# 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.

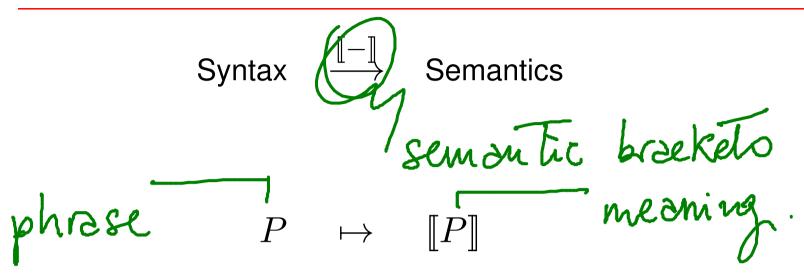
# Axiomatic.

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

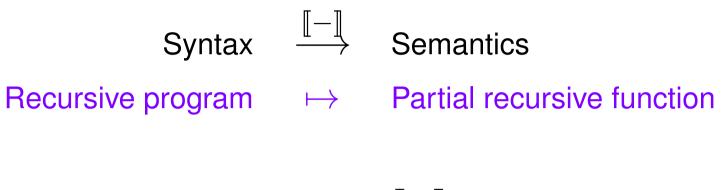
#### **Denotational**.

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**

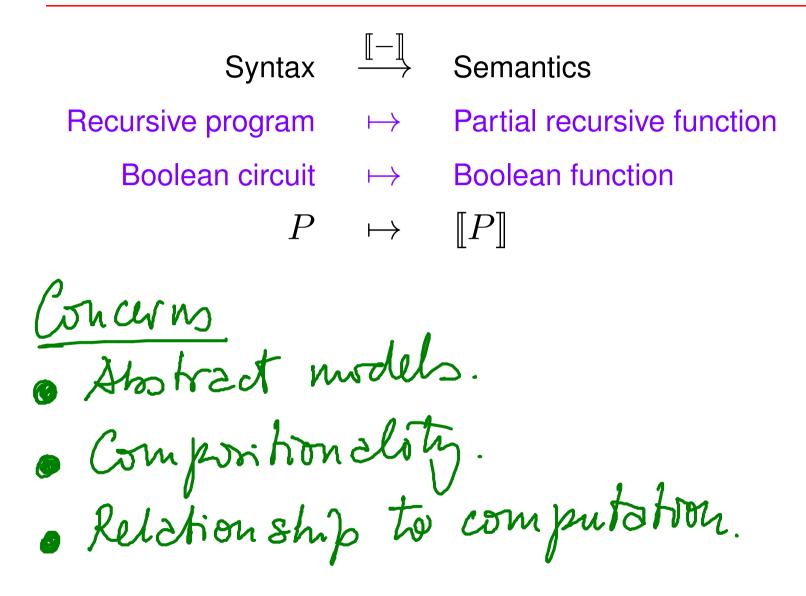


#### **Basic idea of denotational semantics**



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

## **Basic idea of denotational semantics**



# 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

$$\begin{array}{rcl} \mathcal{A}: & \mathbf{Aexp} \rightarrow (State \rightarrow \mathbb{Z}) \\ & \mathcal{A}[[E]]: & State \rightarrow \mathbb{Z} & seState \\ & \text{where} & EeAexp & \mathcal{A}[[E]](s) \in \mathbb{Z} \\ & \mathbb{Z} &= \{\ldots, -1, 0, 1, \ldots\} \end{array}$$

