Q

Lecture Notes on

Denotational Semantics

Part II of the Computer Science Tripos 2009/10

Dr Marcelo Fiore
Cambridge University Computer Laboratory

Contents

Notes **ii**

1 Introduction **1**

1.1 Basic example of denotational semantics 2
1.2 Example: while-loops as fixed points 7
1.3 Exercises 12

2 Least Fixed Points **13**

2.1 Posets and monotone functions 13
2.1.1 Posets 13
2.1.2 Monotone functions 16
2.2 Least elements and pre-fixed points 16
2.3 Cpo's and continuous functions 19
2.3.1 Domains 19
2.3.2 Continuous functions 26
2.4 Tarski's fixed point theorem 28
2.5 Exercises 30

3 Constructions on Domains **31**

3.1 Flat domains 31
3.2 Products of domains 32
3.3 Function domains 35
3.4 Exercises 39

4 Scott Induction **41**

4.1 Chain-closed and admissible subsets 41
4.2 Examples 42
4.3 Building chain-closed subsets 44
4.3.1 Basic relations 44
4.3.2 Inverse image and substitution 45
4.3.3 Logical operations 46
4.4 Exercises 47

5 PCF **49**

5.1 Terms and types 49
5.2 Free variables, bound variables, and substitution 50
5.3 Typing 51
5.4 Evaluation 54
5.5 Contextual equivalence 58
5.6 Denotational semantics 60
5.7 Exercises 62

6 Denotational Semantics of PCF **65**
6.1 Denotation of types 65
6.2 Denotation of terms 66
6.3 Compositionality 73
6.4 Soundness 75
6.5 Exercises 76

7 Relating Denotational and Operational Semantics **77**
7.1 Formal approximation relations 77
7.2 Proof of the Fundamental Property of $\lhd$ 80
7.3 Extensionality 83
7.4 Exercises 86

8 Full Abstraction **87**
8.1 Failure of full abstraction 87
8.2 PCF+por 93
8.3 Fully abstract semantics for PCF 94
8.4 Exercises 95

Notes

These notes are designed to accompany 8–10 lectures on Denotational Semantics for Part II of the Cambridge University Computer Science Tripos. They are substantially those of Andrew Pitts (who lectured the course from 1997 to 1999) with some changes and additions by Glynn Winskel (who lectured the course from 2000 to 2007) and by Marcelo Fiore (who lectured the course from 2008). The material has been drawn from several different sources, including the books mentioned below, previous versions of this course, and similar courses at some other universities.

Recommended books

- Winskel, G. (1993). *The Formal Semantics of Programming Languages*. MIT Press.

  This is an excellent introduction to both the operational and denotational semantics of programming languages. As far as this course is concerned, the relevant chapters are 5, 8, 9, 10 (Sections 1 and 2), and 11.

- Tennent, R. D. (1991). *Semantics of Programming Languages*. Prentice-Hall.

  Parts I and II are relevant to this course.

Further reading

- Gunter, C. A. (1992). *Semantics of Programming Languages. Structures and Techniques*. MIT Press.

  This is a graduate-level text containing much material not covered in this course. As far as this course is concerned, the relevant chapters are 1, 2, and 4–6.

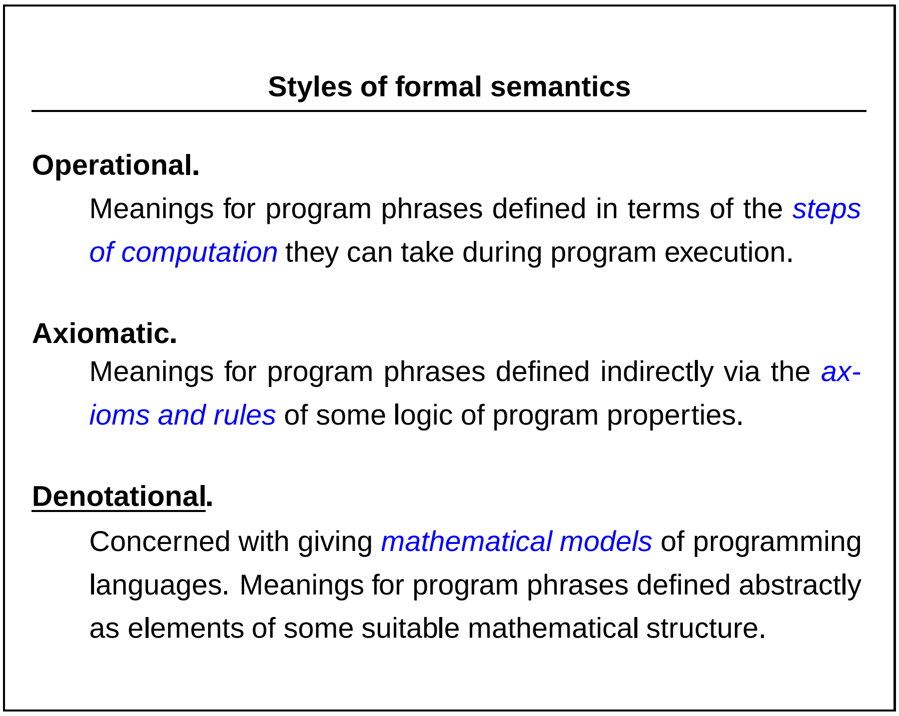
Feedback

Please fill out the online lecture course feedback form via the URL `https://lecture-feedback.cl.cam.ac.uk/feedback`.

Marcelo Fiore
`Marcelo.Fiore@cl.cam.ac.uk`

1 Introduction

Slide 1 gives a reminder of various approaches to giving formal semantics for programming languages. The operational approach was introduced in the Part IB course on **Semantics of Programming Languages** and the axiomatic approach is illustrated in the Part II course on **Specification and Verification I**. This course gives a brief introduction to some of the techniques of the denotational approach. One of the aims of Denotational Semantics is to specify programming language constructs in as abstract and implementation-independent way as possible: in this way one may gain insight into the fundamental concepts underlying programming languages, their inter-relationships, and (sometimes) new ways of realising those concepts in language designs. Of course, it is crucial to verify that denotational specifications of languages are implementable—in other words to relate denotational semantics to operational semantics: we will illustrate how this is done later in the course.



Styles of formal semantics

Operational.

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

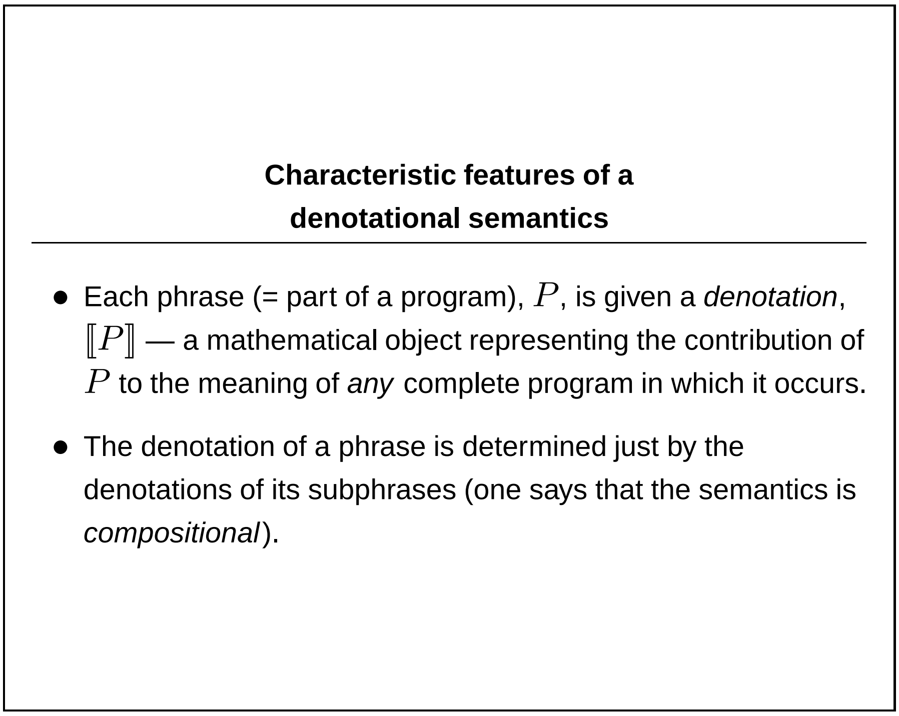
Axiomatic.

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.

Slide 1



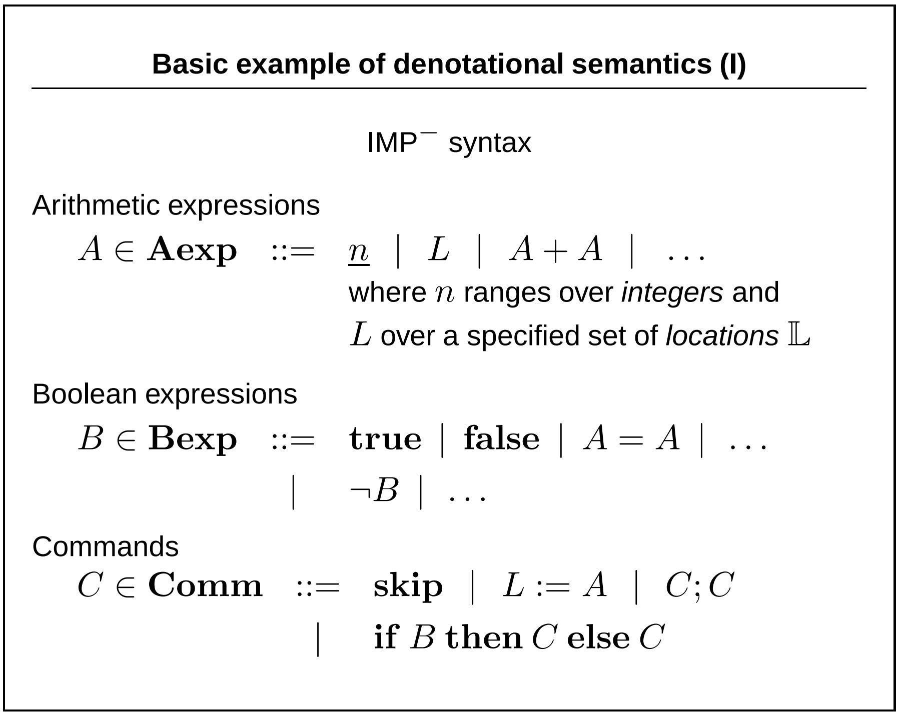
Characteristic features of a denotational semantics

- 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*).

Slide 2

1.1 Basic example of denotational semantics

Consider the basic programming language IMP^{-} over arithmetic and boolean expressions with control structures given by assignment, sequencing, and conditionals described on Slide 3.



Basic example of denotational semantics (I)

IMP$^-$ syntax

Arithmetic expressions

$$A \in \mathbf{Aexp} \quad ::= \quad \underline{n} \;\mid\; L \;\mid\; A + A \;\mid\; \ldots$$

where n ranges over *integers* and
L over a specified set of *locations* $\mathbb{L}$

Boolean expressions

$$\begin{aligned} B \in \mathbf{Bexp} \quad ::= \quad & \mathbf{true} \mid \mathbf{false} \mid A = A \mid \ldots \\ \mid \quad & \neg B \mid \ldots \end{aligned}$$

Commands

$$\begin{aligned} C \in \mathbf{Comm} \quad ::= \quad & \mathbf{skip} \;\mid\; L := A \;\mid\; C ; C \\ \mid \quad & \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C \end{aligned}$$

Slide 3

A *denotational semantics* for a programming language is constructed by giving a domain of interpretation to each of the program-phrase categories together with semantic functions that compositionally describe the meaning of the phrase-forming constructs. For IMP$^-$ this is done in Slides 4–10, and is easily implementable in SML.

Basic example of denotational semantics (II)

Semantic functions

$$\begin{aligned} \mathcal{A}: &\quad \mathbf{Aexp} \to (\mathit{State} \to \mathbb{Z}) \\ \mathcal{B}: &\quad \mathbf{Bexp} \to (\mathit{State} \to \mathbb{B}) \\ \mathcal{C}: &\quad \mathbf{Comm} \to (\mathit{State} \rightharpoonup \mathit{State}) \end{aligned}$$

where

$$\begin{aligned} \mathbb{Z} &= \{\ldots, -1, 0, 1, \ldots\} \\ \mathbb{B} &= \{\, \mathit{true}, \mathit{false} \,\} \\ \mathit{State} &= (\mathbb{L} \to \mathbb{Z}) \end{aligned}$$

Slide 4

Basic example of denotational semantics (III)

Semantic function $\mathcal{A}$

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

$$\mathcal{A}[\![L]\!] = \lambda s \in \mathit{State}.\, s(L)$$

$$\mathcal{A}[\![A_1 + A_2]\!] = \lambda s \in \mathit{State}.\, \mathcal{A}[\![A_1]\!](s) + \mathcal{A}[\![A_2]\!](s)$$

Slide 5

Basic example of denotational semantics (IV)

Semantic function $\mathcal{B}$

$$\begin{aligned}
\mathcal{B}[\![\mathbf{true}]\!] &= \lambda s \in \mathit{State}.\, \mathit{true} \\
\mathcal{B}[\![\mathbf{false}]\!] &= \lambda s \in \mathit{State}.\, \mathit{false} \\
\mathcal{B}[\![A_1 = A_2]\!] &= \lambda s \in \mathit{State}.\, \mathit{eq}\big(\mathcal{A}[\![A_1]\!](s), \mathcal{A}[\![A_2]\!](s)\big)
\end{aligned}$$

where $\mathit{eq}(a, a') = \begin{cases} \mathit{true} & \text{if } a = a' \\ \mathit{false} & \text{if } a \neq a' \end{cases}$

Slide 6

Basic example of denotational semantics (V)

Semantic function $\mathcal{C}$

$$[\![\mathbf{skip}]\!] = \lambda s \in \mathit{State}.\, s$$

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

Slide 7

A simple example of compositionality

Given partial functions $[\![C]\!], [\![C']\!] : \mathit{State} \rightharpoonup \mathit{State}$ and a function $[\![B]\!] : \mathit{State} \rightarrow \{\mathit{true}, \mathit{false}\}$, we can define

$$
\begin{aligned}
&[\![\textbf{if } B \textbf{ then } C \textbf{ else } C']\!] = \\
&\qquad\qquad \lambda s \in \mathit{State}.\, \mathit{if}\big([\![B]\!](s), [\![C]\!](s), [\![C']\!](s)\big)
\end{aligned}
$$

where

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

Slide 8

Basic example of denotational semantics (VI)

Semantic function $\mathcal{C}$

$$
[\![L := A]\!] \quad = \quad \lambda s \in \mathit{State}.\, \lambda \ell \in \mathbb{L}.\, \mathit{if}\big(\ell = L, [\![A]\!](s), s(\ell)\big)
$$

Slide 9

Denotational semantics of sequential composition

Denotation of sequential composition $C; C'$ of two commands

$$[\![C; C']\!] = [\![C']\!] \circ [\![C]\!] = \lambda s \in \mathit{State}.\, [\![C']\!]\big([\![C]\!](s)\big)$$

given by composition of the partial functions from states to states $[\![C]\!], [\![C']\!] : \mathit{State} \rightharpoonup \mathit{State}$ which are the denotations of the commands.

Cf. operational semantics of sequential composition:

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

Slide 10

1.2 Example: $\mathbf{while}$-loops as fixed points

The requirement of *compositionality* mentioned on Slide 2 is quite a tough one. It means that the collection of mathematical objects we use to give denotations to program phases has to be sufficiently rich that it supports operations for modelling all the phrase-forming constructs of the programming language in question. Some phrase-forming constructs are easy to deal with, others less so. For example, conditional expressions involving state-manipulating commands can be given a denotational semantics in terms of a corresponding branching function applied to the denotations of the immediate subexpressions: see Slide 8. Similarly, the denotational semantics of the sequential composition of commands can be given by the operation of composition of partial functions from states to states, as shown on Slide 10.

We now proceed to consider the denotational semantics of the basic programming language IMP, obtained by extending IMP$^-$ with $\mathbf{while}$-loops:

$$C \in \mathbf{Comm} \quad ::= \quad \ldots \mid \mathbf{while}\ B\ \mathbf{do}\ C$$

However, this looping construct is not so easy to explain compositionally!

The transition semantics of a $\mathbf{while}$-loop

$$\langle \mathbf{while}\ B\ \mathbf{do}\ C, s\rangle \rightarrow \langle \mathbf{if}\ B\ \mathbf{then}\ C; (\mathbf{while}\ B\ \mathbf{do}\ C)\ \mathbf{else}\ \mathbf{skip}, s\rangle$$

suggests that its denotation as a partial function from states to states should satisfy

(1) $\llbracket \mathbf{while}\ B\ \mathbf{do}\ C \rrbracket = \llbracket \mathbf{if}\ B\ \mathbf{then}\ C; (\mathbf{while}\ B\ \mathbf{do}\ C)\ \mathbf{else}\ \mathbf{skip} \rrbracket.$

Note that this cannot be used directly to define $\llbracket \mathbf{while}\ B\ \mathbf{do}\ C \rrbracket$, since the right-hand side contains as a subphrase the very phrase whose denotation we are trying to define. Using the denotational semantics of sequential composition and **if** (and using the fact that the denotation of **skip** is the identity function $\lambda s \in \mathit{State}.s$), (1) amounts to saying that $\llbracket \mathbf{while}\ B\ \mathbf{do}\ C \rrbracket$ should be a solution of the *fixed point equation* given on Slide 11.

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 $b : \mathit{State} \to \{\mathit{true}, \mathit{false}\}$ and $c : \mathit{State} \rightharpoonup \mathit{State}$, we define

$$f_{b,c} : (\mathit{State} \rightharpoonup \mathit{State}) \to (\mathit{State} \rightharpoonup \mathit{State})$$

as

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

- 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 $\llbracket \mathbf{while}\ B\ \mathbf{do}\ C \rrbracket$?

Slide 11

Such fixed point equations arise very often in giving denotational semantics to languages with recursive features. Beginning with Dana Scott's pioneering work in the late 60's, a mathematical theory called *domain theory* has been developed to provide a setting in which not only can we always find solutions for the fixed point equations arising from denotational semantics, but also we can pick out solutions that are minimal in a suitable sense—and this turns out to ensure a good match between denotational and operational semantics. The key idea is to consider a partial order between the mathematical objects used as denotations—this partial order expresses the fact that one object is *approximated by*, or *carries more information than*, or is *more well-defined than* another one below it in the ordering. Then the minimal solution of a fixpoint equation can be constructed as the limit of an increasing chain of approximations to the solution. These ideas will be made mathematically precise and general in the next section; but first we illustrate how they work out concretely for the particular problem on Slide 11.

For definiteness, let us consider the particular **while**-loop

(2) $$\mathbf{while}\ X > 0\ \mathbf{do}\ (Y := X * Y\ ;\ X := X - 1)$$

where X and Y are two distinct integer storage locations (variables) and where the set of locations $\mathbb{L}$ is $\{\, X, Y \,\}$.

In this case we can just take a state to be an assignment $[X \mapsto x, Y \mapsto y]$ with $x, y \in \mathbb{Z}$, recording the current contents of the locations X and Y respectively. Thus, $\mathit{State} = (\mathbb{L} \to \mathbb{Z})$.

We are trying to define the denotation of (2) as a partial function

$$w : \mathit{State} \rightharpoonup \mathit{State}$$

that should be a solution to the fixed-point equation

$$w = f_{[\![X>0]\!],[\![Y:=X*Y;X:=X-1]\!]}(w)$$

on Slide 11.

