

normalisation / termination?

$\forall \mathsf{E}.\ \exists n.\ \mathsf{E} \to^* n$	$\forall \mathsf{E}.\ \exists n.\ \mathsf{E} \longrightarrow n$	$\forall \mathsf{E}.\ \exists n.\ \mathcal{E}[\![\mathsf{E}]\!] = n$
must be proved	must be proved	obvious

Comparison

determinacy?

$$\forall \mathsf{E}, n, m.\ \begin{pmatrix} \mathsf{E} \rightarrow^* n \\ \wedge \\ \mathsf{E} \rightarrow^* m \end{pmatrix} \Rightarrow n = m$$

must be proved

$$\forall \mathsf{E}, n, m.\ \begin{pmatrix} \mathsf{E} \longrightarrow n \\ \wedge \\ \mathsf{E} \longrightarrow m \end{pmatrix} \Rightarrow n = m$$

must be proved

$$\forall \mathsf{E}, n, m.\ \begin{pmatrix} \mathcal{E}[\![\mathsf{E}]\!] = n \\ \wedge \\ \mathcal{E}[\![\mathsf{E}]\!] = m \end{pmatrix} \Rightarrow n = m$$

obvious

Comparison

$\mathsf{E} \rightarrow^* n$ $\qquad$ $\mathsf{E} \longrightarrow n$ $\qquad$ $\mathcal{E}[\![\mathsf{E}]\!] = n$

consistency?

$$\forall \mathsf{E}, n.\ (\mathsf{E} \rightarrow^* n \quad \Leftrightarrow \quad \mathsf{E} \longrightarrow n \quad \Leftrightarrow \quad \mathcal{E}[\![\mathsf{E}]\!] = n)$$

must be proved

Comparison

induced equivalences

$$\mathsf{E}_0 \equiv_s \mathsf{E}_1$$
$$\forall n.\ (\mathsf{E}_0 \to^* n \Leftrightarrow \mathsf{E}_1 \to^* n)$$

$$\mathsf{E}_0 \equiv_b \mathsf{E}_1$$
$$\forall n.\ (\mathsf{E}_0 \longrightarrow n \Leftrightarrow \mathsf{E}_1 \longrightarrow n)$$

$$\mathsf{E}_0 \equiv_d \mathsf{E}_1$$
$$\mathcal{E}[\![\mathsf{E}_0]\!] = \mathcal{E}[\![\mathsf{E}_1]\!]$$

do they all coincide?

Comparison

then we can prove / disprove:

properties of specific expressions

$$2 \otimes 6 \equiv_s 3 \otimes 4$$

properties of generic expressions

$$\forall \mathsf{E}, \mathsf{E}_1, \mathsf{E}_2.\ \mathsf{E} \otimes (\mathsf{E}_1 \oplus \mathsf{E}_2) \equiv_d (\mathsf{E} \otimes \mathsf{E}_1) \oplus (\mathsf{E} \otimes \mathsf{E}_2)$$

Comparison

congruences?

$$\mathsf{C}[\bullet] \ ::= \ [\bullet] \ \mid \ \mathsf{C}[\bullet] \oplus \mathsf{E} \ \mid \ \mathsf{E} \oplus \mathsf{C}[\bullet] \ \mid \ \mathsf{C}[\bullet] \otimes \mathsf{E} \ \mid \ \mathsf{E} \otimes \mathsf{C}[\bullet]$$

contexts with a hole $\mathsf{C}[\bullet]$

filled context $\mathsf{C}[\mathsf{E}]$

$\forall \mathsf{E}_0, \mathsf{E}_1, \mathsf{C}[\bullet].\ (\mathsf{E}_0 \equiv_s \mathsf{E}_1 \Rightarrow \mathsf{C}[\mathsf{E}_0] \equiv_s \mathsf{C}[\mathsf{E}_1])$ must be proved

$\forall \mathsf{E}_0, \mathsf{E}_1, \mathsf{C}[\bullet].\ (\mathsf{E}_0 \equiv_b \mathsf{E}_1 \Rightarrow \mathsf{C}[\mathsf{E}_0] \equiv_b \mathsf{C}[\mathsf{E}_1])$ must be proved

$\forall \mathsf{E}_0, \mathsf{E}_1, \mathsf{C}[\bullet].\ (\mathsf{E}_0 \equiv_d \mathsf{E}_1 \Rightarrow \mathsf{C}[\mathsf{E}_0] \equiv_d \mathsf{C}[\mathsf{E}_1])$ obvious

Exercise

Expressions with variables $\mathsf{E} ::= x \mid \mathsf{N} \mid \mathsf{E} \oplus \mathsf{E} \mid \mathsf{E} \otimes \mathsf{E}$

How to evaluate expressions such as $(x \oplus 4) \otimes y$?

Need some memories $\mathbb{M} \triangleq \{\sigma \mid \sigma : X \to \mathbb{N}\}$

machine states $\langle \mathsf{E}, \sigma \rangle$

interpretation function $\mathcal{E}[\![\cdot]\!] : Exp \to (\mathbb{M} \to \mathbb{N})$

Let's redefine the various semantics and properties