## 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?

#### Why do we care?

- Rigour.
  - ... specification of programming languages
  - ... justification of program transformations

#### 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

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

#### **Styles of formal semantics**

Operational.

Axiomatic.

**Denotational**.

#### Operational.

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

Axiomatic.

**Denotational**.

#### 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**.

#### **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**.

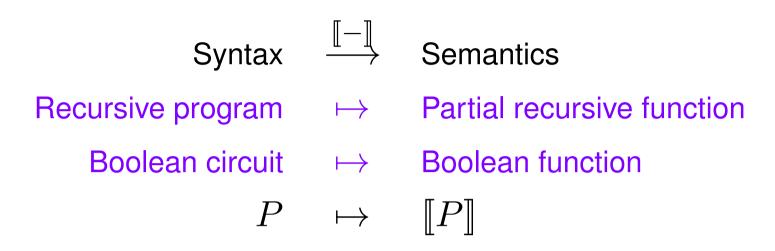
Concerned with giving *mathematical models* of programming languages. Meanings for program phrases defined abstractly as elements of some suitable mathematical structure.

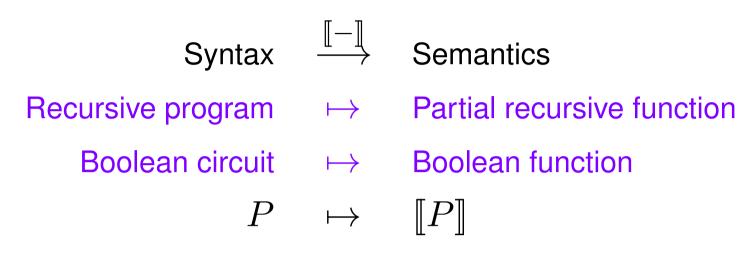
Syntax  $\xrightarrow{\mathbb{I}-\mathbb{I}}$  Semantics

$$P \quad \mapsto \quad \llbracket P \rrbracket$$



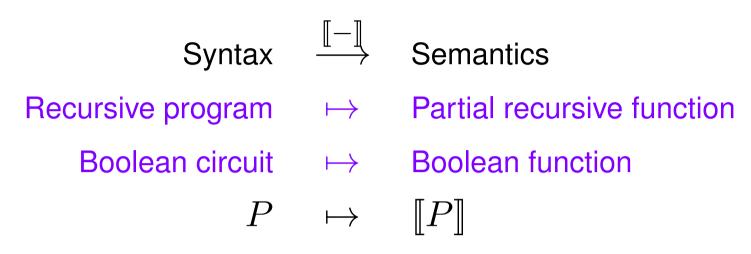
$$P \quad \mapsto \quad \llbracket P \rrbracket$$





#### Concerns:

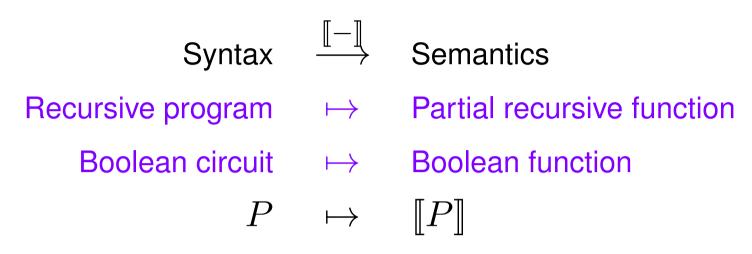
Abstract models (*i.e.* implementation/machine independent).
 ~> Lectures 2, 3 and 4.



#### Concerns:

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

 $\rightsquigarrow$  Lectures 5 and 6.



#### 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

 $B \in \mathbf{Bexp} \quad ::= \quad \mathbf{true} \mid \mathbf{false} \mid A = A \mid \dots \\ \mid \quad \neg B \mid \dots$ 