For the particular boolean expression $B = (X > 0)$ and command $C = (Y := X * Y\ ;\ X := X - 1)$, the function $f_{[\![B]\!],[\![C]\!]}$ coincides with the function f defined on Slide 12.

$$[\![\mathbf{while}\ X > 0\ \mathbf{do}\ (Y := X * Y\ ;\ X := X - 1)]\!]$$

Let

$\mathit{State} \overset{\text{def}}{=} (\mathbb{L} \to \mathbb{Z})$	integer assignments to locations
$D \overset{\text{def}}{=} (\mathit{State} \rightharpoonup \mathit{State})$	partial functions on states

For $[\![\mathbf{while}\ X > 0\ \mathbf{do}\ Y := X * Y\ ;\ X := X - 1]\!] \in D$ we seek a minimal solution to $w = f(w)$, where $f : D \to D$ is defined by:

$$\begin{aligned} & f(w)\big([X \mapsto x, Y \mapsto y]\big) \\ & \quad = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w\big([X \mapsto x-1, Y \mapsto x * y]\big) & \text{if } x > 0. \end{cases} \end{aligned}$$

Slide 12

$$D \overset{\text{def}}{=} (State \rightharpoonup State)$$

- **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 $w \in D$).

Slide 13

Consider the partial order, $\sqsubseteq$, between the elements of $D = (State \rightharpoonup State)$ given on Slide 13. Note that $\sqsubseteq$ does embody the kind of 'information ordering' mentioned above: if $w \sqsubseteq w'$, then w' agrees with w wherever the latter is defined, but it may be defined at some other arguments as well. Note also that D contains an element which is least with respect to this partial order: for the totally undefined partial function, which we will write as $\perp$, satisfies $\perp \sqsubseteq w$ for any $w \in D$.

Starting with $\perp$, we apply the function f over and over again to build up a sequence of partial functions $w_0, w_1, w_2, \ldots$:

$$\begin{cases} w_0 & \overset{\text{def}}{=} \perp \\ w_{n+1} & \overset{\text{def}}{=} f(w_n). \end{cases}$$

Using the definition of f on Slide 12, one finds that

$$w_1[X \mapsto x, Y \mapsto y] = f(\perp)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ \text{undefined} & \text{if } x \geq 1 \end{cases}$$

$$w_2[X \mapsto x, Y \mapsto y] = f(w_1)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto y] & \text{if } x = 1 \\ \text{undefined} & \text{if } x \geq 2 \end{cases}$$

$$w_3[X \mapsto x, Y \mapsto y] = f(w_2)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto y] & \text{if } x = 1 \\ [X \mapsto 0, Y \mapsto 2 * y] & \text{if } x = 2 \\ \text{undefined} & \text{if } x \geq 3 \end{cases}$$

$$w_4[X \mapsto x, Y \mapsto y] = f(w_3)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto y) & \text{if } x = 1 \\ [X \mapsto 0, Y \mapsto 2 * y] & \text{if } x = 2 \\ [X \mapsto 0, Y \mapsto 6 * y] & \text{if } x = 3 \\ \text{undefined} & \text{if } x \geq 4 \end{cases}$$

and in general

$$w_n[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto (!x) * y] & \text{if } 0 < x < n \\ \text{undefined} & \text{if } x \geq n \end{cases}$$

where as usual, $!x$ is the factorial of x.

Thus we get an increasing sequence of partial functions

$$w_0 \sqsubseteq w_1 \sqsubseteq w_2 \sqsubseteq \ldots \sqsubseteq w_n \sqsubseteq \ldots$$

defined on larger and larger sets of states (x, y) and agreeing where they are defined. The union of all these partial functions is the element $w_\infty \in D$ given by

$$w_\infty[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto (!x) * y] & \text{if } x > 0. \end{cases}$$

Note that w_∞ is a fixed point of the function f, since for all $[X \mapsto x, Y \mapsto y]$ we have

$$\begin{aligned} f(w_\infty)[X \mapsto x, Y \mapsto y] &= \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w_\infty[X \mapsto x - 1, Y \mapsto x * y] & \text{if } x > 0 \end{cases} && \text{(by definition of } f\text{)} \\ &= \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto 1 * y] & \text{if } x = 1 \\ [X \mapsto 0, Y \mapsto !(x-1) * x * y] & \text{if } x > 1 \end{cases} && \text{(by definition of } w_\infty\text{)} \\ &= w_\infty[X \mapsto x, Y \mapsto y] \ . \end{aligned}$$

In fact one can show that w_∞ is the *least* fixed point of f, in the sense that for all $w \in D$

(3) $$w = f(w) \quad \Rightarrow \quad w_\infty \sqsubseteq w.$$

This least fixed point w_∞ is what we take as the denotation of

$$\mathbf{while}\ X > 0\ \mathbf{do}\ (Y := X * Y\ ;\ X := X - 1).$$

Its construction is an instance of Tarski's Fixed Point Theorem to be proved in the next section. Note also that w_∞ is indeed the function from states to states that we get from the structural operational semantics of the command $\mathbf{while}\ X > 0\ \mathbf{do}\ (Y := X * Y\ ;\ X := X - 1)$, as given in the Part IB course on **Semantics of Programming Languages**.

1.3 Exercises

Exercise 1.3.1. Implement the denotational semantics of IMP$^-$ in SML.

Exercise 1.3.2. Consider the function

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

defined on Slide 11.
(1) Show by induction on n that

$$f_{b,c}{}^n(\bot) = \lambda s \in State. \begin{cases} c^k(s) & \text{if } 0 \le k < n \text{ is such that } b(c^i(s)) = \mathit{true} \\ & \text{for all } 0 \le i < k \text{ and } b(c^k(s)) = \mathit{false} \\ \text{undefined} & \text{if } b(c^i(s)) = \mathit{true} \text{ for all } 0 \le i < n \end{cases}$$

(2) Let $w_{b,c} : State \rightharpoonup State$ be the partial function defined as

$$w_{b,c} \overset{\text{def}}{=} \lambda s \in State. \begin{cases} c^k(s) & \text{if } k \ge 0 \text{ is such that } b(c^i(s)) = \mathit{true} \\ & \text{for all } 0 \le i < k \text{ and } b(c^k(s)) = \mathit{false} \\ \text{undefined} & \text{if } b(c^i(s)) = \mathit{true} \text{ for all } i \ge 0 \end{cases}$$

Show that $w_{b,c}$ satisfies the fixed-point equation

$$w_{b,c} = f_{b,c}(w_{b,c}) \ .$$

(3) Describe the function $f_{b,c}$ for $b = [\![\mathbf{true}]\!] = \lambda s \in State.\mathit{true}$ and $c = [\![\mathbf{skip}]\!] = \lambda s \in State.s$. Which partial functions from states to states are fixed points of this $f_{b,c}$? What is its least fixed point (with respect to the $\sqsubseteq$ ordering defined above)? Does this least fixed point agree with the partial function from states to states determined by the operational semantics of $\mathbf{while\ true\ do\ skip}$?

Exercise 1.3.3. Show that the relation $\sqsubseteq$ defined on Slide 13 is a partial order with least element $\bot$.

Exercise 1.3.4. Prove the statement (3). More generally, with the definitions of Slide 13 and Exercise 1.3.2, prove that

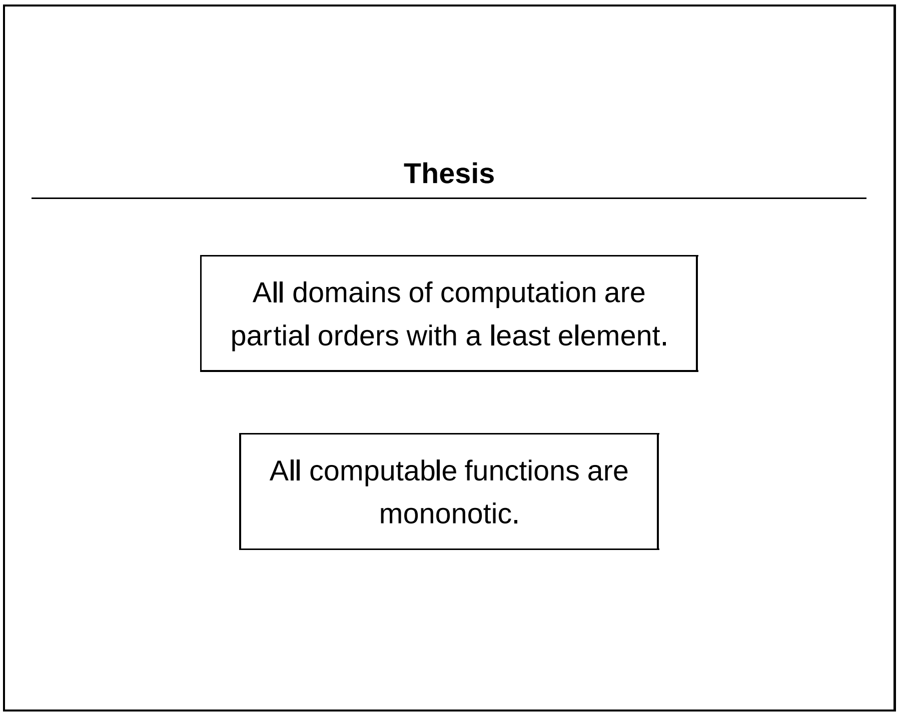
$$w = f_{b,c}(w) \implies w_{b,c} \sqsubseteq w$$

for all $w \in (State \rightharpoonup State)$.

2 Least Fixed Points

This section introduces a mathematical theory, *domain theory*, which amongst other things provides a general framework for constructing the least fixed points used in the denotational semantics of various programming language features. The theory was introduced by Dana Scott.

2.1 Posets and monotone functions



Thesis

All domains of computation are partial orders with a least element.

All computable functions are mononotic.

Slide 14

2.1.1 Posets

Domain theory makes use of partially ordered sets satisfying certain completeness properties. The definition of a *partial order* is recalled on Slide 15. D is called the *underlying set* of the poset $(D, \sqsubseteq)$. Most of the time we will refer to posets just by naming their underlying sets and use the same symbol $\sqsubseteq$ to denote the partial order in a variety of different posets.

Partially ordered sets

A binary relation $\sqsubseteq$ on a set D is a *partial order* iff it is

reflexive: $\forall d \in D.\ d \sqsubseteq d$

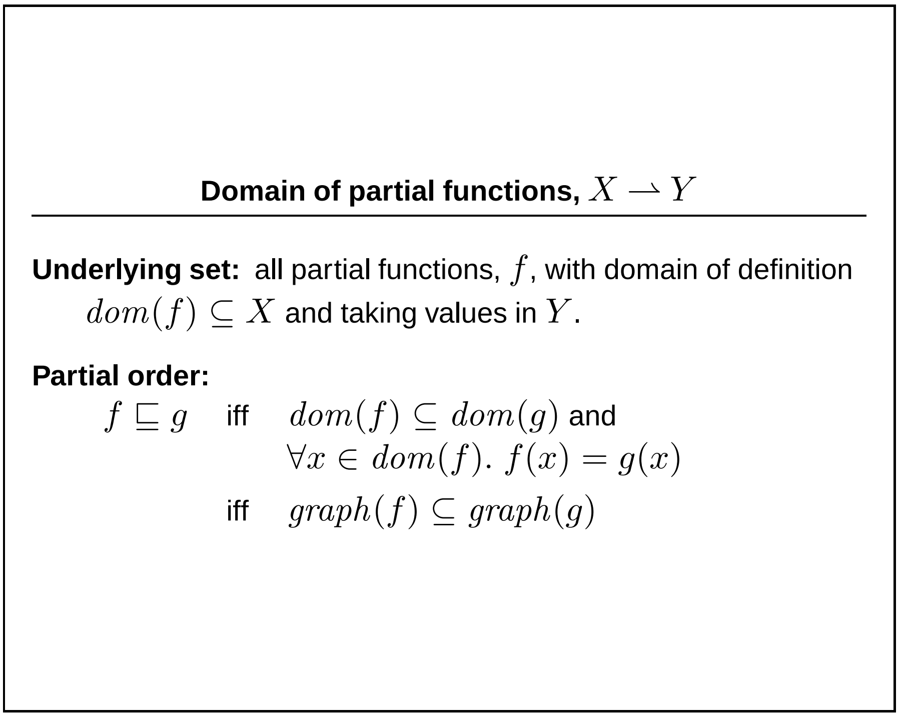
transitive: $\forall d, d', d'' \in D.\ d \sqsubseteq d' \sqsubseteq d'' \Rightarrow d \sqsubseteq d''$

anti-symmetric: $\forall d, d' \in D.\ d \sqsubseteq d' \sqsubseteq d \Rightarrow d = d'$.

Such a pair $(D, \sqsubseteq)$ is called a *partially ordered set*, or *poset*.

Slide 15

Example 2.1.1. The set $(X \rightharpoonup Y)$ of all partial functions from a set X to a set Y can be made into a poset, as indicated on Slide 16. It was this domain for the case $X = Y = \mathit{State}$ (some set of states) that we used for the denotation of commands in Section 1.2.



Domain of partial functions, $X \rightharpoonup Y$

Underlying set: all partial functions, f, with domain of definition $dom(f) \subseteq X$ and taking values in Y.

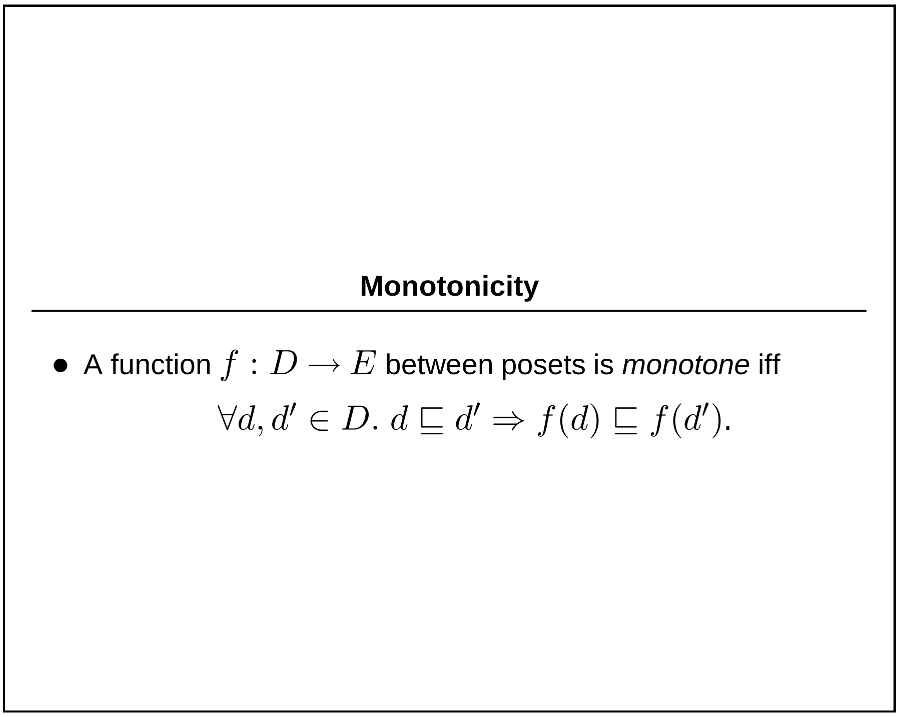
Partial order:

$$\begin{array}{lll} f \sqsubseteq g & \text{iff} & dom(f) \subseteq dom(g) \text{ and} \\ & & \forall x \in dom(f).\ f(x) = g(x) \\ & \text{iff} & graph(f) \subseteq graph(g) \end{array}$$

Slide 16

2.1.2 Monotone functions

The notion of mapping between posets is given in Slide 17.



Monotonicity

- A function $f : D \to E$ between posets is *monotone* iff

$$\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d').$$

Slide 17

Example 2.1.2. Given posets D and E, for each $e \in E$ it is easy to see that the *constant function* $D \to E$ *with value* e, $\lambda d \in D\,.\,e$, is monotone.

Example 2.1.3. When D is the domain of partial functions $(State \rightharpoonup State)$ (cf. Slide 16), the function $f_{b,c} : D \to D$ defined on Slide 11 in connection with the denotational semantics of **while**-loops is a monotone function. We leave the verification of this as an exercise.

2.2 Least elements and pre-fixed points

Definition 2.2.1. Suppose that D is a poset and that S is a subset of D. An element $d \in S$ is the *least* element of S if it satisfies

$$\forall x \in S.\ d \sqsubseteq x\ \ .$$

Note that because $\sqsubseteq$ is anti-symmetric, S has at most one least element. Note also that a poset may not have least element. For example, $\mathbb{Z}$ with its usual partial order does not have a least element.

A *fixed point* for a function $f : D \to D$ is by definition an element $d \in D$ satisfying $f(d) = d$. If D is a poset, we can consider a weaker notion, of *pre-fixed point*, as defined on Slide 18.

Pre-fixed points

Let D be a poset and $f : D \to D$ be a function.

An element $d \in D$ is a *pre-fixed point of* f if it satisfies $f(d) \sqsubseteq d$.

The *least pre-fixed point* of f, if it exists, will be written

$$\boxed{\mathit{fix}(f)}$$

It is thus (uniquely) specified by the two properties:

(lfp1) $\quad f(\mathit{fix}(f)) \sqsubseteq \mathit{fix}(f)$

(lfp2) $\quad \forall d \in D.\ f(d) \sqsubseteq d \ \Rightarrow\ \mathit{fix}(f) \sqsubseteq d.$

Slide 18

Proof principle

Let D be a poset and let $f : D \to D$ be a function with a least pre-fixed point $\mathit{fix}(f) \in D$.

For all $x \in D$, to prove that $\mathit{fix}(f) \sqsubseteq x$ it is enough to establish that $f(x) \sqsubseteq x$.

Slide 19

Proposition 2.2.2. *Suppose D is a poset and $f : D \to D$ is a function possessing a least pre-fixed point, $\mathit{fix}(f)$, as defined on Slide* 18.

Provided f is monotone, $\mathit{fix}(f)$ *is in particular a fixed point for f (and hence is the least element of the set of fixed points for f).*

Proof. By definition, $\mathit{fix}(f)$ satisfies property (lfp1) on Slide 18. If f is monotone (Slide 17) we can apply f to both sides of (lfp1) to conclude that

$$f(f(\mathit{fix}(f))) \sqsubseteq f(\mathit{fix}(f)).$$

Then applying property (lfp2) with $d = f(\mathit{fix}(f))$, we get that

$$\mathit{fix}(f) \sqsubseteq f(\mathit{fix}(f)).$$

Combining this with (lfp1) and the anti-symmetry property of the partial order $\sqsubseteq$, we get $f(\mathit{fix}(f)) = \mathit{fix}(f)$, as required. □

2.3 Cpo’s and continuous functions

Thesis*

All domains of computation are complete partial orders with a least element.

All computable functions are continuous.

Slide 20

2.3.1 Domains

Definition 2.3.1. (i) If it exists, we will write the least element of a poset D as $\bot_D$, or just $\bot$ when D is understood from the context. Thus $\bot$ is uniquely determined by the property:

$$\forall d \in D.\ \bot \sqsubseteq d.$$

The least element of a poset is sometimes called its *bottom* element.

(ii) A countable, increasing *chain* in a poset D is a sequence of elements of D satisfying

$$d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \dots$$

An *upper bound* for the chain is any $d \in D$ satisfying $\forall n \in \mathbb{N}.\ d_n \sqsubseteq d$. If it exists, the *least upper bound*, or *lub*, of the chain will be written as

$$\bigsqcup_{n \geq 0} d_n.$$

Thus by definition:

- $\forall m \in \mathbb{N}.\ d_m \sqsubseteq \bigsqcup_{n \geq 0} d_n$.
- For any $d \in D$, if $\forall m \in \mathbb{N}.\ d_m \sqsubseteq d$, then $\bigsqcup_{n \geq 0} d_n \sqsubseteq d$.

Remark 2.3.2. The following points should be noted.

(i) We will not need to consider uncountable, or decreasing chains in a poset: so a ‘chain’ will always mean a countable, increasing chain.

