A FIRST COURSE IN DOMAIN THEORY

GRAHAM HUTTON

CHALMERS UNIVERSITY, APRIL 1994.

Lecture 1: introduction

denotational semantics, non-termination and $\perp$, partially ordered sets, monotonic functions.

Lecture 2: recursively defined programs

chains, directed sets, least upper bound, cpo's and continuous functions, fixpoints.

Lecture 3: constructions on cpo's

sums, products, function spaces, ...

Lecture 4: domains

"finite" elements, algebraicity, Scott domains

Lecture 5: recursively defined domains

$D \simeq D \to D$, Scott's "inverse limit construction".

Lecture 6: powerdomains and concurrent programs

Hoare, Plotkin and Smyth powerdomains.

LECTURE 1: INTRODUCTION

What is domain theory about?

"The subject has its roots in the seminal work of Dana Scott and Christopher Strachey in the early 1970s, and was established in order to have appropriate spaces on which to define semantic functions in the denotational approach to programming language semantics."

DENOTATIONAL SEMANTICS

Let P be a programming language with abstract syntax formally defined using a BNF grammar.

A denotational semantics for P consists of:

① a semantic domain for each syntactic category (expressions, commands, ...)

② a valuation function for each syntactic category, which assigns each phrase of syntax a denotation in the appropriate semantic domain.

Note: valuation functions must be homomorphisms (the denotation of each phrase is defined purely in terms of the denotation of its subphrases)

This property is normally called compositionality.

EXAMPLE: A SIMPLE IMPERATIVE LANGUAGE

$$E ::= Z \mid I \mid E_1 + E_2 \mid E_1 - E_2$$

$$B ::= \text{false} \mid \text{true} \mid \neg B \mid E_1 = E_2$$

$$C ::= I := E \mid C_1 ; C_2 \mid \text{if } B \text{ then } C_1 \text{ else } C_2$$

semantic domains:

$$\text{expr} = \text{state} \rightarrow \mathbb{Z}$$

$$\text{bool} = \text{state} \rightarrow \mathbb{B}$$

$$\text{comm} = \text{state} \rightarrow (\mathbb{Z} \times \text{state})$$

$$\text{where state} = I \rightarrow \mathbb{Z}$$

valuation functions:

$$\mathcal{E}[\![-]\!] : E \rightarrow \text{expr}$$

$$\mathcal{E}[\![I]\!]\sigma = \sigma(I)$$

$$\mathcal{E}[\![E_1 + E_2]\!]\sigma = \mathcal{E}[\![E_1]\!]\sigma + \mathcal{E}[\![E_2]\!]\sigma$$

$$\mathcal{B}[\![-]\!] : B \to \mathrm{bool}$$

$$\mathcal{B}[\![\mathrm{false}]\!]\sigma = F$$

$$\mathcal{B}[\![\mathrm{true}]\!]\sigma = T$$

$$\mathcal{B}[\![\neg B]\!]\sigma = \begin{cases} T & \text{if } \mathcal{B}[\![B]\!]\sigma = F \\ F & \text{otherwise} \end{cases}$$

$$\mathcal{B}[\![E_1 = E_2]\!]\sigma = \begin{cases} T & \text{if } \mathcal{E}[\![E_1]\!]\sigma = \mathcal{E}[\![E_2]\!]\sigma \\ F & \text{otherwise} \end{cases}$$

$$\mathcal{C}[\![-]\!] : C \to \mathrm{comm}$$

$$\mathcal{C}[\![I := E]\!]\sigma = (v, \sigma[I \mapsto v]) \quad \text{where } v = \mathcal{E}[\![E]\!]\sigma$$

$$\mathcal{C}[\![C_1 ; C_2]\!]\sigma = \mathcal{C}[\![C_2]\!]\sigma' \quad \text{where } (v, \sigma') = \mathcal{C}[\![C_1]\!]\sigma$$

$$\mathcal{C}[\![\text{if } B \text{ then } C_1 \text{ else } C_2]\!]\sigma = \begin{cases} \mathcal{C}[\![C_1]\!]\sigma & \text{if } \mathcal{B}[\![B]\!]\sigma = T \\ \mathcal{C}[\![C_2]\!]\sigma & \text{otherwise} \end{cases}$$

FOUNDATIONAL PROBLEMS

A key idea in Strachey's early work (c.a. 1964) on denotational semantics was that, formally, denotations were specified using the untyped λ-calculus.

<u>problem</u>: the untyped λ-calculus had no known model.

<u>solution</u>: Scott found one, and went on to establish the theory of semantic domains.

We begin our study of Scott's ideas by considering why semantic domains can't simply be sets.