Commands

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

**Basic example of denotational semantics (II)** 

Semantic functions

$$\mathcal{A}: \mathbf{Aexp} \to (State \to \mathbb{Z})$$

where

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

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

Semantic functions

$$\mathcal{A}: \quad \mathbf{Aexp} \to (State \to \mathbb{Z})$$
$$\mathcal{B}: \quad \mathbf{Bexp} \to (State \to \mathbb{B})$$

where

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

Semantic functions

$$\mathcal{A}: \quad \mathbf{Aexp} \to (State \to \mathbb{Z})$$
$$\mathcal{B}: \quad \mathbf{Bexp} \to (State \to \mathbb{B})$$
$$\mathcal{C}: \quad \mathbf{Comm} \to (State \to State)$$

where

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

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

Semantic function  $\mathcal{A}$ 

 $\mathcal{A}[\![\underline{n}]\!] = \lambda s \in State. n$ 

 $\mathcal{A}\llbracket L \rrbracket = \lambda s \in State. \, s(L)$ 

 $\mathcal{A}\llbracket A_1 + A_2 \rrbracket = \lambda s \in State. \, \mathcal{A}\llbracket A_1 \rrbracket(s) + \mathcal{A}\llbracket A_2 \rrbracket(s)$ 

Semantic function  $\mathcal{B}$ 

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

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

Semantic function  $\mathcal{C}$ 

 $\llbracket skip \rrbracket = \lambda s \in State. s$ 

**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  $\mathcal{C}$ 

## $\llbracket L := A \rrbracket = \lambda s \in State. \ \lambda \ell \in \mathbb{L}. \ if \left( \ell = L, \llbracket A \rrbracket(s), s(\ell) \right)$

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  $\llbracket C \rrbracket, \llbracket C' \rrbracket : State \longrightarrow State$  which are the denotations of the commands.

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:

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

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

Fixed point property of  $\llbracket \mathbf{while} \ B \ \mathbf{do} \ C \rrbracket$ 

 $\llbracket \mathbf{while} \ B \ \mathbf{do} \ C \rrbracket = f_{\llbracket B \rrbracket, \llbracket C \rrbracket}(\llbracket \mathbf{while} \ B \ \mathbf{do} \ C \rrbracket)$ where, for each  $b : State \to \{true, false\}$  and  $c : State \to State$ , we define

 $f_{b,c}: (State \rightarrow State) \rightarrow (State \rightarrow State)$ 

as

 $f_{b,c} = \lambda w \in (State \rightarrow State). \ \lambda s \in State. \ if (b(s), w(c(s)), s).$ 

 $\llbracket \mathbf{while} \ B \ \mathbf{do} \ C \rrbracket = f_{\llbracket B \rrbracket, \llbracket C \rrbracket}(\llbracket \mathbf{while} \ B \ \mathbf{do} \ C \rrbracket)$ where, for each  $b : State \to \{true, false\}$  and  $c : State \to State$ , we define

$$f_{b,c}: (State \rightarrow State) \rightarrow (State \rightarrow State)$$

as

 $f_{b,c} = \lambda w \in (State \rightarrow State). \ \lambda s \in State. \ if (b(s), w(c(s)), s).$ 

- Why does  $w = f_{\llbracket B \rrbracket, \llbracket C \rrbracket}(w)$  have a solution?
- What if it has several solutions—which one do we take to be
  [while B do C]?

Approximating  $\llbracket$  while  $B \operatorname{do} C \rrbracket$ 

Approximating  $\llbracket while B \operatorname{do} C \rrbracket$ 

$$\begin{split} f_{\llbracket B \rrbracket, \llbracket C \rrbracket}^{n}(\bot) \\ &= \lambda s \in State. \\ & \left\{ \begin{array}{ll} \llbracket C \rrbracket^{k}(s) & \text{if } \exists \ 0 \leq k < n. \ \llbracket B \rrbracket(\llbracket C \rrbracket^{k}(s)) = false \\ & \text{and } \forall \ 0 \leq i < k. \ \llbracket B \rrbracket(\llbracket C \rrbracket^{i}(s)) = true \\ \uparrow & \text{if } \forall \ 0 \leq i < n. \ \llbracket B \rrbracket(\llbracket C \rrbracket^{i}(s)) = true \end{array} \right. \end{split}$$

$$D \stackrel{\mathrm{def}}{=} (State \rightharpoonup State)$$

• Partial order  $\sqsubseteq$  on D:

 $w \sqsubseteq w'$  iff for all  $s \in State$ , if w is defined at s then so is w' and moreover w(s) = w'(s).

iff the graph of w is included in the graph of w'.

- Least element  $\perp \in D$  w.r.t.  $\sqsubseteq$ :
  - $\perp$  = totally undefined partial function
    - = partial function with empty graph

(satisfies  $\perp \sqsubseteq w$ , for all  $w \in D$ ).