(ii) Like the least element of any subset of a poset, the lub of a chain is unique if it exists. (It does not have to exist: for example the chain $0 \leq 1 \leq 2 \leq \ldots$ in $\mathbb{N}$ has no upper bound, hence no lub.)

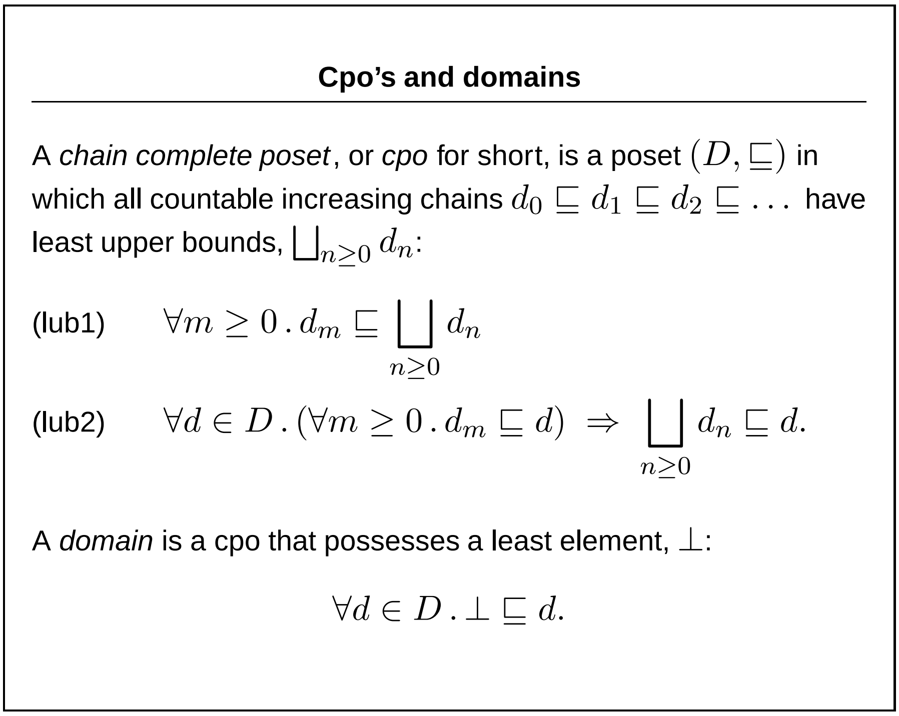
(iii) A least upper bound is sometimes called a *supremum*. Some other common notations for $\bigsqcup_{n \geq 0} d_n$ are:

$$\bigsqcup_{n=0}^{\infty} d_n \qquad \text{and} \qquad \bigsqcup \{d_n \mid n \geq 0\}\ .$$

(iv) The elements of a chain do not necessarily have to be distinct. In particular, we say that a chain $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$ is *eventually constant* if for some $N \in \mathbb{N}$ it is the case that $\forall n \geq N.\ d_n = d_N$. Note that in this case $\bigsqcup_{n \geq 0} d_n = d_N$.

(v) If we discard any finite number of elements at the beginning of a chain, we do not affect its set of upper bounds and hence do not change its lub:

$$\bigsqcup_{n \geq 0} d_n = \bigsqcup_{n \geq 0} d_{N+n}, \quad \text{for any } N \in \mathbb{N}.$$



Cpo's and domains

A *chain complete poset*, or *cpo* for short, is a poset $(D, \sqsubseteq)$ in which all countable increasing chains $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \dots$ have least upper bounds, $\bigsqcup_{n \geq 0} d_n$:

(lub1) $$\forall m \geq 0 \,.\, d_m \sqsubseteq \bigsqcup_{n \geq 0} d_n$$

(lub2) $$\forall d \in D \,.\, (\forall m \geq 0 \,.\, d_m \sqsubseteq d) \;\Rightarrow\; \bigsqcup_{n \geq 0} d_n \sqsubseteq d.$$

A *domain* is a cpo that possesses a least element, $\perp$:

$$\forall d \in D \,.\, \perp \sqsubseteq d.$$

Slide 21

In this course we will be concerned with posets enjoying certain completeness properties, as defined on Slide 21. It should be noted that the term 'domain' is used rather loosely in the literature on denotational semantics: there are many different kinds of domain, enjoying various extra order-theoretic properties over and above the rather minimal ones of chain-completeness and possession of a least element that we need for this course.

Example 2.3.3. The set $(X \rightharpoonup Y)$ of all partial functions from a set X to a set Y can be made into a domain, as indicated on Slide 22. It was this domain for the case $X = Y = \mathit{State}$ (some set of states) that we used for the denotation of commands in Section 1.2. Note that the f which is claimed to be the lub of $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \dots$ on Slide 22 is a well-defined partial function because the f_n agree where they are defined. We leave it as an exercise to check that this f is indeed the least upper bound of $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \dots$ in the poset $(X \rightharpoonup Y, \sqsubseteq)$.

Domain of partial functions, $X \rightharpoonup Y$

Underlying set: all partial functions, f, with domain of definition $dom(f) \subseteq X$ and taking values in Y.

Partial order:

$$\begin{array}{lll} f \sqsubseteq g & \text{iff} & dom(f) \subseteq dom(g) \text{ and} \\ & & \forall x \in dom(f).\ f(x) = g(x) \\ & \text{iff} & graph(f) \subseteq graph(g) \end{array}$$

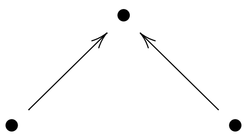
Lub of chain $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \ldots$ is the partial function f with $dom(f) = \bigcup_{n \geq 0} dom(f_n)$ and

$$f(x) = \begin{cases} f_n(x) & \text{if } x \in dom(f_n), \text{ some } n \\ \text{undefined} & \text{otherwise} \end{cases}$$

Least element $\perp$ is the totally undefined partial function.

Slide 22

Example 2.3.4. Any poset $(D, \sqsubseteq)$ whose underlying set D is finite is a cpo. For in such a poset any chain is eventually constant and we noted in Remark 2.3.2(iv) that such a chain always possesses a lub. Of course, a finite poset need not have a least element, and hence need not be a domain—for example, consider the poset with Hasse diagram

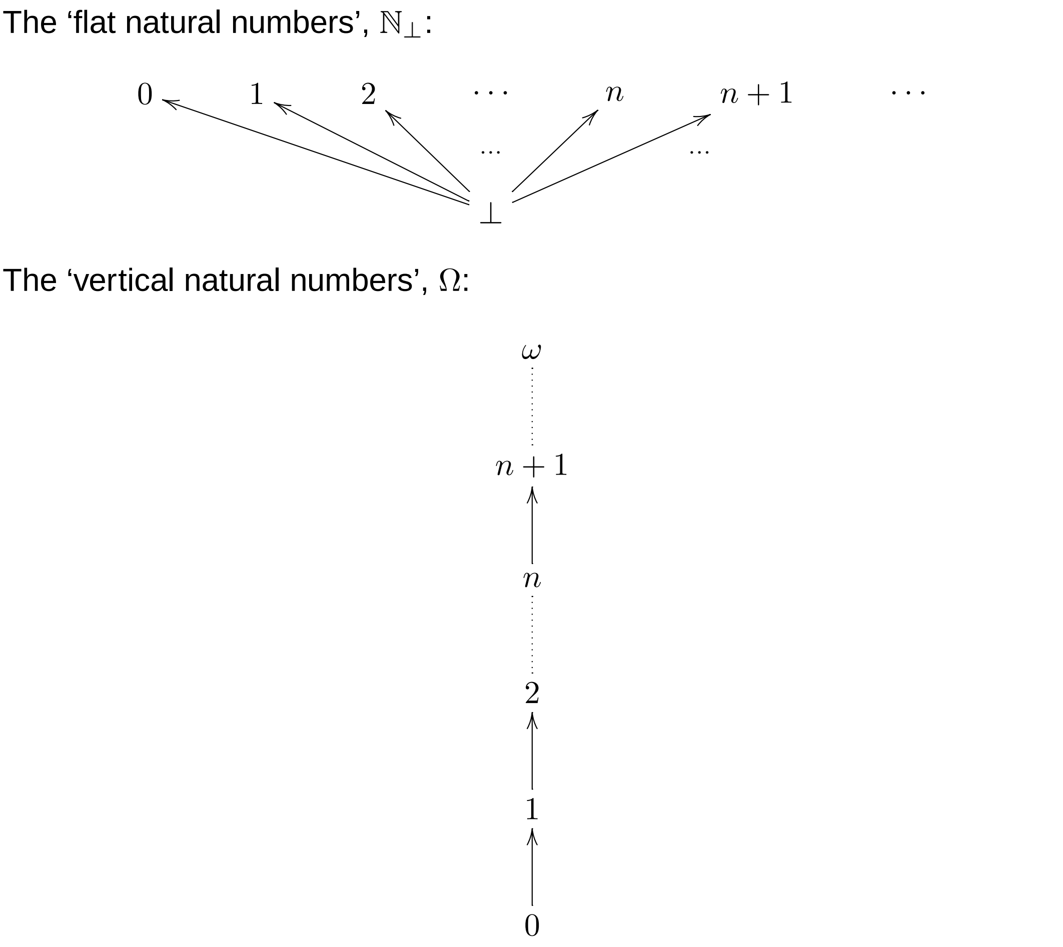


(The *Hasse diagram* of a poset is the directed graph whose vertices are the elements of the underlying set of the poset and in which there is an edge from vertex x to vertex y iff $x \neq y$ and $\forall z.\ (x \sqsubseteq z \ \&\ z \sqsubseteq y) \Rightarrow (z = x \vee z = y)$.)

Figure 1 shows two very simple, but infinite domains. Here are two examples of posets that are not cpos.

Example 2.3.5. The set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ equipped with the usual partial order, $\leq$, is not a cpo. For the increasing chain $0 \leq 1 \leq 2 \leq \ldots$ has no upper bound in $\mathbb{N}$.

Example 2.3.6. Consider a modified version of the second example in Figure 1 in which we adjoin two different upper bounds, $\omega_1 \neq \omega_2$, for $\mathbb{N}$. In other words,



The 'flat natural numbers', $\mathbb{N}_\perp$:

The 'vertical natural numbers', Ω:

Figure 1: Two domains

consider $D \overset{\text{def}}{=} \mathbb{N} \cup \{\omega_1, \omega_2\}$ with partial order $\sqsubseteq$ defined by:

$$d \sqsubseteq d' \overset{\text{def}}{\Leftrightarrow} \begin{cases} & d, d' \in \mathbb{N} \;\; \& \;\; d \leq d', \\ \text{or} & d \in \mathbb{N} \;\; \& \;\; d' \in \{\omega_1, \omega_2\}, \\ \text{or} & d = d' = \omega_1, \\ \text{or} & d = d' = \omega_2. \end{cases}$$

Then the increasing chain $0 \sqsubseteq 1 \sqsubseteq 2 \sqsubseteq \ldots$ in D has two upper bounds (ω_1 and ω_2), but no least one (since $\omega_1 \not\sqsubseteq \omega_2$ and $\omega_2 \not\sqsubseteq \omega_1$). So $(D, \sqsubseteq)$ is not a cpo.

Some properties of lubs of chains

Let D be a cpo.

1. For $d \in D$, $\bigsqcup_n d = d$.
2. For every chain $d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n \sqsubseteq \ldots$ in D,

   $$\bigsqcup_n d_n \; = \; \bigsqcup_n d_{N+n}$$

   for all $N \in \mathbb{N}$.
3. For every pair of chains $d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n \sqsubseteq \ldots$ and $e_0 \sqsubseteq e_1 \sqsubseteq \ldots \sqsubseteq e_n \sqsubseteq \ldots$ in D,

   if $d_n \sqsubseteq e_n$ for all $n \in \mathbb{N}$ then $\bigsqcup_n d_n \;\sqsubseteq\; \bigsqcup_n e_n$.

Slide 23

Diagonalising a double chain

Lemma. *Let D be a cpo. Suppose that the doubly-indexed family of elements $d_{m,n} \in D$ ($m, n \geq 0$) satisfies*

$$(\dagger) \qquad m \leq m' \;\&\; n \leq n' \;\Rightarrow\; d_{m,n} \sqsubseteq d_{m',n'}.$$

Then

$$\bigsqcup_{n\geq 0} d_{0,n} \;\sqsubseteq\; \bigsqcup_{n\geq 0} d_{1,n} \;\sqsubseteq\; \bigsqcup_{n\geq 0} d_{2,n} \;\sqsubseteq\; \ldots$$

and

$$\bigsqcup_{m\geq 0} d_{m,0} \;\sqsubseteq\; \bigsqcup_{m\geq 0} d_{m,1} \;\sqsubseteq\; \bigsqcup_{m\geq 0} d_{m,3} \;\sqsubseteq\; \ldots$$

Moreover

$$\bigsqcup_{m\geq 0}\left(\bigsqcup_{n\geq 0} d_{m,n}\right) = \bigsqcup_{k\geq 0} d_{k,k} = \bigsqcup_{n\geq 0}\left(\bigsqcup_{m\geq 0} d_{m,n}\right) \quad .$$

Slide 24

Proof of the Lemma on Slide 24. We make use of the defining properties of lubs of chains—(lub1) and (lub2) on Slide 21. First note that if $m \leq m'$ then

$$\begin{aligned} d_{m,n} &\sqsubseteq d_{m',n} && \text{by property } (\dagger) \text{ of the } d_{m,n} \\ &\sqsubseteq \bigsqcup_{n'\geq 0} d_{m',n'} && \text{by (lub1)} \end{aligned}$$

for all $n \geq 0$, and hence $\bigsqcup_{n\geq 0} d_{m,n} \sqsubseteq \bigsqcup_{n'\geq 0} d_{m',n'}$ by (lub2). Thus we do indeed get a chain of lubs

$$\bigsqcup_{n\geq 0} d_{0,n} \sqsubseteq \bigsqcup_{n\geq 0} d_{1,n} \sqsubseteq \bigsqcup_{n\geq 0} d_{2,n} \sqsubseteq \ldots$$

and can form its lub, $\bigsqcup_{m\geq 0} \bigsqcup_{n\geq 0} d_{m,n}$. Using property (lub1) twice we have

$$d_{k,k} \sqsubseteq \bigsqcup_{n\geq 0} d_{k,n} \sqsubseteq \bigsqcup_{m\geq 0} \bigsqcup_{n\geq 0} d_{m,n}$$

for each $k \geq 0$, and hence by (lub2)

$$(4) \qquad \bigsqcup_{k\geq 0} d_{k,k} \sqsubseteq \bigsqcup_{m\geq 0} \bigsqcup_{n\geq 0} d_{m,n}.$$

Conversely, for each $m, n \geq 0$, note that

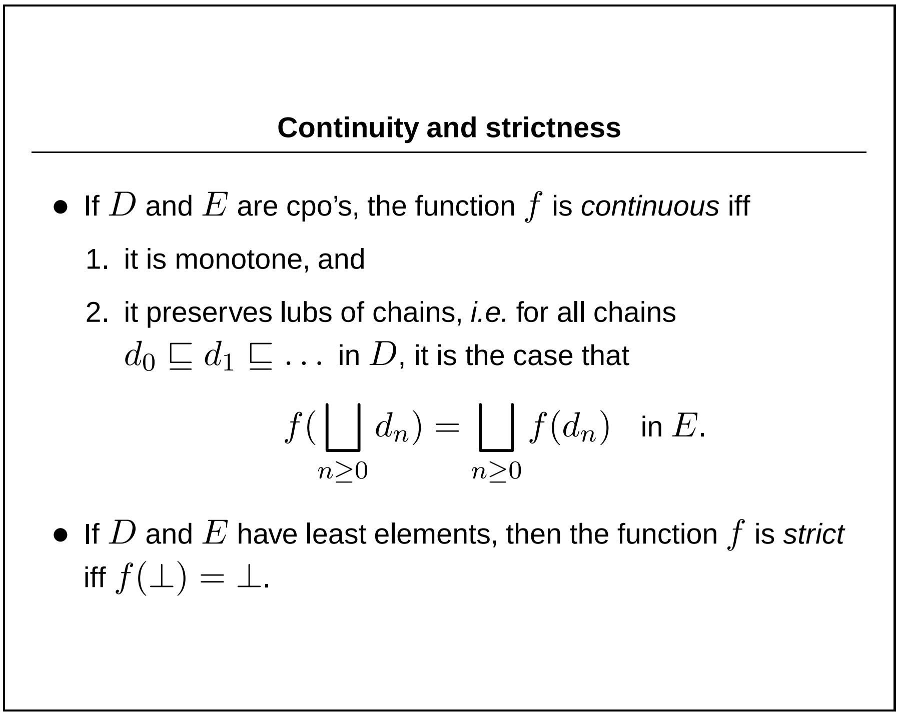
$$\begin{aligned} d_{m,n} &\sqsubseteq d_{\max\{m,n\},\max\{m,n\}} && \text{by property (†)} \\ &\sqsubseteq \bigsqcup_{k\geq 0} d_{k,k} && \text{by (lub1)} \end{aligned}$$

and hence applying (lub2) twice we have

$$\bigsqcup_{m\geq 0} \bigsqcup_{n\geq 0} d_{m,n} \sqsubseteq \bigsqcup_{k\geq 0} d_{k,k}. \tag{5}$$

Combining (4) and (5) with the anti-symmetry property of $\sqsubseteq$ yields the desired equality. We obtain the additional equality by the same argument but interchanging the roles of m and n. □

2.3.2 Continuous functions



Continuity and strictness

- If D and E are cpo's, the function f is *continuous* iff
  1. it is monotone, and
  2. it preserves lubs of chains, *i.e.* for all chains $d_0 \sqsubseteq d_1 \sqsubseteq \dots$ in D, it is the case that
  $$f(\bigsqcup_{n\geq 0} d_n) = \bigsqcup_{n\geq 0} f(d_n) \quad \text{in } E.$$
- If D and E have least elements, then the function f is *strict* iff $f(\bot) = \bot$.

Slide 25

Remark 2.3.7. Note that if $f : D \to E$ is monotone and $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \dots$ is a chain in D, then applying f we get a chain $f(d_0) \sqsubseteq f(d_1) \sqsubseteq f(d_2) \sqsubseteq \dots$ in E. Moreover, if d is an upper bound of the first chain, then $f(d)$ is an upper bound of the second and hence is greater than its lub. Hence if $f : D \to E$ is a monotone function between cpo's, we always have

$$\bigsqcup_{n\geq 0} f(d_n) \sqsubseteq f(\bigsqcup_{n\geq 0} d_n)$$

Therefore (using the antisymmetry property of $\sqsubseteq$), *to check that a monotone function f between cpo's is continuous, it suffices to check for each chain $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$ in D that*

$$f(\bigsqcup_{n\geq 0} d_n) \sqsubseteq \bigsqcup_{n\geq 0} f(d_n)$$

holds in E.

Example 2.3.8. Given cpo's D and E, for each $e \in E$ it is easy to see that the *constant function $D \to E$ with value e, $\lambda d \in D \,.\, e$,* is continuous.

Example 2.3.9. When D is the domain of partial functions $(State \rightharpoonup State)$ (cf. Slide 22), the function $f_{b,c} : D \to D$ defined on Slide 11 in connection with the denotational semantics of **while**-loops is a continuous function. We leave the verification of this as an exercise.

