# **Denotational Semantics**

Lectures for Part II CST 2022/23

**Prof Marcelo Fiore** 

Course web page:

http://www.cl.cam.ac.uk/teaching/2223/DenotSem/

# Topic 1

Introduction

## What is this course about?

• General area.

*Formal methods*: Mathematical techniques for the specification, development, and verification of software and hardware systems.

• Specific area.

*Formal semantics*: Mathematical theories for ascribing meanings to computer languages.

# Why do we care?

- Rigour.
  - ... specification of programming languages
  - ... justification of program transformations
- Insight.
  - ... generalisations of notions computability
  - ... higher-order functions
  - ... data structures

- Feedback into language design.
  - ... continuations
  - ... monads
- Reasoning principles.
  - ... Scott induction
  - ... Logical relations
  - ... Co-induction

## **Operational.**

Meanings for program phrases defined in terms of the *steps of computation* they can take during program execution.

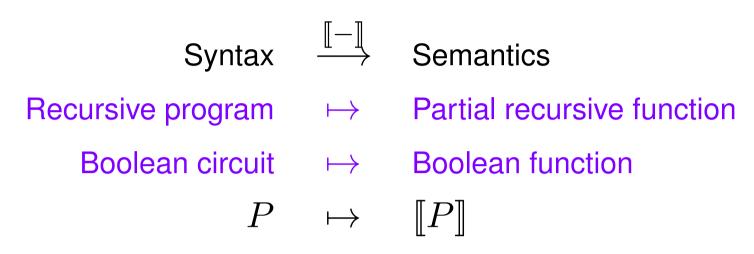
# Axiomatic.

Meanings for program phrases defined indirectly via the *axioms and rules* of some logic of program properties.

# **Denotational**.

Concerned with giving *mathematical models* of programming languages. Meanings for program phrases defined abstractly as elements of some suitable mathematical structure.

# **Basic idea of denotational semantics**



# Concerns:

- Abstract models (*i.e.* implementation/machine independent).
   ~> Lectures 2, 3 and 4.
- Compositionality.

 $\rightsquigarrow$  Lectures 5 and 6.

Relationship to computation (*e.g.* operational semantics).
 ~> Lectures 7 and 8.

# Characteristic features of a denotational semantics

- Each phrase (= part of a program), P, is given a denotation,
   [P] a mathematical object representing the contribution of P to the meaning of any complete program in which it occurs.
- The denotation of a phrase is determined just by the denotations of its subphrases (one says that the semantics is compositional).

IMP<sup>-</sup> syntax

Arithmetic expressions

 $A \in \mathbf{Aexp} ::= \underline{n} \mid L \mid A + A \mid \dots$ 

where n ranges over *integers* and L over a specified set of *locations* L

Boolean expressions

 $B \in \mathbf{Bexp}$  ::= true | false | A = A | ... |  $\neg B$  | ...

Commands

 $C \in \mathbf{Comm} \quad ::= \quad \mathbf{skip} \quad | \quad L := A \quad | \quad C; C$  $| \quad \mathbf{if} \ B \mathbf{then} \ C \mathbf{else} \ C$ 

Semantic functions

$$\mathcal{A}: \operatorname{\mathbf{Aexp}} \to (\operatorname{State} \to \mathbb{Z})$$
$$\mathcal{A}[\operatorname{AI}: \operatorname{State} \to \operatorname{\mathcal{H}}]$$

where

$$\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$$

State =  $(\mathbb{L} \to \mathbb{Z})$ 

Semantic functions

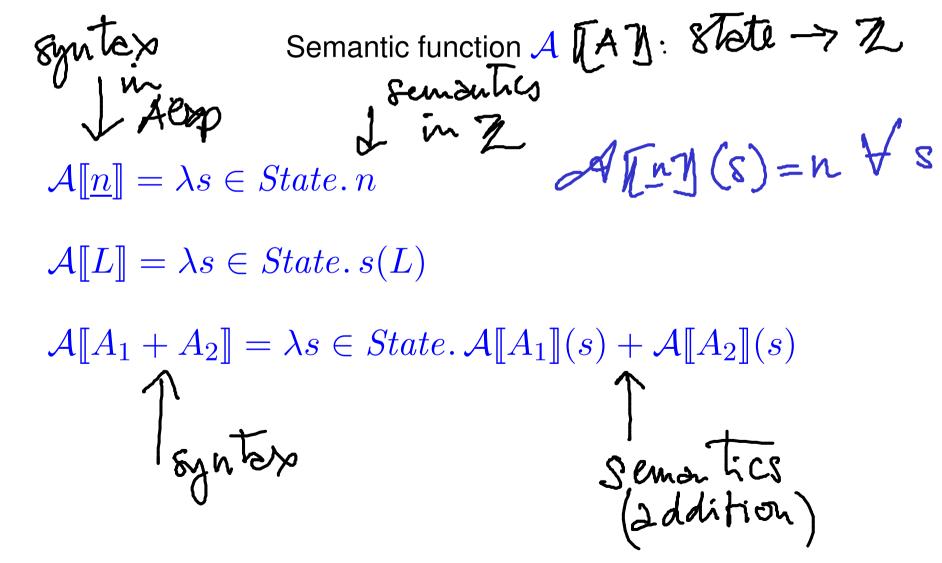
where

$$\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$$
$$\mathbb{B} = \{true, false\}$$
$$State = (\mathbb{L} \to \mathbb{Z})$$

where

Semantic functions  $\mathcal{C}: \mathbf{Comm} \to (State \rightharpoonup State)$  $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$  $\mathbb{B} = \{ true, false \}$ State =  $(\mathbb{L} \to \mathbb{Z})$ 