There are two main reasons:

1. recursively defined programs;

2. recursively defined semantic domains.

① RECURSIVELY DEFINED PROGRAMS

Consider the programs $f, g : nat \to nat$ "defined" by

① $f(x) = f(x) + 1$

② $g(x) = g(x)$

Intuitively,

evaluating $f(a)$ or $g(a)$ for any $a : nat$ will loop

But semantically (with sets as domains),

there is no function $f : \mathbb{N} \to \mathbb{N}$ satisfying ①

any function $g : \mathbb{N} \to \mathbb{N}$ satisfies ②

> To properly deal with the semantics of recursion (or iteration) we need an explicit notion of "non-termination" at the semantics level.

(2) RECURSIVELY DEFINED SEMANTIC DOMAINS

Suppose we extend our example imperative language with (parameterless) procedures:

$$E ::= \ldots \mid \text{proc } C$$

with semantics such that

$$(\text{inc} := \text{proc } (a := a + 1)) \; ; \; \text{inc} \; ; \; \text{inc}$$

yields the same result value as

$$a := a + 1 \; ; \; a := a + 1 \, .$$

The semantic domains are now:

$$\begin{aligned} \text{expr} &= \text{state} \to \text{value} \\ \text{bool} &= \text{state} \to \mathbb{B} \\ \text{comm} &= \text{state} \to (\text{value} \times \text{state}) \end{aligned}$$

$$\text{where } \text{value} = \mathbb{Z} \boxed{+ \text{ comm}} \quad \text{the new part}$$

$$\text{state} = I \to \text{value}$$

Now the equation for comm is recursive:

$$\text{comm} = \ldots \text{comm} \ldots \to \ldots \text{comm} \ldots$$

This equation has <u>no</u> set-theoretic solution, even if we weaken equality $=$ to "isomorphism" $\cong$.

As a simpler example, consider

$$X = X \to \mathbb{2} \leftarrow \text{any 2-element set}$$

Cantor's theorem states that there is no set X such that $X \cong \mathbb{P}(X)$. Since $\mathbb{P}(X) \cong X \to \mathbb{2}$, it follows that $X = X \to \mathbb{2}$ has <u>no</u> solution.

> Higher-order programming constructs (functions and procedures as first-class citizens, i.e. values) lead to recursive domain equations which have no (non-trivial) set-theoretic solutions.

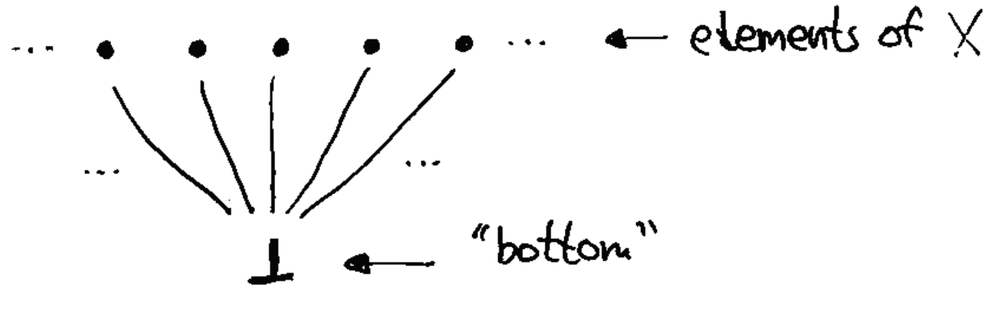
A FIRST STEP TO SCOTT-DOMAINS : LIFTED SETS

To avoid worrying about partial functions and undefined results, Scott introduced a special value $\perp$ (<u>bottom</u>) into each primitive semantic domain.

$$\perp \text{ represents } \begin{cases} \text{an undefined value;} \\ \text{an error value;} \\ \text{a non-terminating computation.} \end{cases}$$

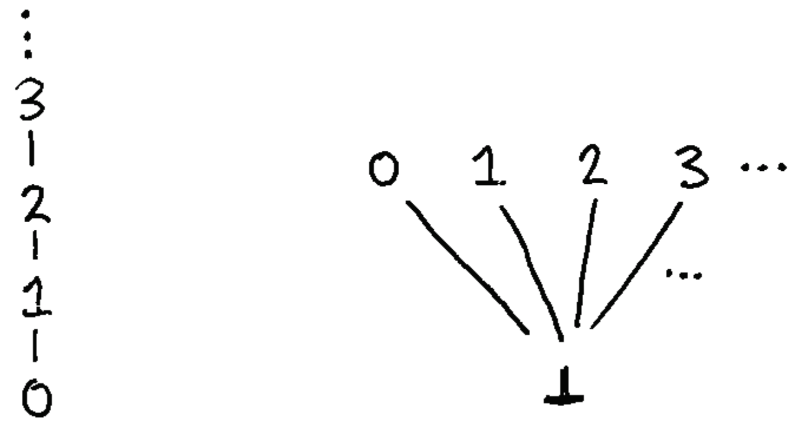
Given a set X the <u>flat domain</u> (or <u>lifted set</u>) $X_\perp$ is the set $X \cup \{\perp\}$, where $\perp \notin X$.