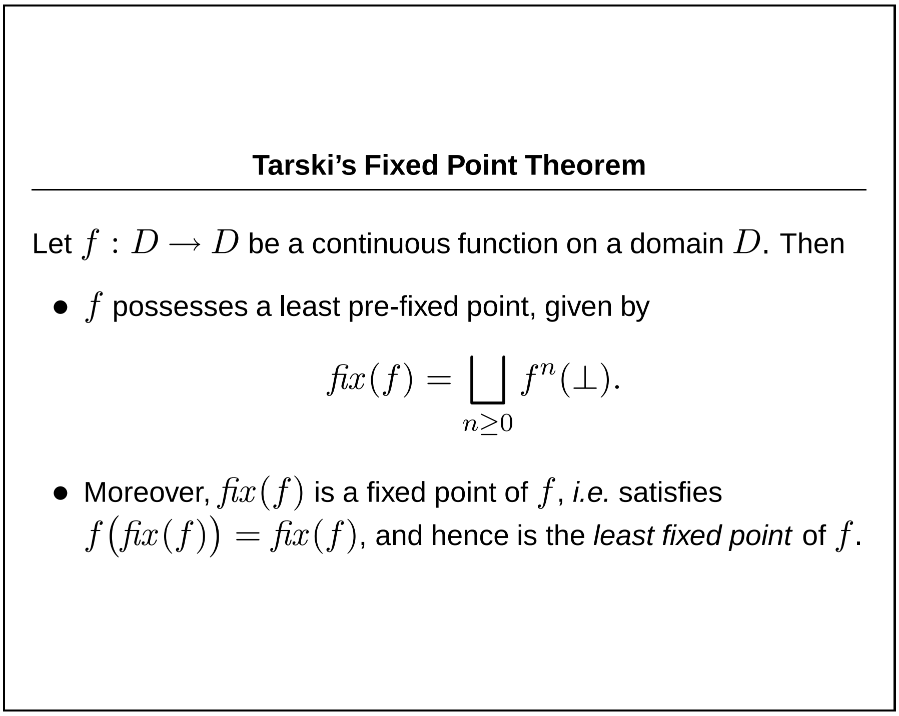
Example 2.3.10. Let Ω be the domain of vertical natural numbers, as defined in Figure 1. Then the function $f : \Omega \to \Omega$ defined by

$$\begin{cases} f(n) = 0 & (n \in \mathbb{N}) \\ f(\omega) = \omega. \end{cases}$$

is monotone and strict, but it is not continuous because

$$f(\bigsqcup_{n\geq 0} n) = f(\omega) = \omega \neq 0 = \bigsqcup_{n\geq 0} 0 = \bigsqcup_{n\geq 0} f(n).$$

2.4 Tarski's fixed point theorem



Tarski's Fixed Point Theorem

Let $f : D \to D$ be a continuous function on a domain D. Then

- f possesses a least pre-fixed point, given by
$$\mathit{fix}(f) = \bigsqcup_{n \geq 0} f^n(\bot).$$
- Moreover, $\mathit{fix}(f)$ is a fixed point of f, *i.e.* satisfies $f\big(\mathit{fix}(f)\big) = \mathit{fix}(f)$, and hence is the *least fixed point* of f.

Slide 26

Slide 26 gives the key result about continuous functions on domains which permits us to give denotational semantics of programs involving recursive features. The notation $f^n(\bot)$ used on the slide is defined as follows:

$$\begin{cases} f^0(\bot) & \overset{\text{def}}{=} \bot \\ f^{n+1}(\bot) & \overset{\text{def}}{=} f(f^n(\bot)). \end{cases} \tag{6}$$

Note that since $\forall d \in D.\ \bot \sqsubseteq d$, one has $f^0(\bot) = \bot \sqsubseteq f^1(\bot)$; and by monotonicity of f

$$f^n(\bot) \sqsubseteq f^{n+1}(\bot) \ \Rightarrow\ f^{n+1}(\bot) = f(f^n(\bot)) \sqsubseteq f(f^{n+1}(\bot)) = f^{n+2}(\bot).$$

Therefore, by induction on $n \in \mathbb{N}$, it is the case that $\forall n \in \mathbb{N}.\ f^n(\bot) \sqsubseteq f^{n+1}(\bot)$. In other words the elements $f^n(\bot)$ do form a chain in D. So since D is a cpo, the least upper bound used to define $\mathit{fix}(f)$ on Slide 26 does make sense.

Proof of Tarski's Fixed Point Theorem. First note that

$$\begin{aligned}
f(\mathit{fix}(f)) &= f(\bigsqcup_{n \geq 0} f^n(\bot)) \\
&= \bigsqcup_{n \geq 0} f(f^n(\bot)) && \text{by continuity of } f \\
&= \bigsqcup_{n \geq 0} f^{n+1}(\bot) && \text{by (6)} \\
&= \bigsqcup_{n \geq 0} f^n(\bot) && \text{by Remark 2.3.2(v)} \\
&= \mathit{fix}(f).
\end{aligned}$$

So $\mathit{fix}(f)$ is indeed a fixed point for f and hence in particular satisfies condition (lfp1) on Slide 18. To verify the second condition (lfp2) needed for a least pre-fixed point, suppose that $d \in D$ satisfies $f(d) \sqsubseteq d$. Then since $\bot$ is least in D

$$f^0(\bot) = \bot \sqsubseteq d$$

and

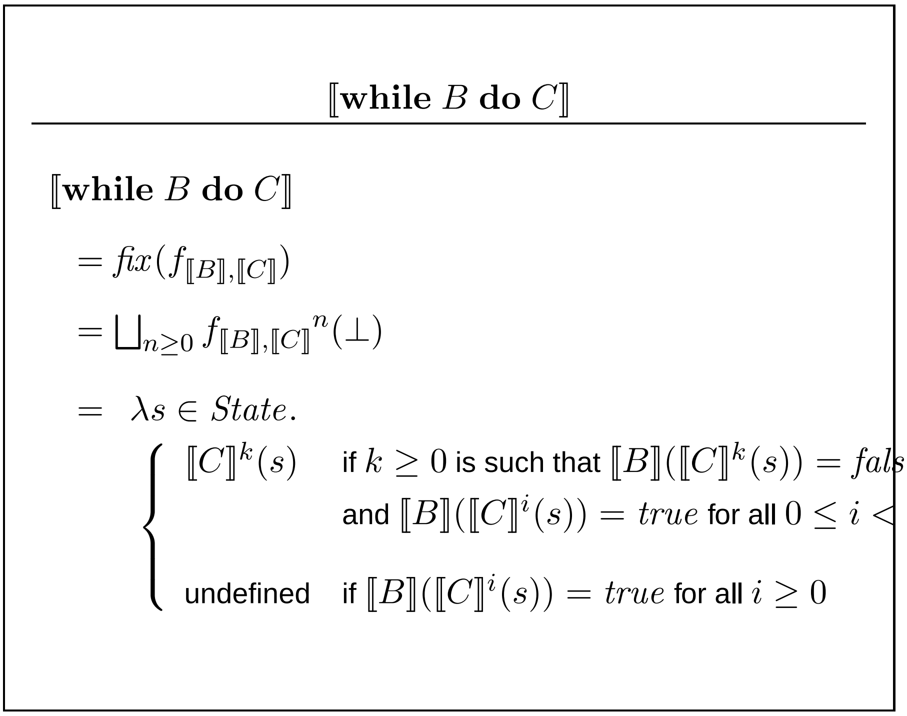
$$\begin{aligned}
f^n(\bot) \sqsubseteq d \ \Rightarrow \ f^{n+1}(\bot) = f(f^n(\bot)) &\sqsubseteq f(d) && \text{monotonicity of } f \\
&\sqsubseteq d && \text{by assumption on } d.
\end{aligned}$$

Hence by induction on $n \in \mathbb{N}$ we have $\forall n \in \mathbb{N}.\ f^n(\bot) \sqsubseteq d$. Therefore d is an upper bound for the chain and hence lies above the least such, i.e.

$$\mathit{fix}(f) = \bigsqcup_{n \geq 0} f^n(\bot) \sqsubseteq d$$

as required for (lfp2). □

Example 2.4.1. The function $f_{[\![B]\!],[\![C]\!]}$ defined on Slide 11 is a continuous function (Exercise 2.5.3) on the domain $(\mathit{State} \rightharpoonup \mathit{State})$ (Slide 22). So we can apply the Fixed Point Theorem and define $[\![\mathbf{while}\ B\ \mathbf{do}\ C]\!]$ to be $\mathit{fix}(f_{[\![B]\!],[\![C]\!]})$. In particular, the method used to construct the partial function w_∞ at the end of Section 1.2 is an instance of the method used in the proof of the Fixed Point Theorem to construct least pre-fixed points.



$[\![\textbf{while}\ B\ \textbf{do}\ C]\!]$

$$
\begin{aligned}
&[\![\textbf{while}\ B\ \textbf{do}\ C]\!]\\
&\quad = \mathit{fix}(f_{[\![B]\!],[\![C]\!]})\\
&\quad = \textstyle\bigsqcup_{n\geq 0} {f_{[\![B]\!],[\![C]\!]}}^{n}(\bot)\\
&\quad = \ \lambda s \in \mathit{State}.\\
&\qquad \begin{cases}
[\![C]\!]^{k}(s) & \text{if } k \geq 0 \text{ is such that } [\![B]\!]([\![C]\!]^{k}(s)) = \mathit{false}\\
 & \text{and } [\![B]\!]([\![C]\!]^{i}(s)) = \mathit{true} \text{ for all } 0 \leq i < k\\
\text{undefined} & \text{if } [\![B]\!]([\![C]\!]^{i}(s)) = \mathit{true} \text{ for all } i \geq 0
\end{cases}
\end{aligned}
$$

Slide 27

2.5 Exercises

Exercise 2.5.1. Verify the claims implicit on Slide 22: that the relation $\sqsubseteq$ defined there is a partial order; that f is indeed the lub of the chain $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \ldots$; and that the totally undefined partial function is the least element.

Exercise 2.5.2. Prove the claims in Slides 23 and 24.

Exercise 2.5.3. Verify the claim made in Example 2.3.9 that $f_{b,c}$ is continuous. When is it strict?

3 Constructions on Domains

In this section we give various ways of building domains and continuous functions, concentrating on the ones that will be needed for a denotational semantics of the programming language PCF studied in the second half of the course. Note that to specify a cpo one must *define* a set equipped with a binary relation and then *prove*

(i) the relation is a partial order;

(ii) lubs exist for all chains in the partially ordered set.

Furthermore, for the cpo to be a domain, one just has to prove

(iii) there is a least element.

Note that since lubs of chains and least elements are unique if they exist, a cpo or domain is completely determined by its underlying set and partial order. In what follows we will give various recipes for constructing cpos and domains and leave as an exercise the task of checking that properties (i), (ii), and (iii) do hold.

3.1 Flat domains

In order to model the PCF ground types nat and $bool$, we will use the notion of *flat domain* given on Slide 28.

Discrete cpo's and flat domains

For any set X, the relation of equality

$$x \sqsubseteq x' \overset{\text{def}}{\Leftrightarrow} x = x' \qquad (x, x' \in X)$$

makes $(X, \sqsubseteq)$ into a cpo, called the *discrete* cpo with underlying set X.

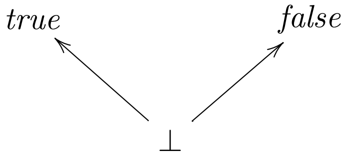
Let $X_\perp \overset{\text{def}}{=} X \cup \{\perp\}$, where $\perp$ is some element not in X. Then

$$d \sqsubseteq d' \overset{\text{def}}{\Leftrightarrow} (d = d') \vee (d = \perp) \qquad (d, d' \in X_\perp)$$

makes $(X_\perp, \sqsubseteq)$ into a domain (with least element $\perp$), called the *flat* domain determined by X.

Slide 28

The flat domain of natural numbers, $\mathbb{N}_\perp$, is pictured in Figure 1; the flat domain of booleans, $\mathbb{B}_\perp$ looks like:



The following instances of continuous functions involving flat domains will also be needed for the denotational semantics of PCF. We leave the proofs as exercises.

Proposition 3.1.1. *Let $f : X \rightharpoonup Y$ be a partial function between two sets. Then*

$$\begin{aligned} &f_\perp : X_\perp \to Y_\perp \\ &f_\perp(d) \overset{\text{def}}{=} \begin{cases} f(d) & \text{if } d \in X \text{ and } f \text{ is defined at } d \\ \perp & \text{if } d \in X \text{ and } f \text{ is not defined at } d \\ \perp & \text{if } d = \perp \end{cases} \end{aligned}$$

defines a continuous function between the corresponding flat domains.

3.2 Products of domains

Binary product of cpo's and domains

The *product* of two cpo's $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$ has underlying set

$$D_1 \times D_2 = \{(d_1, d_2) \mid d_1 \in D_1 \ \& \ d_2 \in D_2\}$$

and partial order $\sqsubseteq$ defined by

$$(d_1, d_2) \sqsubseteq (d_1', d_2') \overset{\text{def}}{\Leftrightarrow} d_1 \sqsubseteq_1 d_1' \ \& \ d_2 \sqsubseteq_2 d_2'$$

Lubs of chains are calculated componentwise:

$$\bigsqcup_{n \geq 0} (d_{1,n}, d_{2,n}) = (\bigsqcup_{i \geq 0} d_{1,i}, \bigsqcup_{j \geq 0} d_{2,j}).$$

If $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$ are domains so is $(D_1 \times D_2, \sqsubseteq)$ and $\perp_{D_1 \times D_2} = (\perp_{D_1}, \perp_{D_2})$.

Slide 29

Proposition 3.2.1 (Projections and pairing). *Let D_1 and D_2 be cpo's. The* projections

$$\begin{aligned} &\pi_1 : D_1 \times D_2 \to D_1 && \pi_2 : D_1 \times D_2 \to D_2 \\ &\pi_1(d_1, d_2) \overset{\text{def}}{=} d_1 && \pi_2(d_1, d_2) \overset{\text{def}}{=} d_2 \end{aligned}$$

are continuous functions. If $f_1 : D \to D_1$ and $f_2 : D \to D_2$ are continuous functions from a cpo D, then

$$\begin{aligned} &\langle f_1, f_2 \rangle : D \to D_1 \times D_2 \\ &\langle f_1, f_2 \rangle(d) \overset{\text{def}}{=} (f_1(d), f_2(d)) \end{aligned}$$

is continuous.

Proof. Continuity of these functions follows immediately from the characterisation of lubs of chains in $D_1 \times D_2$ given on Slide 29. □

Proposition 3.2.2. *For each domain D the function*

$$\begin{aligned} &\mathit{if} : \mathbb{B}_\perp \times (D \times D) \to D \\ &\mathit{if}(x, (d, d')) \overset{\text{def}}{=} \begin{cases} d & \mathit{if}\, x = \mathit{true} \\ d' & \mathit{if}\, x = \mathit{false} \\ \perp_D & \mathit{if}\, x = \perp \end{cases} \end{aligned}$$

is continuous.

We will need the following generalised version of the product construction.

Definition 3.2.3 (Dependent products). Given a set I, suppose that for each $i \in I$ we are given a cpo $(D_i, \sqsubseteq_i)$. The *product* of this whole family of cpo's has

- underlying set equal to the I-fold cartesian product, $\prod_{i \in I} D_i$, of the sets D_i—so it consists of all functions p defined on I and such that the value of p at each $i \in I$ is an element $p(i) \in D_i$ of the cpo D_i;
- partial order $\sqsubseteq$ defined by

$$p \sqsubseteq p' \overset{\text{def}}{\Leftrightarrow} \forall i \in I.\, p(i) \sqsubseteq_i p'(i).$$

As for the binary product (which is the particular case when I is a two-element set), lubs in $(\prod_{i \in I} D_i\,, \sqsubseteq)$ can be calculated componentwise: if $p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq \ldots$ is a chain in the product cpo, its lub is the function mapping each $i \in I$ to the lub in D_i of the chain $p_0(i) \sqsubseteq p_1(i) \sqsubseteq p_2(i) \sqsubseteq \ldots$. Thus

$$(\bigsqcup_{n \geq 0} p_n)(i) = \bigsqcup_{n \geq 0} p_n(i) \qquad (i \in I).$$

In particular, for each $i \in I$ the ith *projection function*

$$\begin{aligned} \pi_i &: \prod_{j \in I} D_j \to D_i \\ \pi_i(p) &\stackrel{\text{def}}{=} p(i) \end{aligned}$$

is continuous. If all the D_i are domains, then so is their product—the least element being the function mapping each $i \in I$ to the least element of D_i.

Continuous functions of two arguments

Proposition. *Let D, E, F be cpo's. A function $f : (D \times E) \to F$ is monotone if and only if it is monotone in each argument separately:*

$$\begin{aligned} &\forall d, d' \in D, e \in E.\, d \sqsubseteq d' \;\Rightarrow\; f(d,e) \sqsubseteq f(d',e) \\ &\forall d \in D, e, e' \in E.\, e \sqsubseteq e' \;\Rightarrow\; f(d,e) \sqsubseteq f(d,e'). \end{aligned}$$

Moreover, it is continuous if and only if it preserves lubs of chains in each argument separately:

$$\begin{aligned} f(\bigsqcup_{m \geq 0} d_m\,,\, e) &= \bigsqcup_{m \geq 0} f(d_m, e) \\ f(d\,,\, \bigsqcup_{n \geq 0} e_n) &= \bigsqcup_{n \geq 0} f(d, e_n). \end{aligned}$$

Slide 30

Proof of the Proposition on Slide 30. The 'only if' direction is straightforward; its proof rests on the simple observations that if $d \sqsubseteq d'$ then $(d,e) \sqsubseteq (d',e)$, and $(\bigsqcup_{m \geq 0} d_m\,,\, e) = \bigsqcup_{m \geq 0}(d_m\,,\, e)$, as well as the companion facts for the right argument. For the 'if' direction, suppose first that f is monotone in each argument separately. Then given $(d,e) \sqsubseteq (d',e')$ in $D \times E$, by definition of the partial order on the binary product we have $d \sqsubseteq d'$ in D and $e \sqsubseteq e'$ in E. Hence

$$\begin{aligned} f(d,e) &\sqsubseteq f(d',e) && \text{by monotonicity in first argument} \\ &\sqsubseteq f(d',e') && \text{by monotonicity in second argument} \end{aligned}$$

and therefore by transitivity, $f(d,e) \sqsubseteq f(d',e')$, as required for monotonicity of f.

Now suppose f is continuous in each argument separately. Then given a chain $(d_0, e_0) \sqsubseteq (d_1, e_1) \sqsubseteq (d_2, e_2) \sqsubseteq \dots$ in the binary product, we have

$$\begin{aligned}
f(\bigsqcup_{n\geq 0}(d_n, e_n)) &= f(\bigsqcup_{i\geq 0} d_i\,, \bigsqcup_{j\geq 0} e_j) && \text{(cf. Slide 29)}\\
&= \bigsqcup_{i\geq 0} f(d_i, \bigsqcup_{j\geq 0} e_j) && \text{by continuity in first argument}\\
&= \bigsqcup_{i\geq 0}\left(\bigsqcup_{j\geq 0} f(d_i, e_j)\right) && \text{by continuity in second argument}\\
&= \bigsqcup_{n\geq 0} f(d_n, e_n) && \text{by lemma on Slide 24}
\end{aligned}$$

as required for continuity of f. □

3.3 Function domains

The set of continuous functions between two cpo's/domains can be made into a cpo/domain as shown on Slide 31. The terminology 'exponential cpo/domain' is sometimes used instead of 'function cpo/domain'.

Function cpo's and domains

Given cpo's $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$, the *function cpo* $(D \to E, \sqsubseteq)$ has underlying set

$$D \to E \stackrel{\text{def}}{=} \{f \mid f : D \to E \text{ is a } \textit{continuous} \text{ function}\}$$

and partial order: $f \sqsubseteq f' \stackrel{\text{def}}{\Leftrightarrow} \forall d \in D \,.\, f(d) \sqsubseteq_E f'(d)$.

Lubs of chains are calculated 'argumentwise' (using lubs in E):

$$\bigsqcup_{n\geq 0} f_n \;=\; \lambda d \in D.\ \bigsqcup_{n\geq 0} f_n(d)\ .$$

If E is a domain, then so is $D \to E$ and $\bot_{D\to E}(d) = \bot_E$, all $d \in D$.