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

Semantic functions

$$\begin{array}{rcl} \mathcal{A}: & \mathbf{Aexp} \to (State \to \mathbb{Z}) \\ \mathcal{B}: & \mathbf{Bexp} \to (State \to \mathbb{B}) \\ \end{array}$$

$$\begin{array}{rcl} \mathcal{E} \in \mathcal{BErp} \\ \mathsf{where} \end{array}, & \mathcal{S} \in \mathcal{State} \\ \mathbb{Z} &= \{\ldots, -1, 0, 1, \ldots\} \\ \mathbb{B} &= \{true, false\} \\ \end{array}$$

$$\begin{array}{rcl} State &= (\mathbb{L} \to \mathbb{Z}) \end{array}$$

Semantic functions

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

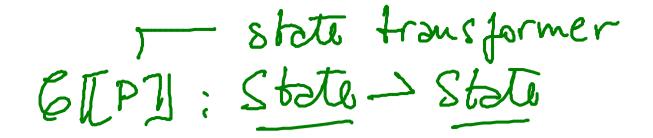
where

$$\mathbb{Z} = \{\dots, -1, 0, 1, \dots\} \qquad \begin{array}{l} \text{LEL} \\ \text{sell} \\ \text{second} \\ \text{second} \\ \text{memory} \\ \text{state} \\ \text{store} \\$$

11

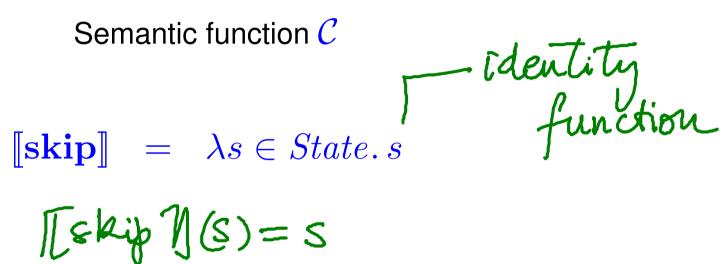
Recall 
$$\lambda z. e[z]$$
 denotes the function that on  
input, say a, results in output e(a).  
Basic example of denotational semantics (III)  
Semantic function A  
 $A[[n]] = \lambda s \in State.n$   
 $A[[n]] = \lambda s \in State.n$   
 $A[[L]] = \lambda s \in State.s(L)$   
 $A[[L]] = \lambda s \in State.s(L)$   
 $A[[A_1 + A_2]] = \lambda s \in State. A[[A_1]](s) + A[[A_2]](s)$   
 $A[[A_1 + A_2]] \in State. A[[A_1]](s) + A[[A_1]](s) + A[[A_1]](s) + A[[A_2]](s)$   
 $A[[A_1 + A_2]] \in State. A[[A_1]](s) + A[[A_2]](s)$   
 $A[[A_1 + A_2]](s) + A[[A_1]](s) + A[[A_2]](s)$   
 $A[[A_1 + A_2]](s) + A[[A_1]](s) + A[[A_2]](s)$   
 $A[[A_1 + A_2]](s) + A[[A_1]](s) + A[[A_1]](s) + A[[A_1]]$ 

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





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



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

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  $\mathcal{C}$ 

$$\begin{bmatrix} L := A \end{bmatrix} = \lambda s \in State. \ \lambda \ell \in \mathbb{L}. \ if \left(\ell = L, \llbracket A \rrbracket(s), s(\ell)\right)$$
$$\begin{bmatrix} L := A \rrbracket(s) = \lambda \ell \in \mathbb{L}. \ \int \llbracket A \rrbracket(s) = \lambda \ell \in \mathbb{L}. \ \int I \llbracket A \rrbracket(s) = \lambda \ell \in \mathbb{L}. \ \int I \llbracket A \rrbracket(s) = \lambda \ell \in \mathbb{L}. \ \int I \llbracket A \rrbracket(s) = \lambda \ell \in \mathbb{L}.$$

Denotation of sequential composition C; C' of two commands

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

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.

$$\Pi[C;C'](S) = \Pi[C'](\Pi[C](S))$$

Denotation of sequential composition C; C' of two commands

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

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:

$$\frac{C,s \Downarrow s' C',s' \Downarrow s''}{C;C',s \Downarrow s''}.$$
  
One conshow  
 $\forall C,s,s'. \Pi C \Pi (s) = s' \Pi C, s \Downarrow s'.$ 

[while 
$$B \text{ do } C$$
]: State  $\longrightarrow$  State  
 $C \in Comm ::= \cdots | While B \text{ do } C | \cdots$   
[while  $B \text{ do } C \mathcal{Y}(S)$   
 $= \cdots [B](S) \cdots [C](S) \cdots$ 

[while B do C] : State -> State

Program équivalence while B do C = if B Then (C; while B do C) else skip. Expect Ruhibe B do CZ [If B Then (C; while B do C) else skip 7] That is,

for all states s Tulike B do CJ(S) = II of B Then (C; while B do C) else ship] (S) = if (IEBJ(S), IT while B do CD(ICCJS), S) Templing to define [while B do c 7] (S) def f(IBM(s), Il while B do CM(IICT)s), s) but This is non-sense!