There is a natural "<u>information ordering</u>" $\sqsubseteq$ on $X_\perp$



<u>Formally</u>: $x \sqsubseteq y$ iff $(x = y \text{ or } x = \perp)$

<u>Note</u>: don't confuse the information ordering $\sqsubseteq$ on $\mathbb{N}_\perp$ with the standard ordering $\leqslant$ on $\mathbb{N}$.



e.g. $0 \leqslant 1$ but $0 \not\sqsubseteq 1$ and $1 \not\sqsubseteq 0$ (from an information content point of view, 0 and 1 are incomparible.)

<u>Example</u>: consider again the recursive definition

$$f(x) = f(x) + 1$$

If we define $\perp + x = \perp$ (adding an undefined value to any natural gives an undefined value) then this equation has a (unique) solution $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$:

$$f(x) = \perp$$

This result makes precise (at the <u>object level</u>) our intuition that evaluating $f(x)$ will not terminate.

PARTIAL ORDERINGS

When we consider functions $f : X \times Y \to Z$ with multiple arguments, we find that flat domains are no longer sufficient for our purposes.

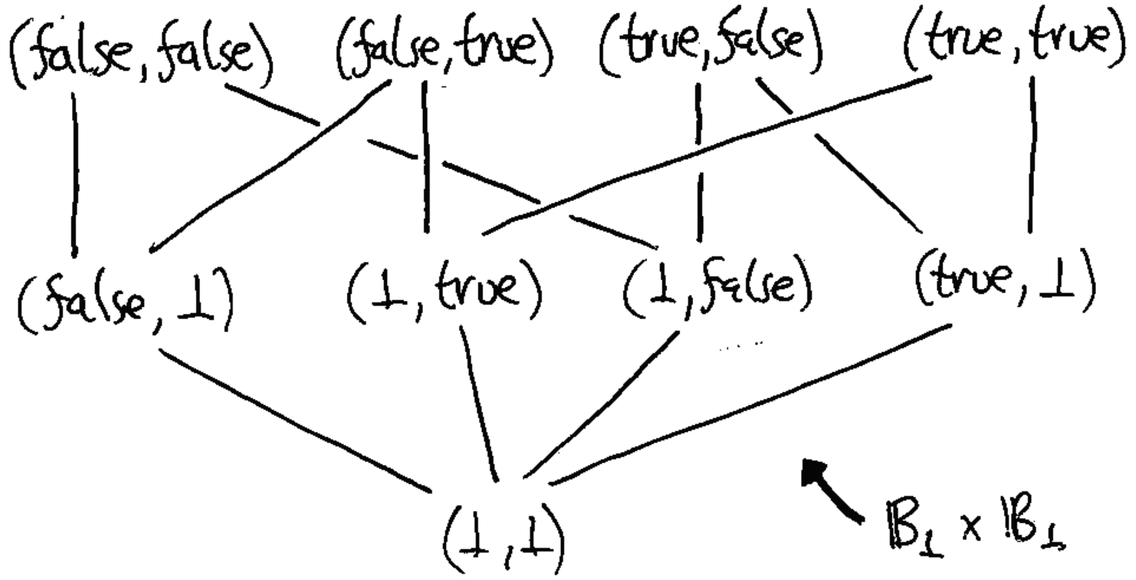
If X and Y are sets, their <u>Cartesian product</u> $X \times Y$ is the set $X \times Y = \{(x, y) \mid x \in X \text{ and } y \in Y\}$.

If X and Y are flat domains, there is a natural information ordering $\sqsubseteq$ on $X \times Y$, defined in terms of the orderings $\sqsubseteq$ on X and Y separately.

$$(x, y) \sqsubseteq (x', y') \quad \text{iff} \quad (x \sqsubseteq x' \text{ and } y \sqsubseteq y').$$

"the amount of information content in a pair of values is increased by increasing the information content of either (or both) of its component values."

semantic domains are no longer flat sets,



... they are partially ordered sets (posets):

$\forall x.\ x \sqsubseteq x$ reflexivity

$\forall x, y.\ x \sqsubseteq y$ and $y \sqsubseteq x$ implies $x = y$ antisymmetry

$\forall x, y, z.\ x \sqsubseteq y$ and $y \sqsubseteq z$ implies $x \sqsubseteq z$ transitivity

Fact: if A and B are posets, so is $A \times B$, with $(a,b) \sqsubseteq (a',b')$ iff $a \sqsubseteq a'$ and $b \sqsubseteq b'$.