Slide 31

Proof of Slide 31. We should show that the lub of a chain of functions, $\bigsqcup_{n\geq 0} f_n$, is continuous. The proof uses the 'interchange law' of Slide 24]. Given a chain in D,

$$\begin{aligned}
(\bigsqcup_{n\geq 0} f_n)((\bigsqcup_{m\geq 0} d_m)) &= \bigsqcup_{n\geq 0}(f_n(\bigsqcup_{m\geq 0} d_m)) && \text{definition of } \bigsqcup_{n\geq 0} f_n \\
&= \bigsqcup_{n\geq 0}(\bigsqcup_{m\geq 0} f_n(d_m)) && \text{continuity of each } f_n \\
&= \bigsqcup_{m\geq 0}(\bigsqcup_{n\geq 0} f_n(d_m)) && \text{interchange law} \\
&= \bigsqcup_{m\geq 0}((\bigsqcup_{n\geq 0} f_n)(d_m)) && \text{definition of } \bigsqcup_{n\geq 0} f_n.
\end{aligned}$$

□

Proposition 3.3.1 (Evaluation and 'Currying'). *Given cpo's D and E, the function*

$$\begin{aligned}
&ev : (D \to E) \times D \to E \\
&ev(f,d) \stackrel{\text{def}}{=} f(d)
\end{aligned}$$

is continuous. Given any continuous function $f : D' \times D \to E$ (with D' a cpo), for each $d' \in D'$ the function $d \in D \mapsto f(d',d)$ is continuous and hence determines an element of the function cpo $D \to E$ that we denote by $cur(f)(d')$. Then

$$\begin{aligned}
&cur(f) : D' \to (D \to E) \\
&cur(f)(d') \stackrel{\text{def}}{=} \lambda d \in D \, . \, f(d',d)
\end{aligned}$$

is a continuous function.[1]

[1] This 'Curried' version of f is named in honour of the logician H. B. Curry, a pioneer of combinatory logic and lambda calculus.

Proof. For continuity of ev note that

$$
\begin{aligned}
ev(\bigsqcup_{n\geq 0}(f_n, d_n)) &= ev(\bigsqcup_{i\geq 0} f_i\,, \bigsqcup_{j\geq 0} d_j) && \text{lubs in products are componenwise}\\
&= (\bigsqcup_{i\geq 0} f_i)\,(\bigsqcup_{j\geq 0} d_j) && \text{by definition of } ev\\
&= \bigsqcup_{i\geq 0} f_i(\bigsqcup_{j\geq 0} d_j) && \text{lubs in function cpo's are argumentwise}\\
&= \bigsqcup_{i\geq 0}\bigsqcup_{j\geq 0} f_i(d_j) && \text{by continuity of each } f_i\\
&= \bigsqcup_{n\geq 0} f_n(d_n) && \text{by the Lemma on Slide 24}\\
&= \bigsqcup_{n\geq 0} ev(f_n, d_n) && \text{by definition of } ev.
\end{aligned}
$$

The continuity of each $cur(f)(d')$ and then of $cur(f)$ follows immediately from the fact that lubs of chains in $D_1 \times D_2$ can be calculated componentwise. □

Continuity of composition

For cpo's D, E, F, the composition function

$$\circ : \big((E \to F) \times (D \to E)\big) \longrightarrow (D \to F)$$

defined by setting, for all $f \in (D \to E)$ and $g \in (E \to F)$,

$$g \circ f \;=\; \lambda d \in D.\, g\big(f(d)\big)$$

is continuous.

Slide 32

Continuity of the fixpoint operator

Let D be a domain.

By Tarski's Fixed Point Theorem we know that each continuous function $f \in (D \to D)$ possesses a least fixed point, $\mathit{fix}(f) \in D$.

Proposition. *The function*

$$\mathit{fix} : (D \to D) \to D$$

is continuous.

Slide 33

Proof of the Proposition on Slide 33. We must first prove that $\mathit{fix} : (D \to D) \to D$ is a monotone function. Suppose $f_1 \sqsubseteq f_2$ in the function domain $D \to D$. We have to prove $\mathit{fix}(f_1) \sqsubseteq \mathit{fix}(f_2)$. But:

$$\begin{aligned} f_1(\mathit{fix}(f_2)) &\sqsubseteq f_2(\mathit{fix}(f_2)) && \text{since } f_1 \sqsubseteq f_2 \\ &\sqsubseteq \mathit{fix}(f_2) && \text{by (lfp1) for } \mathit{fix}(f_2). \end{aligned}$$

So $\mathit{fix}(f_2)$ is a pre-fixed point for f_1 and hence by (lfp2) (for $\mathit{fix}(f_1)$) we have $\mathit{fix}(f_1) \sqsubseteq \mathit{fix}(f_2)$, as required.

Turning now to the preservation of lubs of chains, suppose $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \ldots$ in $D \to D$. Recalling Remark 2.3.7, we just have to prove that

$$\mathit{fix}(\bigsqcup_{n \geq 0} f_n) \sqsubseteq \bigsqcup_{n \geq 0} \mathit{fix}(f_n)$$

and by the property (lfp2) of least pre-fixed points (see Slide 18), for this it suffices to show that $\bigsqcup_{n \geq 0} \mathit{fix}(f_n)$ is a pre-fixed point for the function $\bigsqcup_{n \geq 0} f_n$. This is the case because:

$$\begin{aligned}
(\bigsqcup_{m\geq 0} f_m)(\bigsqcup_{n\geq 0} \mathit{fix}(f_n)) &= \bigsqcup_{m\geq 0} f_m(\bigsqcup_{n\geq 0} \mathit{fix}(f_n)) && \text{function lubs are argumentwise}\\
&= \bigsqcup_{m\geq 0}\bigsqcup_{n\geq 0} f_m(\mathit{fix}(f_n)) && \text{by continuity of each } f_m\\
&= \bigsqcup_{k\geq 0} f_k(\mathit{fix}(f_k)) && \text{by the Lemma on Slide 24}\\
&\sqsubseteq \bigsqcup_{k\geq 0} \mathit{fix}(f_k) && \text{by (lfp1) for each } f_k.
\end{aligned}$$

□

3.4 Exercises

Exercise 3.4.1. Verify that the constructions given on Slide 29, in Definition 3.2.3, and on Slides 31 and 28 do give cpo's and domains (i.e. that properties (i), (ii) and (ii) mentioned at the start of this section do hold in each case). Give the proofs of Propositions 3.1.1 and 3.2.2.

Exercise 3.4.2. Let X and Y be sets and $X_\perp$ and $Y_\perp$ the corresponding flat domains, as on Slide 28. Show that a function $f : X_\perp \to Y_\perp$ is continuous if and only if one of (a) or (b) holds:

(a) f is strict, i.e. $f(\perp) = \perp$.

(b) f is constant, i.e. $\forall x \in X \,.\, f(x) = f(\perp)$.

Exercise 3.4.3. Let $\{\top\}$ be a one-element set and $\{\top\}_\perp$ the corresponding flat domain. Let Ω be the domain of 'vertical natural numbers', pictured in Figure 1. Show that the function domain $(\Omega \to \{\top\}_\perp)$ is in bijection with Ω.

Exercise 3.4.4. Prove the Proposition on Slide 32.

4 Scott Induction

4.1 Chain-closed and admissible subsets

In Section 2 we saw that the least fixed point of a continuous function $f : D \to D$ on a domain D can be expressed as the lub of the chain obtained by repeatedly applying f starting with the least element $\bot$ of D: $\mathit{fix}(f) = \bigsqcup_{n \geq 0} f^n(\bot)$ (cf. Slide 26). This construction allows one to prove properties of $\mathit{fix}(f)$ by using Mathematical Induction for n to show that each $f^n(\bot)$ has the property, *provided* the property in question satisfies the condition shown on Slide 34. It is convenient to package up this use of Mathematical Induction in a way that hides the explicit construction of $\mathit{fix}(f)$ as the lub of a chain. This is done on Slide 35. To see the validity of the statement on that slide, note that $f^0(\bot) = \bot \in S$ by the **Base case**; and $f^n(\bot) \in S$ implies $f^{n+1}(\bot) = f(f^n(\bot)) \in S$ by the **Induction step**. Hence by induction on n, we have $\forall n \geq 0 \,.\, f^n(\bot) \in S$. Therefore by the chain-closedness of S, $\mathit{fix}(f) = \bigsqcup_{n \geq 0} f^n(\bot) \in S$, as required.

Chain-closed and admissible subsets

Let D be a cpo. A subset $S \subseteq D$ is called *chain-closed* iff for all chains $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$ in D

$$(\forall n \geq 0 \,.\, d_n \in S) \;\Rightarrow\; \Big(\bigsqcup_{n \geq 0} d_n\Big) \in S$$

If D is a domain, $S \subseteq D$ is called *admissible* iff it is a chain-closed subset of D and $\bot \in S$.

A property $\Phi(d)$ of elements $d \in D$ is called *chain-closed* (resp. *admissible*) iff $\{d \in D \mid \Phi(d)\}$ is a *chain-closed* (resp. *admissible*) subset of D.

Slide 34

Note. The terms *inclusive*, or *inductive*, are often used as synonyms of ‘chain-closed’.

Example 4.1.1. Consider the domain Ω of ‘vertical natural numbers’ pictured in Figure 1. Then

- any *finite* subset of Ω is chain-closed;

- $\{0, 2, 4, 6, \ldots\}$ is not a chain-closed subset of Ω;
- $\{0, 2, 4, 6, \ldots\} \cup \{\omega\}$ is a chain-closed (indeed, is an admissible) subset of Ω.

Scott's Fixed Point Induction Principle

Let $f : D \to D$ be a continuous function on a domain D.

For any <u>admissible</u> subset $S \subseteq D$, to prove that the least fixed point of f is in S, *i.e.* that

$$\mathit{fix}(f) \in S \ ,$$

it suffices to prove

$$\forall d \in D \ (d \in S \ \Rightarrow \ f(d) \in S) \ .$$

Slide 35

The difficulty with applying Scott's Fixed Point Induction Principle in any particular case usually lies in identifying an appropriate admissible subset S; i.e. in finding a suitably strong 'induction hypothesis'.

4.2 Examples

Example 4.2.1. Suppose that D is a domain and that $f : (D \times (D \times D)) \to D$ is a continuous function. Let $g : (D \times D) \to (D \times D)$ be the continuous function defined by

$$g(d_1, d_2) \stackrel{\text{def}}{=} (f(d_1, (d_1, d_2)), f(d_1, (d_2, d_2))) \qquad (d_1, d_2 \in D).$$

Then $u_1 = u_2$, where $(u_1, u_2) \stackrel{\text{def}}{=} \mathit{fix}(g)$. (Note that g *is* continuous because we can express it in terms of composition, projections and pairing and hence apply Proposition 3.2.1 and Slide 32: $g = \langle f \circ \langle \pi_1, \langle \pi_1, \pi_2 \rangle \rangle, f \circ \langle \pi_1, \langle \pi_2, \pi_2 \rangle \rangle \rangle$.)

Proof. We have to show that $\mathit{fix}(g) \in \Delta$ where

$$\Delta \stackrel{\text{def}}{=} \{(d_1, d_2) \in D \times D \mid d_1 = d_2\}.$$

It is not hard to see that Δ is an admissible subset of the product domain $D \times D$. So by Scott's Fixed Point Induction Principle, we just have to check that

$$\forall (d_1, d_2) \in D \times D\ ((d_1, d_2) \in \Delta \ \Rightarrow\ g(d_1, d_2) \in \Delta)$$

or equivalently, that

$$\forall (d_1, d_2) \in D \times D\ (d_1 = d_2 \ \Rightarrow\ f(d_1, d_1, d_2) = f(d_1, d_2, d_2)),$$

which is clearly true. □

The next example shows that Scott's Induction Principle can be useful for proving (the denotational version of) *partial correctness* assertions about programs, i.e. assertions of the form 'if the program terminates, then such-and-such a property holds of the results'. By contrast, a *total* correctness assertion would be 'the program does terminate and such-and-such a property holds of the results'. Because Scott Induction can only be applied for properties Φ for which $\Phi(\bot)$ holds, it is not so useful for proving total correctness.

Example 4.2.2. Let $f : D \to D$ be the continuous function defined on Slide 12 whose least fixed point is the denotation of the command

$$\mathbf{while}\ X > 0\ \mathbf{do}\ (Y := X * Y\ ; X := X - 1).$$

We will use Scott Induction to prove
(7)

$$\begin{array}{l} \forall x, y \geq 0\,. \\ \quad \mathit{fix}(f)[X \mapsto x, Y \mapsto y] \downarrow \Rightarrow\ \mathit{fix}(f)[X \mapsto x, Y \mapsto y] = [X \mapsto 0, Y \mapsto (!x) * y] \end{array}$$

where for $w \in D = \big((\mathbb{Z} \times \mathbb{Z}) \rightharpoonup (\mathbb{Z} \times \mathbb{Z})\big)$ we write $w(x, y) \downarrow$ to mean 'the partial function w is defined at the state $[X \mapsto x, Y \mapsto y]$'.

Proof. Let

$$S \stackrel{\text{def}}{=} \left\{ w \in D \;\middle|\; \begin{array}{l} \forall x, y \geq 0\,. \\ \quad w[X \mapsto x, Y \mapsto y] \downarrow \\ \qquad \Rightarrow\ w[X \mapsto x, Y \mapsto y] = [X \mapsto 0, Y \mapsto (!x) * y] \end{array} \right\}.$$

It is not hard to see that S is admissible. Therefore, to prove (7), by Scott Induction it suffices to check that $w \in S$ implies $f(w) \in S$, for all $w \in D$. So suppose $w \in S$, that $x, y \geq 0$, and that $f(w)[X \mapsto x, Y \mapsto y] \downarrow$. We have to show that $f(w)[X \mapsto x, Y \mapsto y] = [X \mapsto 0, Y \mapsto (!x) * y]$. We consider the two cases $x = 0$ and $x > 0$ separately.

If $x = 0$, then by definition of f (see Slide 12)

$$\begin{array}{rclcl} f(w)[X \mapsto x, Y \mapsto y] & = & [X \mapsto x, Y \mapsto y] & = & [X \mapsto 0, Y \mapsto y] \\ & = & [X \mapsto 0, Y \mapsto 1 * y] & = & [X \mapsto 0, Y \mapsto (!0) * y] \\ & = & [X \mapsto 0, Y \mapsto (!x) * y]\ . & & \end{array}$$

On the other hand, if $x > 0$, then by definition of f

$$w[X \mapsto x-1, Y \mapsto x*y] = f(w)[X \mapsto x, Y \mapsto y] \downarrow \quad \text{(by assumption)}$$

and then since $w \in S$ and $x-1, x*y \geq 0$, we must have

$$w[X \mapsto x-1, Y \mapsto x*y] = [X \mapsto 0, Y \mapsto !(x-1)*(x*y)]$$

and hence once again

$$\begin{array}{rcl} f(w)[X \mapsto x, Y \mapsto y] & = & w[X \mapsto x-1, Y \mapsto x*y] \\ & = & [X \mapsto 0, Y \mapsto !(x-1)*(x*y)] \\ & = & [X \mapsto 0, Y \mapsto (!x)*y] \ . \end{array}$$

□

4.3 Building chain-closed subsets

The power of Scott induction depends on having a good stock of chain-closed subsets. Fortunately we are able to ensure that a good many subsets are chain-closed by virtue of the way in which they are built up.

4.3.1 Basic relations

Let D be a cpo. The subsets

$$\{(x,y) \in D \times D \mid x \sqsubseteq y\} \text{ and } \{(x,y) \in D \times D \mid x = y\}$$

of $D \times D$ are chain-closed (Why?). The properties (or predicates) $x \sqsubseteq y$ and $x = y$ on $D \times D$ determine chain-closed sets.

Example (I): Least pre-fixed point property

Let D be a domain and let $f : D \to D$ be a continuous function.

$$\forall d \in D.\, f(d) \sqsubseteq d \implies \mathit{fix}(f) \sqsubseteq d$$

Proof by Scott induction.

Let $d \in D$ be a pre-fixed point of f. Then,

$$\begin{array}{rcl} x \in {\downarrow}(d) & \implies & x \sqsubseteq d \\ & \implies & f(x) \sqsubseteq f(d) \\ & \implies & f(x) \sqsubseteq d \\ & \implies & f(x) \in {\downarrow}(d) \end{array}$$

Hence,

$$\mathit{fix}(f) \in {\downarrow}(d)\ .$$

Slide 36

4.3.2 Inverse image and substitution

Let $f : D \to E$ be a continuous function between cpos D and E. Suppose S is a chain-closed subset of E. Then the inverse image

$$f^{-1}S = \{x \in D \mid f(x) \in S\}$$

is an chain-closed subset of D (Why?).

Suppose the subset S is defined by the property P on E *i.e.*

$$S = \{y \in E \mid P(y)\}.$$

Then

$$f^{-1}S = \{x \in D \mid P(f(x))\}.$$

So, if a property $P(y)$ on E determines a chain-closed subset of E and $f : D \to E$ is a continuous function, then the property $P(f(x))$ on D determines a chain-closed subset of D.

Example (II)

Let D be a domain and let $f, g : D \to D$ be continuous functions such that $f \circ g \sqsubseteq g \circ f$. Then,

$$f(\bot) \sqsubseteq g(\bot) \implies \mathit{fix}(f) \sqsubseteq \mathit{fix}(g) \ .$$

Proof by Scott induction.

Consider the admissible property $\Phi(x) \equiv \big(f(x) \sqsubseteq g(x)\big)$ of D.

Since

$$f(x) \sqsubseteq g(x) \Rightarrow g(f(x)) \sqsubseteq g(g(x)) \Rightarrow f(g(x)) \sqsubseteq g(g(x))$$

we have that

$$f(\mathit{fix}(g)) \sqsubseteq g(\mathit{fix}(g)) \ .$$

Slide 37

4.3.3 Logical operations

Let D be a cpo. Let $S \subseteq D$ and $T \subseteq D$ be chain-closed subsets of D. Then

$$S \cup T \ \text{ and } \ S \cap T$$

are chain-closed subsets (Why?). In terms of properties, if $P(x)$ and $Q(x)$ determine chain-closed subsets of D, then so do

$$P(x) \text{ or } Q(x), \qquad P(x) \ \& \ Q(x).$$

If $S_i, \ i \in I$, is a family of chain-closed subsets of D indexed by a set I, then $\bigcap_{i \in I} S_i$ is a chain-closed subset of D (Why?).

Consequently, if a property $P(x, y)$ determines a chain-closed subset of $D \times E$, then the property $\forall x \in D.\ P(x, y)$ determines a chain-closed subset of E. This is because

$$\begin{aligned} \{y \in E \mid \forall x \in D.\ P(x,y)\} &= \bigcap_{d \in D} \{y \in E \mid P(d,y)\} \\ &= \bigcap_{d \in D} {f_d}^{-1}\{(x,y) \in D \times E \mid P(x,y)\} \end{aligned}$$

where $f_d : D \times E \to D \times E$ is the continuous function such that $f_d(x, y) = (d, y)$ for each $d \in D$.

In fact, any property built-up as a universal quantification over several variables of conjunctions and disjunctions of basic properties of the form $f(x_1, \cdots, x_k) \sqsubseteq g(x_1, \cdots, x_l)$ or $f(x_1, \cdots, x_k) = g(x_1, \cdots, x_l)$, where f and g are continuous, will determine a chain-closed subset of the product cpo appropriate to the non-quantified variables.

Note, however, that infinite unions of chain-closed subsets need not be chain-closed; finite subsets are always chain complete but arbitrary unions of them need not be. Accordingly, we cannot in general build chain-closed subsets with existential quantifications.

4.4 Exercises

Exercise 4.4.1. Answer all the “Why?”s above in the building of chain-closed subsets.