Ruhile B do CM rs a fixed point real that a fixed of a function f is an element x such that x = f(x). Tuhike B do CJ = A SE State. if (TBJ(S), Tuhle B do CJ(ACJS), S) f TBM, TICD: (Stabe → State) → (State → State) s.t. f TBY, TICY (Tuhile B do CY)=[[uhile Bdo C]]

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  $b: State \rightarrow \{true, false\}$  and  $c: State \rightarrow State$ , we define  $f_{b,c}: (State \rightarrow State) \rightarrow (State \rightarrow State)$ as  $\overline{f_{b,c}} = \lambda w \in (State \rightarrow State). \ \lambda s \in State. \ if (b(s), w(c(s)), s).$ [Inhile B doc]] = fix (f[B], [C])

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

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

- 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]?

fielfEBY, ECZ) by approximation

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

• Wo: State -> State d'empty partial function Wo (S) = 1 ("undefined") •  $W_1 = f_{ITBN, ECM}(w_0)$  $W_1(S) = \mathcal{A}(\Pi B \mathcal{Y}(S), W_0(\Gamma C \mathcal{Y}_S), S)$  $= i \left( (TBU(s), \uparrow, s) \right)$ 

•  $w_2 = f Tren, TCY(w_1)$  $w_2(s) = f(\pi B \eta(s), \omega_1(\pi C \eta(s)), s)$ =  $\#(\Pi B \mathcal{Y}(S)),$  $\mathcal{F}(\mathcal{I}\mathcal{B}\mathcal{Y}(\mathcal{I}\mathcal{C}\mathcal{V}(\mathcal{S})), \mathcal{T}, \mathcal{I}\mathcal{C}\mathcal{Y}(\mathcal{S})),$ (2 •  $\omega_n = f(TB), acy(\omega_{n-1})$ sporonimetes Twhile B do CJ up to n iterations.

union  $\bigcup_{n \in \mathcal{A}} W_n = \bigcup_{n \in \mathcal{A}} f_{\overline{L}B} Y, \overline{L}(f)$ prin  $\bigcup_{n \in \mathcal{A}} W_n = \bigcup_{n \in \mathcal{A}} f_{\overline{L}B} Y, \overline{L}(f)$ Approximating [while B do C]

$$w_{n} = f_{[B],[C]}^{n}(\bot)$$

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

• while true do C  $\omega_0 =$  $w_1 = \lambda s. i (IItrue I(s), \hat{\tau}, s)$  $= \lambda S. \Lambda = Wo$ Ewhite true do  $CJ = C_n w_n = O_L$ = \_

· white plse do C  $\omega_0 = \bot$  $\omega_1 = \lambda s. \mathcal{F}(\mathcal{I} \neq \mathcal{I}(\mathcal{S}), \mathcal{T}, s) = \lambda s. s$ for all  $n = \lambda s. s$ .  $II while folse do C I = U(L, \lambda S.S, ...)$  $= \lambda s.s$ = [[skip]]

The domain of state transformers.  

$$D \stackrel{\text{def}}{=} (State \rightarrow State)$$

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

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

For a chain of approximations of state transformers NOEWIE .... EWNE .... (nEW)

We let  $U(\omega_0 \leq \omega_1 \leq \dots \leq \omega_n \leq \dots) = \bigcup_{n \in \mathcal{N}} \omega_n$ be The state transformer whose graph is  $\bigcup_{n \in \mathcal{N}} g(aph(\omega_n)).$