Fact: if A and B are pointed (have a least element), so is $A \times B$, with $\perp_{A\times B} = (\perp_A, \perp_B)$.

$\boxed{\text{thesis: semantic domains are pointed posets.}}$

MONOTONIC FUNCTIONS

If we model semantic domains by (pointed) posets, then programs will be modelled by functions between posets. But not all functions are suitable.

It is natural to expect that the amount of information content in the output of a function grows as we increase the information in the input:

$$\forall x,y. \quad x \sqsubseteq y \text{ implies } f(x) \sqsubseteq f(y).$$

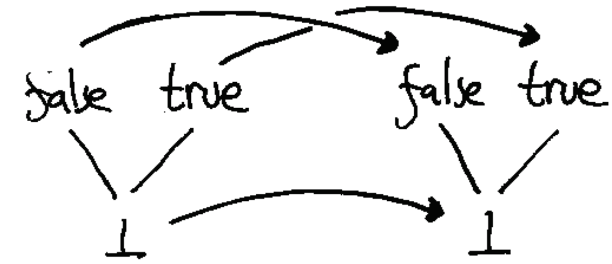
Such functions are called <u>monotonic</u>.

<u>Note</u>: monotonic functions "preserve" the poset structure, but are not required to preserve $\bot$. Functions for which $f(\bot) = \bot$ are called <u>strict</u>

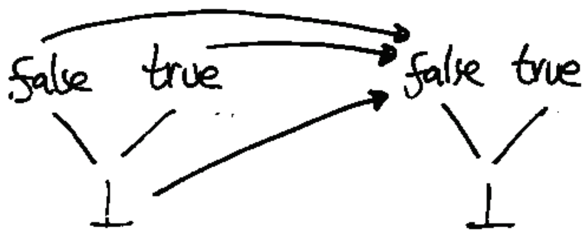
$$\boxed{\text{thesis: computable functions are monotonic.}}$$

Examples:

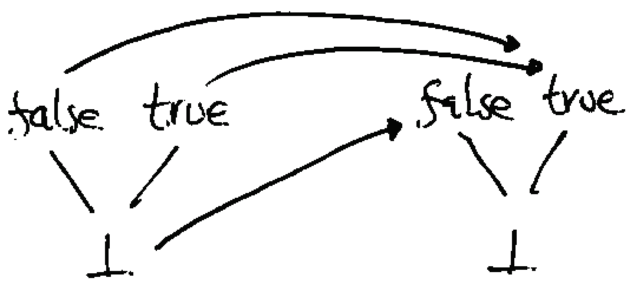
$f : \mathbb{B}_\perp \to \mathbb{B}_\perp$ monotonic?



✓ (identity fn)



✓ (constant fn)



✗ ($\perp \sqsubseteq$ false, but $f(\perp) \not\sqsubseteq f(\text{false})$.)

EXERCISES

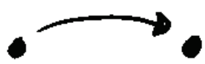
① Give a semantics to the "proc" construct:

$$\mathcal{E}[\![\text{proc } C]\!]\sigma \;=\; ? \qquad \text{(new)}$$

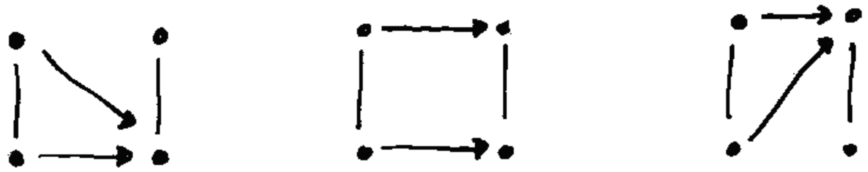
$$\mathcal{E}[\![I]\!]\sigma \;=\; ? \qquad \text{(revised)}$$

② Let $\mathbb{N}$ denote the chain  N points

* There is one monotonic function $\mathbb{1} \to \mathbb{1}$:



* There are three monotonic functions $\mathbb{2} \to \mathbb{2}$:



* Write down the monotonic functions $\mathbb{3} \to \mathbb{3}$.

* Write a simple recursive program to calculate the number of monotonic functions $\mathbb{N} \to \mathbb{M}$.