Exercise 4.4.2. Give an example of a subset $S \subseteq D \times D'$ of a product cpo that is not chain-closed, but which satisfies:

(a) for all $d \in D$, $\{d' \mid (d, d') \in S\}$ is a chain-closed subset of D'; and

(b) for all $d' \in D'$, $\{d \mid (d, d') \in S\}$ is a chain-closed subset of D.

[Hint: consider $D = D' = \Omega$, the cpo in Figure 1.]

(Compare this with the property of continuous functions given on Slide 30.)

5 PCF

The language PCF (‘Programming Computable Functions’) is a simple functional programming language that has been used extensively as an example language in the development of the theory of both denotational and operational semantics (and the relationship between the two). Its syntax was introduced by Dana Scott *circa* 1969 as part of a ‘Logic of Computable Functions’[1] and was studied as a programming language in a highly influential paper by Plotkin (1977).

In this section we describe the syntax and operational semantics of the particular version of PCF we use in these notes. In Section 6 we will see how to give it a denotational semantics using domains and continuous function.

5.1 Terms and types

The *types*, *expressions*, and *terms* of the PCF language are defined on Slide 38.

PCF syntax

Types

$$\tau ::= nat \mid bool \mid \tau \to \tau$$

Expressions

$$\begin{array}{rcl} M & ::= & \mathbf{0} \mid \mathbf{succ}(M) \mid \mathbf{pred}(M) \\ & \mid & \mathbf{true} \mid \mathbf{false} \mid \mathbf{zero}(M) \\ & \mid & x \mid \mathbf{if}\ M\ \mathbf{then}\ M\ \mathbf{else}\ M \\ & \mid & \mathbf{fn}\, x : \tau \,.\, M \mid M\,M \mid \mathbf{fix}(M) \end{array}$$

where $x \in \mathbb{V}$, an infinite set of *variables*.

Technicality: We identify expressions up to α-conversion of bound variables (created by the $\mathbf{fn}$ expression-former): by definition a PCF *term* is an α-equivalence class of expressions.

Slide 38

The intended meaning of the various syntactic forms is as follows.

- nat is the type of the natural numbers, $0, 1, 2, 3, \ldots$. In PCF these are generated from 0 by repeated application of the successor operation, $\mathbf{succ}(-)$, whose intended meaning is to add 1 to its argument. The predecessor operation $\mathbf{pred}(-)$ subtracts 1 from strictly positive natural numbers (and is undefined at 0).

[1] This logic was the stimulus for the development of the ML language and LCF system for machine-assisted proofs by Milner, Gordon *et al*—see Paulson 1987; Scott’s original work was eventually published as Scott 1993.

- $bool$ is the type of booleans, $true$ and $false$. The operation $\mathbf{zero}(-)$ tests whether its argument is zero or strictly positive and returns $\mathbf{true}$ or $\mathbf{false}$ accordingly. The *conditional* expression $\mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3$ behaves like either M_2 or M_3 depending upon whether M_1 evaluates to $\mathbf{true}$ or $\mathbf{false}$ respectively.

- A PCF variable, x, stands for an unknown expression. PCF is a pure functional language—there is no state that changes during expression evaluation and in particular variables are 'identifiers' standing for a fixed expression, rather than 'program variables' whose contents may get mutated during evaluation.

- $\tau \to \tau'$ is the type of (partial) functions taking a single argument of type τ and (possibly) returning a result of type τ'. $\mathbf{fn}\, x : \tau \,.\, M$ is the notation we will use for function abstraction (i.e. lambda abstraction) in PCF; note that the type τ of the abstracted variable x is given explicitly. The application of function M_1 to argument M_2 is indicated by $M_1\, M_2$. As usual, the scope of a function abstraction extends as far to the right of the dot as possible and function application associates to the left (i.e. $M_1\, M_2\, M_3$ means $(M_1\, M_2)\, M_3$, not $M_1\, (M_2\, M_3)$).

- The expression $\mathbf{fix}(M)$ indicates an element x *recursively defined* by $x = M\, x$. The lambda calculus equivalent is $Y\, M$, where Y is a suitable fixpoint combinator.

5.2 Free variables, bound variables, and substitution

PCF contains one variable-binding form: free occurrences of x in M become bound in $\mathbf{fn}\, x : \tau \,.\, M$. The finite set of *free variables* of an expression M, $fv(M)$, is defined by induction on its structure, as follows:

$$fv(\mathbf{0}) = fv(\mathbf{true}) = fv(\mathbf{false}) \stackrel{\text{def}}{=} \emptyset$$

$$fv(\mathbf{succ}(M)) = fv(\mathbf{pred}(M)) = fv(\mathbf{zero}(M)) = fv(\mathbf{fix}(M)) \stackrel{\text{def}}{=} fv(M)$$

$$fv(\mathbf{if}\ M\ \mathbf{then}\ M'\ \mathbf{else}\ M'') \stackrel{\text{def}}{=} fv(M) \cup fv(M') \cup fv(M'')$$

$$fv(M\, M') \stackrel{\text{def}}{=} fv(M) \cup fv(M')$$

$$fv(x) \stackrel{\text{def}}{=} \{x\}$$

$$fv(\mathbf{fn}\, x : \tau \,.\, M) \stackrel{\text{def}}{=} \{x' \in fv(M) \mid x' \neq x\}.$$

One says that M is *closed* if $fv(M) = \emptyset$ and *open* otherwise.

As indicated on Slide 38, we will identify α-convertible PCF expressions, i.e. ones that differ only up to the names of their bound variables. Thus by definition, a PCF *term* is an equivalence class of PCF expressions for the equivalence relation of α-conversion. However, we will always refer to a term via some representative expression, usually choosing one whose bound variables are all distinct from each other and from any other variables in the context in which the term is being used. The operation of *substituting a term M for all free occurrences of a variable x in a term M'* will be written

$$M'[M/x].$$

The operation is carried out by textual substitution of an expression representing M for free occurrences of x in an expression representing M' whose binding variables are distinct from the free variables in M (thereby avoiding 'capture' of free variables in M by binders in M').

5.3 Typing

PCF is a typed language: types are assigned to terms via the relation shown on Slide 39 whose intended meaning is "if each $x \in dom(\Gamma)$ has type $\Gamma(x)$, then M has type τ".

PCF typing relation, $\Gamma \vdash M : \tau$

- Γ is a *type environment*, i.e. a finite partial function mapping variables to types (whose domain of definition is denoted $dom(\Gamma)$)
- M is a term
- τ is a *type*.

Relation is inductively defined by the axioms and rules in Figure 2.

Notation:

$M : \tau$ means M is closed and $\emptyset \vdash M : \tau$ holds.

$\mathrm{PCF}_\tau \overset{\text{def}}{=} \{M \mid M : \tau\}$.

Slide 39

$(:_{0})$ $\quad \Gamma \vdash \mathbf{0} : \mathit{nat}$

$(:_{\mathrm{succ}})$ $\quad \dfrac{\Gamma \vdash M : \mathit{nat}}{\Gamma \vdash \mathbf{succ}(M) : \mathit{nat}}$

$(:_{\mathrm{pred}})$ $\quad \dfrac{\Gamma \vdash M : \mathit{nat}}{\Gamma \vdash \mathbf{pred}(M) : \mathit{nat}}$

$(:_{\mathrm{zero}})$ $\quad \dfrac{\Gamma \vdash M : \mathit{nat}}{\Gamma \vdash \mathbf{zero}(M) : \mathit{bool}}$

$(:_{\mathrm{bool}})$ $\quad \Gamma \vdash b : \mathit{bool} \quad (b = \mathit{true}, \mathit{false})$

$(:_{\mathrm{if}})$ $\quad \dfrac{\Gamma \vdash M_1 : \mathit{bool} \quad \Gamma \vdash M_2 : \tau \quad \Gamma \vdash M_3 : \tau}{\Gamma \vdash \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 : \tau}$

$(:_{\mathrm{var}})$ $\quad \Gamma \vdash x : \tau$ if $x \in \mathit{dom}(\Gamma)\ \&\ \Gamma(x) = \tau$

$(:_{\mathrm{fn}})$ $\quad \dfrac{\Gamma[x \mapsto \tau] \vdash M : \tau'}{\Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M : \tau \to \tau'}$ if $x \notin \mathit{dom}(\Gamma)$

$(:_{\mathrm{app}})$ $\quad \dfrac{\Gamma \vdash M_1 : \tau \to \tau' \quad \Gamma \vdash M_2 : \tau}{\Gamma \vdash M_1\, M_2 : \tau'}$

$(:_{\mathrm{fix}})$ $\quad \dfrac{\Gamma \vdash M : \tau \to \tau}{\Gamma \vdash \mathbf{fix}(M) : \tau}$

In rule $(:_{\mathrm{fn}})$, $\Gamma[x \mapsto \tau]$ denotes the type environment mapping x to τ and otherwise acting like Γ.

Figure 2: Axioms and rules for PCF typing relation

Proposition 5.3.1.

(i) If $\Gamma \vdash M : \tau$ *holds, then* $fv(M) \subseteq dom(\Gamma)$*. If both* $\Gamma \vdash M : \tau$ *and* $\Gamma \vdash M : \tau'$ *hold, then* $\tau = \tau'$*. In particular a closed term has at most one type.*

(ii) If $\Gamma \vdash M : \tau$ *and* $\Gamma[x \mapsto \tau] \vdash M' : \tau'$ *both hold, then so does* $\Gamma \vdash M'[M/x] : \tau'$*.*

Proof. These properties of the inductively defined typing relation are easily proved by rule induction. The fact that a term has at most one type for a given assignment of types to its free variables relies upon the fact that types of bound variables are given explicitly in function abstractions. □

Example 5.3.2 (Partial recursive functions in PCF)**.** Although the PCF syntax is rather terse, the combination of increment, decrement, test for zero, conditionals, function abstraction and application, and fixpoint recursion makes it Turing expressive—in the sense that all partial recursive functions[1] can be coded. For example, recall that the partial function $h : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ defined by *primitive recursion* from $f : \mathbb{N} \rightharpoonup \mathbb{N}$ and $g : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ satisfies that for all $x, y \in \mathbb{N}$

$$\begin{cases} h(x,0) & = f(x) \\ h(x,y+1) & = g(x,y,h(x,y)). \end{cases}$$

Thus if f has been coded in PCF by a term $F : nat \to nat$ and g by a term $G : nat \to (nat \to (nat \to nat))$, then h can be coded by

$$\begin{aligned} H \overset{\text{def}}{=} & \ \mathbf{fix}(\mathbf{fn}\, h : nat \to (nat \to nat) \,.\, \mathbf{fn}\, x : nat \,.\, \mathbf{fn}\, y : nat \,. \\ & \quad \mathbf{if\ zero}(y)\ \mathbf{then}\ F\,x\ \mathbf{else}\ G\,x\,y\,(h\,x\,y)). \end{aligned}$$

Apart from primitive recursion, the other construction needed for defining partial recursive functions is *minimisation*. For example, the partial function $m : \mathbb{N} \rightharpoonup \mathbb{N}$ defined from $k : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ by minimisation satisfies that for all $x \in \mathbb{N}$

$$\begin{aligned} m(x) = & \text{ least } y \geq 0 \text{ such that } k(x,y) = 0 \text{ and} \\ & \quad \forall z.\, 0 \leq z < y \Rightarrow k(x,z) > 0. \end{aligned}$$

This can also be expressed using fixpoints, although not so easily as in the case of primitive recursion. For if k has been coded in PCF by a term $K : nat \to (nat \to nat)$, then in fact m can be coded as $\mathbf{fn}\, x : nat \,.\, M'\, x\, \mathbf{0}$ where

$$\begin{aligned} M' \overset{\text{def}}{=} & \ \mathbf{fix}(\mathbf{fn}\, m' : nat \to (nat \to nat) \,.\, \mathbf{fn}\, x : nat \,.\, \mathbf{fn}\, y : nat \,. \\ & \quad \mathbf{if\ zero}(K\,x\,y)\ \mathbf{then}\ y\ \mathbf{else}\ m'\,x\,\mathbf{succ}(y)). \end{aligned}$$

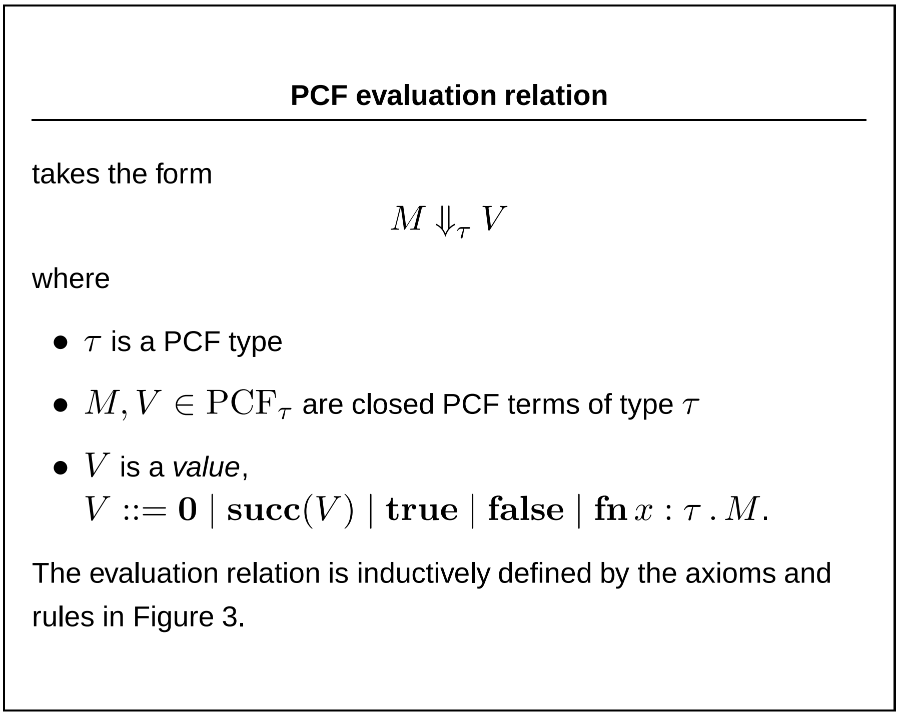
[1] See the Part IB course on **Computation Theory**.

5.4 Evaluation

We give the operational semantics of PCF in terms of an inductively defined relation of evaluation whose form is shown on Slide 40. As indicated there, the results of evaluation are PCF terms of a particular form, called *values* (and sometimes also called 'canonical forms'). The only values of type $bool$ are $\mathbf{true}$ and $\mathbf{false}$. The values of type nat are unary representations of natural numbers, $\mathbf{succ}^n(\mathbf{0})$ ($n \in \mathbb{N}$), where

$$\begin{cases} \mathbf{succ}^0(\mathbf{0}) & \stackrel{\text{def}}{=} \mathbf{0} \\ \mathbf{succ}^{n+1}(\mathbf{0}) & \stackrel{\text{def}}{=} \mathbf{succ}(\mathbf{succ}^n(\mathbf{0})). \end{cases}$$

Values at function types, being function abstractions $\mathbf{fn}\, x : \tau\, .\, M$, are more 'intensional' than those at the ground data types, since the body M is an unevaluated PCF term. The axioms and rules for generating the evaluation relation are given in Figure 3.



PCF evaluation relation

takes the form

$$M \Downarrow_\tau V$$

where

- τ is a PCF type
- $M, V \in \mathrm{PCF}_\tau$ are closed PCF terms of type τ
- V is a *value*,
  $V ::= \mathbf{0} \mid \mathbf{succ}(V) \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{fn}\, x : \tau\, .\, M.$

The evaluation relation is inductively defined by the axioms and rules in Figure 3.

Slide 40

Proposition 5.4.1. *Evaluation in PCF is deterministic: if both $M \Downarrow_\tau V$ and $M \Downarrow_\tau V'$ hold, then $V = V'$.*

Proof. By rule induction: one shows that

$$\{(M, \tau, V) \mid M \Downarrow_\tau V \;\&\; \forall V' \,(M \Downarrow_\tau V' \Rightarrow V = V')\}$$

is closed under the axioms and rules defining $\Downarrow$. We omit the details. □

$(\Downarrow_{\mathrm{val}})$ $\quad V \Downarrow_\tau V \quad$ (V a value of type τ)

$(\Downarrow_{\mathrm{succ}})$
$$\frac{M \Downarrow_{nat} V}{\mathbf{succ}(M) \Downarrow_{nat} \mathbf{succ}(V)}$$

$(\Downarrow_{\mathrm{pred}})$
$$\frac{M \Downarrow_{nat} \mathbf{succ}(V)}{\mathbf{pred}(M) \Downarrow_{nat} V}$$

$(\Downarrow_{\mathrm{zero1}})$
$$\frac{M \Downarrow_{nat} \mathbf{0}}{\mathbf{zero}(M) \Downarrow_{bool} \mathbf{true}}$$

$(\Downarrow_{\mathrm{zero2}})$
$$\frac{M \Downarrow_{nat} \mathbf{succ}(V)}{\mathbf{zero}(M) \Downarrow_{bool} \mathbf{false}}$$

$(\Downarrow_{\mathrm{if1}})$
$$\frac{M_1 \Downarrow_{bool} \mathbf{true} \quad M_2 \Downarrow_\tau V}{\mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \Downarrow_\tau V}$$

$(\Downarrow_{\mathrm{if2}})$
$$\frac{M_1 \Downarrow_{bool} \mathbf{false} \quad M_3 \Downarrow_\tau V}{\mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \Downarrow_\tau V}$$

$(\Downarrow_{\mathrm{cbn}})$
$$\frac{M_1 \Downarrow_{\tau \to \tau'} \mathbf{fn}\, x : \tau \,.\, M_1' \quad M_1'[M_2/x] \Downarrow_{\tau'} V}{M_1\, M_2 \Downarrow_{\tau'} V}$$

$(\Downarrow_{\mathrm{fix}})$
$$\frac{M\, \mathbf{fix}(M) \Downarrow_\tau V}{\mathbf{fix}(M) \Downarrow_\tau V}$$

Figure 3: Axioms and rules for PCF evaluation

Example 5.4.2. The proposition shows that every closed typeable term evaluates to at most one value. Of course there are some typeable terms that do not evaluate to anything. We write $M \not\Downarrow_\tau$ iff $M : \tau$ and $\nexists V .\, M \Downarrow_\tau V$. Then for example

$$\Omega_\tau \overset{\text{def}}{=} \mathbf{fix}(\mathbf{fn}\, x : \tau \,.\, x)$$

satisfies $\Omega_\tau \not\Downarrow_\tau$. (For if for some V there were a proof of $\mathbf{fix}(\mathbf{fn}\, x : \tau \,.\, x) \Downarrow_\tau V$, choose one of minimal height. This proof, call it $\mathcal{P}$, must look like

$$\dfrac{\dfrac{\dfrac{}{\mathbf{fn}\, x : \tau \,.\, x \Downarrow \mathbf{fn}\, x : \tau \,.\, x}\,(\Downarrow_{\mathrm{val}}) \qquad \begin{array}{c}\mathcal{P}' \\ \mathbf{fix}(\mathbf{fn}\, x : \tau \,.\, x) \Downarrow V\end{array}}{(\mathbf{fn}\, x : \tau \,.\, x)\,(\mathbf{fix}(\mathbf{fn}\, x : \tau \,.\, x)) \Downarrow V}\,(\Downarrow_{\mathrm{cbn}})}{\mathbf{fix}(\mathbf{fn}\, x : \tau \,.\, x) \Downarrow V}\,(\Downarrow_{\mathrm{fix}})$$

where $\mathcal{P}'$ is a strictly shorter proof of $\mathbf{fix}(\mathbf{fn}\, x : \tau \,.\, x) \Downarrow_\tau V$, which contradicts the minimality of $\mathcal{P}$.)