### **Basic example of denotational semantics (III)**



Semantic function  $\mathcal{B}$ 

 $\mathcal{B}\llbracket \mathbf{true} \rrbracket = \lambda s \in State. true$  $\mathcal{B}\llbracket \mathbf{false} \rrbracket = \lambda s \in State. false$  $\mathcal{B}\llbracket A_1 = A_2 \rrbracket = \lambda s \in State. eq \left(\mathcal{A}\llbracket A_1 \rrbracket(s), \mathcal{A}\llbracket A_2 \rrbracket(s)\right)$  $\text{where } eq(a, a') = \begin{cases} true & \text{if } a = a' \\ false & \text{if } a \neq a' \end{cases}$ 



Semantic function  $C[C]:(State \rightarrow State)$ 

$$[skip] = \lambda s \in State.s = 12$$
 State

**NB:** From now on the names of semantic functions are omitted!

#### A simple example of compositionality

Given partial functions  $\llbracket C \rrbracket, \llbracket C' \rrbracket : State \rightarrow State$  and a function  $\llbracket B \rrbracket : State \rightarrow \{true, false\}$ , we can define

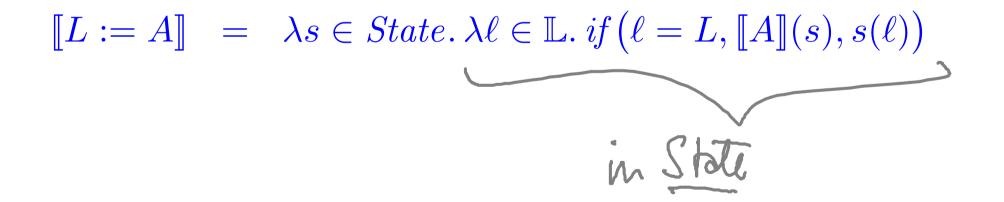
$$\llbracket \mathbf{if} \ B \ \mathbf{then} \ C \ \mathbf{else} \ C' \rrbracket = \\\lambda s \in State. \ if \left( \llbracket B \rrbracket(s), \llbracket C \rrbracket(s), \llbracket C' \rrbracket(s) \right)$$

where

$$if(b, x, x') = \begin{cases} x & \text{if } b = true \\ x' & \text{if } b = false \end{cases}$$

#### **Basic example of denotational semantics (VI)**

Semantic function  $\mathcal{C}$ 



Denotation of sequential composition C; C' of two commands

$$\llbracket C; C' \rrbracket = \llbracket C' \rrbracket \circ \llbracket C \rrbracket = \lambda s \in State. \llbracket C' \rrbracket \left( \llbracket C \rrbracket (s) \right)$$

given by composition of the partial functions from states to states  $\llbracket C \rrbracket, \llbracket C' \rrbracket : State \rightarrow State$  which are the denotations of the commands.

$$(C_{1}; C_{2}); C_{3} \approx C_{1}; (C_{2}; C_{3})$$

Denotation of sequential composition C; C' of two commands

$$\llbracket C; C' \rrbracket = \llbracket C' \rrbracket \circ \llbracket C \rrbracket = \lambda s \in State. \llbracket C' \rrbracket \left( \llbracket C \rrbracket (s) \right)$$

given by composition of the partial functions from states to states  $\llbracket C \rrbracket, \llbracket C' \rrbracket : State \rightarrow State$  which are the denotations of the commands.

Cf. operational semantics of sequential composition:

$$\frac{C, s \Downarrow s' \quad C', s' \Downarrow s''}{C; C', s \Downarrow s''}$$

$$NB \quad C, s \Downarrow s' \iff ICD(s) = s'$$

# $\llbracket \mathbf{while} \ B \ \mathbf{do} \ C \rrbracket$

 $[[while B dr C]](s) = \cdots [B](s) \cdots [C](s) \cdots$ Tuhle true de skap I (s) = 1 modefined. Vsc85te. Tubile true do skip ]: State - State I is The unde field partial function; i.e, with enpty graph denoted of or L

[Inhile plac de skop](s) = s Ruhle folse do skp y = rd goote NB: while filse to skip ~ skip Tutile B do CM while B do C ~ If B Then (C; nhile B doc) Y else skip operationally

Tuhile B do C M(S) = [f B Then (C; while B do C) else enp 7/s) = 4(TBJ(S), TC; while B d CM(S), S) $= \mathcal{F}(\mathsf{FB}\mathcal{J}(\mathcal{S}), [[while B d c C J([[C \mathcal{J}]S), S])$ NB: Tinhile B dr CJ is à state bransformer //w:State ~ State sahrfying  $w = \lambda s \cdot f(TB)(s), w(TCYs), s)$   $he (\lambda w. \lambda s. f(TB)(s), w(TCYs), s)$