Remark 5.4.3. PCF evaluation can be defined in terms of a ‘one-step’ transition relation. Let the relation $M \to_\tau M'$ (for $M, M' \in \mathrm{PCF}_\tau$) be inductively defined by the axioms and rules in Figure 4. Then one can show that for all τ and $M, V \in \mathrm{PCF}_\tau$ with V a value

$$M \Downarrow_\tau V \;\Leftrightarrow\; M(\to_\tau)^* V$$

where $(\to_\tau)^*$ denotes the reflexive-transitive closure of the relation $\to_\tau$.

$$\frac{M \to_{nat} M'}{\mathbf{op}(M) \to_{\tau} \mathbf{op}(M')} \qquad \begin{array}{rl}\text{(where} & \mathbf{op} = \mathbf{succ}, \mathbf{pred} \mathbin{\&} \tau = nat, \\ \text{or} & \mathbf{op} = \mathbf{zero} \mathbin{\&} \tau = bool)\end{array}$$

$$\mathbf{pred}(\mathbf{succ}(V)) \to_{nat} V \quad (V \text{ a value of type } nat)$$

$$\mathbf{zero}(\mathbf{0}) \to_{bool} \mathbf{true}$$

$$\mathbf{zero}(\mathbf{succ}(V)) \to_{bool} \mathbf{false} \quad (V \text{ a value of type } nat)$$

$$\frac{M_1 \to_{bool} M_1'}{\mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \to_{\tau} \mathbf{if}\ M_1'\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3}$$

$$\mathbf{if}\ \mathbf{true}\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 \to_{\tau} M_1$$

$$\mathbf{if}\ \mathbf{false}\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 \to_{\tau} M_2$$

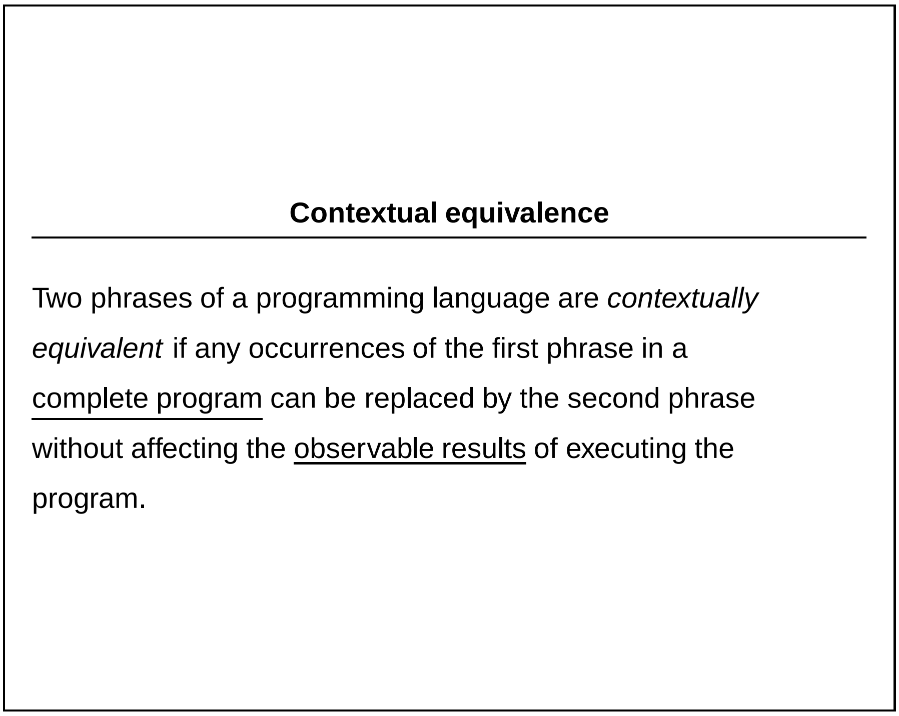
$$\frac{M_1 \to_{\tau \to \tau'} M_1'}{M_1\, M_2 \to_{\tau'} M_1'\, M_2}$$

$$(\mathbf{fn}\, x : \tau \,.\, M_1)\, M_2 \to_{\tau'} M_1[M_2/x]$$

$$\mathbf{fix}(M) \to_{\tau} M\, \mathbf{fix}(M)$$

Figure 4: Axioms and rules for PCF transition relation

5.5 Contextual equivalence



Contextual equivalence

Two phrases of a programming language are *contextually equivalent* if any occurrences of the first phrase in a complete program can be replaced by the second phrase without affecting the observable results of executing the program.

Slide 41

Slide 41 recalls (from the CST Part IB course on **Semantics of Programming Languages**) the general notion of contextual equivalence of phrases in a programming language. It is really a family of notions, parameterised by the particular choices one takes for what constitutes a ‘program’ in the language and what are the ‘observable results’ of executing such programs. For PCF it is reasonable to take the programs to be closed terms of type nat or $bool$ and to observe the values that result from evaluating such terms. This leads to the definition given on Slide 42.

Contextual equivalence of PCF terms

Given PCF terms M_1, M_2, PCF type τ, and a type environment Γ, the relation $\boxed{\Gamma \vdash M_1 \cong_{\mathrm{ctx}} M_2 : \tau}$ is defined to hold iff

- Both the typings $\Gamma \vdash M_1 : \tau$ and $\Gamma \vdash M_2 : \tau$ hold.
- For all PCF contexts $\mathcal{C}$ for which $\mathcal{C}[M_1]$ and $\mathcal{C}[M_2]$ are closed terms of type γ, *where* $\gamma = \mathit{nat}$ *or* $\gamma = \mathit{bool}$, and for all values $V : \gamma$,

$$\mathcal{C}[M_1] \Downarrow_\gamma V \;\Leftrightarrow\; \mathcal{C}[M_2] \Downarrow_\gamma V.$$

Slide 42

Definition 5.5.1 (Contexts)**.** The notation $\mathcal{C}[M]$ used on Slide 42 indicates a PCF term containing occurrences of a term M, and then $\mathcal{C}[M']$ is the term that results from replacing these occurrences by M'. More precisely, the *PCF contexts* are generated by the grammar for PCF expressions augmented by the symbol '$-$' representing a place, or 'hole' that can be filled with a PCF term:

$$\begin{aligned}\mathcal{C} ::= {} & - \mid \mathbf{0} \mid \mathbf{succ}(\mathcal{C}) \mid \mathbf{pred}(\mathcal{C}) \mid \mathbf{zero}(\mathcal{C}) \mid \mathbf{true} \mid \mathbf{false} \\ & \mid \mathbf{if}\ \mathcal{C}\ \mathbf{then}\ \mathcal{C}\ \mathbf{else}\ \mathcal{C} \mid x \mid \mathbf{fn}\, x : \tau \,.\, \mathcal{C} \mid \mathcal{C}\,\mathcal{C} \mid \mathbf{fix}(\mathcal{C})\end{aligned}$$

Given such a context $\mathcal{C}$,[1] we write $\mathcal{C}[M]$ for the PCF expression that results from replacing all the occurrences of $-$ in $\mathcal{C}$ by M. This form of substitution may well involve the capture of free variables in M by binders in $\mathcal{C}$. For example, if $\mathcal{C}$ is $\mathbf{fn}\, x : \tau \,.\, -$, then $\mathcal{C}[x]$ is $\mathbf{fn}\, x : \tau \,.\, x$. Nevertheless it is possible to show that if M and M' are α-convertible then so are $\mathcal{C}[M]$ and $\mathcal{C}[M']$. Hence the operation on PCF expressions sending M to $\mathcal{C}[M]$ induces a well-defined operation on PCF *terms* (= α-equivalence classes of expressions).

[1] It is common practice to write $\mathcal{C}[-]$ instead of $\mathcal{C}$ to indicate the symbol being used to mark the 'holes' in $\mathcal{C}$.

Notation 5.5.2. For closed PCF terms, we write

$$M_1 \cong_{\mathrm{ctx}} M_2 : \tau$$

for $\emptyset \vdash M_1 \cong_{\mathrm{ctx}} M_2 : \tau$.

Although $\cong_{\mathrm{ctx}}$ is a natural notion of semantic equivalence for PCF given its operational semantics, it is hard to work with, because of the universal quantification over contexts that occurs in the definition.

5.6 Denotational semantics

We aim to give a denotational semantics to PCF that is compositional (cf. Slide 2) and that matches its operational semantics. These requirements are made more precise on Slide 43.

PCF denotational semantics — aims

- PCF types $\tau \ \mapsto$ domains $[\![\tau]\!]$.
- Closed PCF terms $M : \tau \ \mapsto$ elements $[\![M]\!] \in [\![\tau]\!]$.
  Denotations of open terms will be continuous functions.
- *Compositionality*.
  In particular: $[\![M]\!] = [\![M']\!] \ \Rightarrow\ [\![\mathcal{C}[M]]\!] = [\![\mathcal{C}[M']]\!]$.
- *Soundness*.
  For any type τ, $M \Downarrow_\tau V \ \Rightarrow\ [\![M]\!] = [\![V]\!]$.
- *Adequacy*.
  For $\tau = bool$ or nat, $[\![M]\!] = [\![V]\!] \in [\![\tau]\!] \implies M \Downarrow_\tau V$.

Slide 43

The *soundness* and *adequacy* properties make precise the connection between the operational and denotational semantics for which we are aiming. Note that the adequacy property only involves the ‘ground’ datatypes nat and $bool$. One cannot expect such a property to hold at function types because of the ‘intensional’ nature of values at such types (mentioned above). Indeed such an adequacy property at function types would contradict the compositionality and soundness properties we want for $[\![-]\!]$, as the following example shows.

Example 5.6.1. Consider the following two PCF value terms of type $nat \to nat$:

$$V \stackrel{\text{def}}{=} \mathbf{fn}\, x : nat \,.\, (\mathbf{fn}\, y : nat \,.\, y)\, \mathbf{0} \quad \text{and} \quad V' \stackrel{\text{def}}{=} \mathbf{fn}\, x : nat \,.\, \mathbf{0}.$$

Now $V \not\Downarrow_{nat \to nat} V'$, since by ($\Downarrow_{\text{val}}$), $V \Downarrow_{nat \to nat} V \neq V'$ and by Proposition 5.4.1 evaluation is deterministic. However, the soundness and compositionality properties of $[\![-]\!]$ imply that $[\![V]\!] = [\![V']\!]$. For using ($\Downarrow_{\text{val}}$) and ($\Downarrow_{\text{cbn}}$) we have

$$(\mathbf{fn}\, y : nat \,.\, y)\, \mathbf{0} \Downarrow_{nat} \mathbf{0}.$$

So by soundness $[\![(\mathbf{fn}\, y : nat \,.\, y)\, \mathbf{0}]\!] = [\![\mathbf{0}]\!]$. Therefore by compositionality for $\mathcal{C}[-] \stackrel{\text{def}}{=} \mathbf{fn}\, x : nat \,.\, -$ we have

$$[\![\mathcal{C}[(\mathbf{fn}\, y : nat \,.\, y)\, \mathbf{0}]]\!] = [\![\mathcal{C}[\mathbf{0}]]\!]$$

i.e. $[\![V]\!] = [\![V']\!]$. □

As the theorem stated on Slide 44 shows, if we have a denotational semantics of PCF satisfying the properties on Slide 43, we can use it to establish instances of contextual equivalence by showing that terms have equal denotation. In many cases this is an easier task than proving contextual equivalence directly from the definition. The theorem on Slide 44 generalises to open terms: if the continuous functions that are the denotations of two open terms (of the same type for some type environment) are equal, then the terms are contextually equivalent.

Theorem. *For all types τ and closed terms $M_1, M_2 \in \mathrm{PCF}_\tau$, if $[\![M_1]\!]$ and $[\![M_2]\!]$ are equal elements of the domain $[\![\tau]\!]$, then $M_1 \cong_{\text{ctx}} M_2 : \tau$.*

Proof.

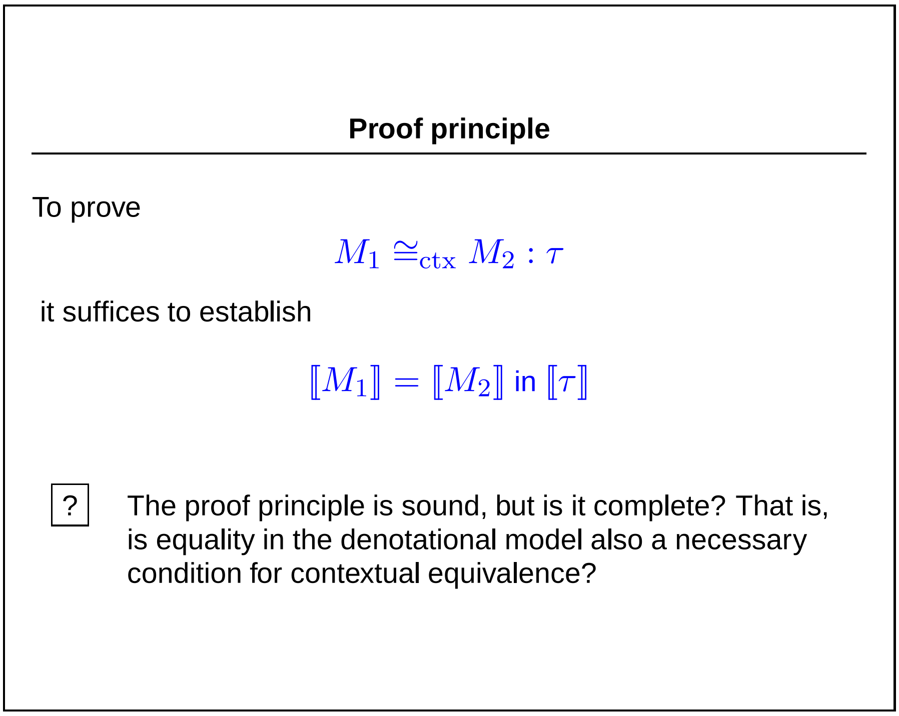
$$\mathcal{C}[M_1] \Downarrow_{nat} V \Rightarrow [\![\mathcal{C}[M_1]]\!] = [\![V]\!] \quad \text{(soundness)}$$

$$\Rightarrow [\![\mathcal{C}[M_2]]\!] = [\![V]\!] \quad \text{(compositionality on } [\![M_1]\!] = [\![M_2]\!]\text{)}$$

$$\Rightarrow \mathcal{C}[M_2] \Downarrow_{nat} V \quad \text{(adequacy)}$$

and symmetrically. □

Slide 44



Proof principle

To prove

$$M_1 \cong_{\mathrm{ctx}} M_2 : \tau$$

it suffices to establish

$$[\![M_1]\!] = [\![M_2]\!] \text{ in } [\![\tau]\!]$$

$\boxed{?}$ The proof principle is sound, but is it complete? That is, is equality in the denotational model also a necessary condition for contextual equivalence?

Slide 45

5.7 Exercises

Exercise 5.7.1. Carry out the suggested proof of Proposition 5.4.1.

Exercise 5.7.2. Recall that Church's fixpoint combinator in the untyped lambda calculus is $Y \overset{\text{def}}{=} \lambda f \,.\, (\lambda x \,.\, f\,(x\,x))\,(\lambda x \,.\, f\,(x\,x))$. Show that there are no PCF types τ_1, τ_2, τ_3 so that the typing relation

$$\emptyset \vdash \mathbf{fn}\, f : \tau_1 \,.\, (\mathbf{fn}\, x : \tau_2 \,.\, f\,(x\,x))\,(\mathbf{fn}\, x : \tau_2 \,.\, f\,(x\,x)) : \tau_3$$

is provable from the axioms and rules in Figure 2.

Exercise 5.7.3. Define the following PCF terms:

$$\begin{aligned}
\mathit{plus} &\overset{\text{def}}{=} \mathbf{fix}(\mathbf{fn}\, p : \mathit{nat} \to (\mathit{nat} \to \mathit{nat})\, .\, \mathbf{fn}\, x : \mathit{nat}\, .\, \mathbf{fn}\, y : \mathit{nat}\, . \\
&\qquad \mathbf{if\ zero}(y)\ \mathbf{then}\ x\ \mathbf{else}\ \mathbf{succ}(p\, x\, \mathbf{pred}(y)))
\end{aligned}$$

$$\begin{aligned}
\mathit{times} &\overset{\text{def}}{=} \mathbf{fix}(\mathbf{fn}\, t : \mathit{nat} \to (\mathit{nat} \to \mathit{nat})\, .\, \mathbf{fn}\, x : \mathit{nat}\, .\, \mathbf{fn}\, y : \mathit{nat}\, . \\
&\qquad \mathbf{if\ zero}(y)\ \mathbf{then}\ \mathbf{0}\ \mathbf{else}\ \mathit{plus}\, (t\, x\, \mathbf{pred}(y))\, x)
\end{aligned}$$

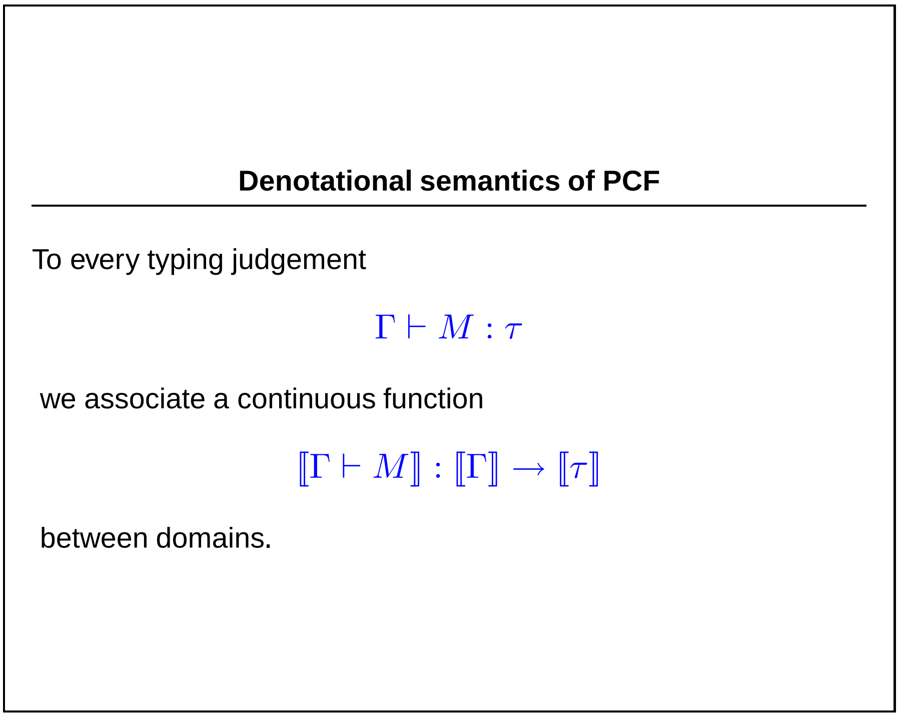
$$\begin{aligned}
\mathit{fact} &\overset{\text{def}}{=} \mathbf{fix}(\mathbf{fn}\, f : \mathit{nat} \to \mathit{nat}\, .\, \mathbf{fn}\, x : \mathit{nat}\, . \\
&\qquad \mathbf{if\ zero}(x)\ \mathbf{then}\ \mathbf{succ}(\mathbf{0})\ \mathbf{else}\ \mathit{times}\, x (f\, \mathbf{pred}(x))).
\end{aligned}$$

Show by induction on $n \in \mathbb{N}$ that for all $m \in \mathbb{N}$

$$\begin{aligned}
\mathit{plus}\, \mathbf{succ}^m(\mathbf{0})\, \mathbf{succ}^n(\mathbf{0}) &\Downarrow_{nat} \mathbf{succ}^{m+n}(\mathbf{0}) \\
\mathit{times}\, \mathbf{succ}^m(\mathbf{0})\, \mathbf{succ}^n(\mathbf{0}) &\Downarrow_{nat} \mathbf{succ}^{m*n}(\mathbf{0}) \\
\mathit{fact}\, \mathbf{succ}^n(\mathbf{0}) &\Downarrow_{nat} \mathbf{succ}^{!n}(\mathbf{0}).
\end{aligned}$$

6 Denotational Semantics of PCF

We turn now to the task of showing that PCF has a denotational semantics with the properties of compositionality, soundness, and adequacy.



Denotational semantics of PCF

To every typing judgement

$$\Gamma \vdash M : \tau$$

we associate a continuous function

$$[\![\Gamma \vdash M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$$

between domains.

Slide 46

6.1 Denotation of types

For each PCF type τ, we define a domain $[\![\tau]\!]$ by induction on the structure of τ as on Slide 47.

Denotational semantics of PCF types

$$[\![nat]\!] \overset{\text{def}}{=} \mathbb{N}_\perp$$ (flat domain)

$$[\![bool]\!] \overset{\text{def}}{=} \mathbb{B}_\perp$$ (flat domain)

$$[\![\tau \to \tau']\!] \overset{\text{def}}{=} [\![\tau]\!] \to [\![\tau']\!]$$ (function domain).

where $\mathbb{N} = \{0, 1, 2, \dots\}$ and $\mathbb{B} = \{true, false\}$.

Slide 47

6.2 Denotation of terms

For each PCF term M and type environment Γ, recall from Proposition 5.3.1 that there is at most one type τ for which the typing relation $\Gamma \vdash M : \tau$ is derivable from the axioms and rules in Figure 2. We only give a denotational semantics to such typeable terms. Specifically, given such M and Γ, we will define a continuous function between domains

$$[\![\Gamma \vdash M]\!] : [\![\Gamma]\!] \to [\![\tau]\!] \tag{8}$$

where τ is the type for which $\Gamma \vdash M : \tau$ holds, and where $[\![\Gamma]\!]$ is the following dependent product domain (see Definition 3.2.3):

$$[\![\Gamma]\!] \overset{\text{def}}{=} \prod_{x \in dom(\Gamma)} [\![\Gamma(x)]\!]. \tag{9}$$

The elements of the domain (9) will be called Γ-*environments*: they are functions ρ mapping each variable x in the domain of definition of Γ to an element $\rho(x) \in [\![\Gamma(x)]\!]$ in the domain which is the denotation of the type $\Gamma(x)$ assigned to x by the type environment Γ.

Denotational semantics of PCF type environments

$$\begin{aligned} [\![\Gamma]\!] &\overset{\text{def}}{=} \textstyle\prod_{x \in dom(\Gamma)} [\![\Gamma(x)]\!] \qquad (\Gamma\text{-environments}) \\ &= \text{the domain of partial functions } \rho \text{ from variables} \\ & \text{ to domains such that } dom(\rho) = dom(\Gamma) \text{ and} \\ & \rho(x) \in [\![\Gamma(x)]\!] \text{ for all } x \in dom(\Gamma) \end{aligned}$$

Example:

1. For the empty type environment $\emptyset$,

$$[\![\emptyset]\!] = \{\, \bot \,\}$$

where $\bot$ denotes the unique partial function with $dom(\bot) = \emptyset$.

Slide 48

2. $[\![\langle x \mapsto \tau \rangle]\!] \;=\; (\{\, x \,\} \to [\![\tau]\!]) \cong \; [\![\tau]\!]$

3.

$$\begin{aligned} &[\![\langle x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n \rangle]\!] \\ &\quad \cong \; (\{\, x_1 \,\} \to [\![\tau_1]\!]) \times \ldots \times (\{\, x_n \,\} \to [\![\tau_n]\!]) \\ &\quad \cong \; [\![\tau_1]\!] \times \ldots \times [\![\tau_n]\!] \end{aligned}$$

Slide 49

The continuous function (8) is defined by induction on the structure of M, or equivalently, by induction on the derivation of the typing relation $\Gamma \vdash M : \tau$. The definition is given on Slides 50–54, where we show the effect of each function on a Γ-environment, ρ.

Denotational semantics of PCF terms, I

$$[\![\Gamma \vdash \mathbf{0}]\!](\rho) \overset{\text{def}}{=} 0 \in [\![nat]\!]$$

$$[\![\Gamma \vdash \mathbf{true}]\!](\rho) \overset{\text{def}}{=} true \in [\![bool]\!]$$

$$[\![\Gamma \vdash \mathbf{false}]\!](\rho) \overset{\text{def}}{=} false \in [\![bool]\!]$$

$$[\![\Gamma \vdash x]\!](\rho) \overset{\text{def}}{=} \rho(x) \in [\![\Gamma(x)]\!] \qquad \big(x \in dom(\Gamma)\big)$$

Slide 50

Denotational semantics of PCF terms, II

$$[\![\Gamma \vdash \mathbf{succ}(M)]\!](\rho)$$

$$\stackrel{\text{def}}{=} \begin{cases} [\![\Gamma \vdash M]\!](\rho) + 1 & \text{if } [\![\Gamma \vdash M]\!](\rho) \neq \bot \\ \bot & \text{if } [\![\Gamma \vdash M]\!](\rho) = \bot \end{cases}$$

$$[\![\Gamma \vdash \mathbf{pred}(M)]\!](\rho)$$

$$\stackrel{\text{def}}{=} \begin{cases} [\![\Gamma \vdash M]\!](\rho) - 1 & \text{if } [\![\Gamma \vdash M]\!](\rho) > 0 \\ \bot & \text{if } [\![\Gamma \vdash M]\!](\rho) = 0, \bot \end{cases}$$

$$[\![\Gamma \vdash \mathbf{zero}(M)]\!](\rho) \stackrel{\text{def}}{=} \begin{cases} \mathit{true} & \text{if } [\![\Gamma \vdash M]\!](\rho) = 0 \\ \mathit{false} & \text{if } [\![\Gamma \vdash M]\!](\rho) > 0 \\ \bot & \text{if } [\![\Gamma \vdash M]\!](\rho) = \bot \end{cases}$$

Slide 51

Denotational semantics of PCF terms, III

$$[\![\Gamma \vdash \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3]\!](\rho)$$

$$\stackrel{\text{def}}{=} \begin{cases} [\![\Gamma \vdash M_2]\!](\rho) & \text{if } [\![\Gamma \vdash M_1]\!](\rho) = \mathit{true} \\ [\![\Gamma \vdash M_3]\!](\rho) & \text{if } [\![\Gamma \vdash M_1]\!](\rho) = \mathit{false} \\ \bot & \text{if } [\![\Gamma \vdash M_1]\!](\rho) = \bot \end{cases}$$

$$[\![\Gamma \vdash M_1\, M_2]\!](\rho) \stackrel{\text{def}}{=} \big([\![\Gamma \vdash M_1]\!](\rho)\big)\,([\![\Gamma \vdash M_2]\!](\rho))$$

Slide 52

Denotational semantics of PCF terms, IV

$$
\begin{aligned}
&[\![\Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M]\!](\rho) \\
&\quad \overset{\text{def}}{=} \lambda d \in [\![\tau]\!] \,.\, [\![\Gamma[x \mapsto \tau] \vdash M]\!](\rho[x \mapsto d])
\end{aligned}
\qquad (x \notin dom(\Gamma)))
$$

NB: $\rho[x \mapsto d] \in [\![\Gamma[x \mapsto \tau]]\!]$ is the function mapping x to $d \in [\![\tau]\!]$ and otherwise acting like ρ.

Slide 53

Denotational semantics of PCF terms, V

$$[\![\Gamma \vdash \mathbf{fix}(M)]\!](\rho) \overset{\text{def}}{=} \mathit{fix}([\![\Gamma \vdash M]\!](\rho))$$

Recall that fix is the function assigning least fixed points to continuous functions.

Slide 54

Denotational semantics of PCF

Proposition. *For all typing judgements* $\Gamma \vdash M : \tau$*, the denotation*

$$[\![\Gamma \vdash M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$$

is a well-defined continous function.

Slide 55

$[\![\Gamma \vdash M]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$ is a well-defined continuous function because the base cases of the definition (on Slide 50) are continuous functions and at each induction step, in giving the denotation of a compound phrase in terms of the denotations of its immediate subphrases, we make use of constructions preserving continuity—as we now indicate.

0, true, and false: The denotation of these terms (Slide 50) are all functions that are constantly equal to a particular value. We noted in Example 2.3.8 that such functions are continuous.

variables: The denotation of a variable (Slide 50) is a projection function. We noted in Definition 3.2.3 that such functions are continuous, because of the way lubs are computed componentwise in dependent product domains.

succ, pred, and zero: We need to make use of the fact that composition of functions preserves continuity—see the Proposition on Slide 32. We leave its proof as a simple exercise. In particular, the denotation of $\mathbf{succ}(M)$ (Slide 51) is the composition

$$s_\bot \circ [\![\Gamma \vdash M]\!]$$

where by induction hypothesis $[\![\Gamma \vdash M]\!] : [\![\Gamma]\!] \to \mathbb{N}_\bot$ is a continuous function, and where $s_\bot : \mathbb{N}_\bot \to \mathbb{N}_\bot$ is the continuous function on the flat domain $\mathbb{N}_\bot$ induced, as in Proposition 3.1.1, by the function $s : \mathbb{N} \rightharpoonup \mathbb{N}$ mapping each n to $n + 1$.

Similarly

$$\llbracket \Gamma \vdash \mathbf{pred}(M) \rrbracket = p_\perp \circ \llbracket \Gamma \vdash M \rrbracket \text{ and } \llbracket \Gamma \vdash \mathbf{zero}(M) \rrbracket = z_\perp \circ \llbracket \Gamma \vdash M \rrbracket,$$

for suitable functions $p : \mathbb{N} \rightharpoonup \mathbb{N}$ and $z : \mathbb{N} \rightharpoonup \mathbb{B}$. (Only p is a properly partial function, undefined at 0; s and z are totally defined functions.)

conditional: By induction hypothesis we have continuous functions $\llbracket \Gamma \vdash M_1 \rrbracket : \llbracket \Gamma \rrbracket \to \mathbb{B}_\perp$, $\llbracket \Gamma \vdash M_2 \rrbracket : \llbracket \Gamma \rrbracket \to \llbracket \tau \rrbracket$, and $\llbracket \Gamma \vdash M_3 \rrbracket : \llbracket \Gamma \rrbracket \to \llbracket \tau \rrbracket$. Then $\llbracket \Gamma \vdash \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \rrbracket$ is continuous because we can express the definition on Slide 52 in terms of composition, the pairing operation of Proposition 3.2.1, and the continuous function : $\mathbb{B}_\perp \times (\llbracket \tau \rrbracket \times \llbracket \tau \rrbracket) \to \llbracket \tau \rrbracket$ of Proposition 3.2.2:

$$\llbracket \Gamma \vdash \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \rrbracket = \mathit{if} \circ \langle \llbracket \Gamma \vdash M_1 \rrbracket, \langle \llbracket \Gamma \vdash M_2 \rrbracket, \llbracket \Gamma \vdash M_3 \rrbracket \rangle \rangle.$$

application: By induction hypothesis we have continuous functions $\llbracket \Gamma \vdash M_1 \rrbracket : \llbracket \Gamma \rrbracket \to (\llbracket \tau \rrbracket \to \llbracket \tau' \rrbracket)$ and $\llbracket \Gamma \vdash M_2 \rrbracket : \llbracket \Gamma \rrbracket \to \llbracket \tau \rrbracket$. Then $\llbracket \Gamma \vdash M_1\, M_2 \rrbracket$ is continuous because we can express the definition on Slide 52 in terms of composition, pairing, and the evaluation function $ev : (\llbracket \tau \rrbracket \to \llbracket \tau' \rrbracket) \times \llbracket \tau \rrbracket \to \llbracket \tau' \rrbracket$ that we proved continuous in Proposition 3.3.1:

$$\llbracket \Gamma \vdash M_1\, M_2 \rrbracket = ev \circ \langle \llbracket \Gamma \vdash M_1 \rrbracket, \llbracket \Gamma \vdash M_2 \rrbracket \rangle.$$

function abstraction: By induction hypothesis we have a continuous function $\llbracket \Gamma[x \mapsto \tau] \vdash M \rrbracket : \llbracket \Gamma[x \mapsto \tau] \rrbracket \to \llbracket \tau' \rrbracket$ with $x \notin dom(\Gamma)$. Note that each $\Gamma[x \mapsto \tau]$-environment, $\rho' \in \llbracket \Gamma[x \mapsto \tau] \rrbracket$, can be uniquely expressed as $\rho[x \mapsto d]$, where ρ is the restriction of the function ρ' to $dom(\Gamma)$ and where $d = \rho'(x)$; furthermore the partial order respects this decomposition: $\rho_1[x \mapsto d_1] \sqsubseteq \rho_2[x \mapsto d_2]$ in $\llbracket \Gamma[x \mapsto \tau] \rrbracket$ iff $\rho_1 \sqsubseteq \rho_2$ in $\llbracket \Gamma \rrbracket$ and $d_1 \sqsubseteq d_2$ in $\llbracket \tau \rrbracket$. Thus we can identify $\llbracket \Gamma[x \mapsto \tau] \rrbracket$ with the binary product domain $\llbracket \Gamma \rrbracket \times \llbracket \tau \rrbracket$. So we can apply the ‘Currying’ operation of Proposition 3.3.1 to obtain a continuous function

$$cur(\llbracket \Gamma[x \mapsto \tau] \vdash M \rrbracket) : \llbracket \Gamma \rrbracket \ \to \ (\llbracket \tau \rrbracket \to \llbracket \tau' \rrbracket) = \llbracket \tau \to \tau' \rrbracket.$$

But this is precisely the function used to define $\llbracket \Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M \rrbracket$ on Slide 53.

fixpoints: By induction hypothesis we have a continuous function $\llbracket \Gamma \vdash M \rrbracket : \llbracket \Gamma \rrbracket \to \llbracket \tau \to \tau \rrbracket$. Now $\llbracket \tau \to \tau \rrbracket$ is the function domain $\llbracket \tau \rrbracket \to \llbracket \tau \rrbracket$ and from the definition on Slide 53 we have that $\llbracket \Gamma \vdash \mathbf{fix}(M) \rrbracket = \mathit{fix} \circ \llbracket \Gamma \vdash M \rrbracket$ is the composition with the function $\mathit{fix} : (\llbracket \tau \rrbracket \to \llbracket \tau \rrbracket) \to \llbracket \tau \rrbracket$ assigning least fixpoints, which we proved continuous in the Proposition on Slide 33.

Denotations of closed terms

If $M \in \mathrm{PCF}_\tau$, then by definition $\emptyset \vdash M : \tau$ holds, so we get $[\![\emptyset \vdash M]\!] : [\![\emptyset]\!] \to [\![\tau]\!]$.

When $\Gamma = \emptyset$, the only Γ-environment is the totally undefined partial function—call it $\bot$.

So in this case $[\![\Gamma]\!]$ is a one-element domain, $\{\bot\}$. Continuous functions $f : \{\bot\} \to D$ are in bijection with elements $f(\bot) \in D$, and in particular we can identify the denotation of closed PCF terms with elements of the domain denoting their type:

$$\boxed{[\![M]\!] \overset{\text{def}}{=} [\![\emptyset \vdash M]\!](\bot) \in [\![\tau]\!] \qquad (M \in \mathrm{PCF}_\tau)}$$

Slide 56

6.3 Compositionality

The fact that the denotational semantics of PCF terms is *compositional*—i.e. that the denotation of a compound term is a function of the denotations of its immediate subterms—is part and parcel of the definition of $[\![\Gamma \vdash M]\!]$ by induction on the structure of M. So in particular, each of the ways of constructing terms in PCF respects equality of denotations: this is summarised in Figure 5. Then the property of closed terms stated on Slide 43, *viz.*

$$[\![M]\!] = [\![M']\!] \ \Rightarrow\ [\![\mathcal{C}[M]]\!] = [\![\mathcal{C}[M']]\!]$$

follows from this by induction on the structure of the context $\mathcal{C}[-]$. More generally, for open terms we have

Proposition 6.3.1. *Suppose*

$$[\![\Gamma \vdash M]\!] = [\![\Gamma \vdash M']\!] : [\![\Gamma]\!] \to [\![\tau]\!]$$

and that $\mathcal{C}[-]$ is a PCF context such that $\Gamma' \vdash \mathcal{C}[M] : \tau'$ and $\Gamma' \vdash \mathcal{C}[M'] : \tau'$ hold for some some type τ' and some type environment Γ'. Then

$$[\![\Gamma' \vdash \mathcal{C}[M]]\!] = [\![\Gamma' \vdash \mathcal{C}[M']]\!] : [\![\Gamma']\!] \to [\![\tau']\!].$$

- If $[\![\Gamma \vdash M]\!] = [\![\Gamma \vdash M']\!] : [\![\Gamma]\!] \to [\![nat]\!]$, then

  $$[\![\Gamma \vdash \mathbf{op}(M)]\!] = [\![\Gamma \vdash \mathbf{op}(M')]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$$

  (where $\mathbf{op} = \mathbf{succ}, \mathbf{pred}$ and $\tau = nat$, or $\mathbf{op} = \mathbf{zero}$ and $\tau = bool$).

- If $[\![\Gamma \vdash M_1]\!] = [\![\Gamma \vdash M_1']\!] : [\![\Gamma]\!] \to [\![bool]\!]$, $[\![\Gamma \vdash M_2]\!] = [\![\Gamma \vdash M_2']\!] : [\![\Gamma]\!] \to [\![\tau]\!]$, and $[\![\Gamma \vdash M_3]\!] = [\![\Gamma \vdash M_3']\!] : [\![\Gamma]\!] \to [\![\tau]\!]$, then

  $$[\![\Gamma \vdash \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3]\!] = [\![\Gamma \vdash \mathbf{if}\ M_1'\ \mathbf{then}\ M_2'\ \mathbf{else}\ M_3']\!] : [\![\Gamma]\!].$$

- If $[\![\Gamma \vdash M_1]\!] = [\![\Gamma \vdash M_1']\!] : [\![\Gamma]\!] \to [\![\tau \to \tau']\!]$ and $[\![\Gamma \vdash M_2]\!] = [\![\Gamma \vdash M_2']\!] : [\![\Gamma]\!] \to [\![\tau]\!]$, then

  $$[\![\Gamma \vdash M_1\, M_2]\!] = [\![\Gamma \vdash M_1'\, M_2']\!] : [\![\Gamma]\!] \to [\![\tau']\!].$$

- If $[\![\Gamma[x \mapsto \tau] \vdash M]\!] = [\![\Gamma[x \mapsto \tau] \vdash M']\!] : [\![\Gamma[x \mapsto \tau]]\!] \to [\![\tau']\!]$, then

  $$[\![\Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M]\!] = [\![\Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M']\!] : [\![\Gamma]\!] \to [\![\tau \to \tau']\!].$$

- If $[\![\Gamma \vdash M]\!] = [\![\Gamma \vdash M']\!] : [\![\Gamma]\!] \to [\![\tau \to \tau]\!]$, then

  $$[\![\Gamma \vdash \mathbf{fix}(M)]\!] = [\![\Gamma \vdash \mathbf{fix}(M')]\!] : [\![\Gamma]\!] \to [\![\tau]\!].$$

Figure 5: Compositionality properties of $[\![-]\!]$

Substitution property of $[\![-]\!]$

Proposition. *Suppose*

$$\begin{gathered}\Gamma \vdash M : \tau \\ \Gamma[x \mapsto \tau] \vdash M' : \tau'\end{gathered}$$

(so that by Proposition 5.3.1*(ii) we also have* $\Gamma \vdash M'[M/x] : \tau'$*). Then for all* $\rho \in [\![\Gamma]\!]$

$$[\![\Gamma \vdash M'[M/x]]\!](\rho) = \\ [\![\Gamma[x \mapsto \tau] \vdash M']\!](\rho[x \mapsto [\![\Gamma \vdash M]\!]]).$$

In particular when $\Gamma = \emptyset$*,* $[\![x \mapsto \tau \vdash M']\!] : [\![\tau]\!] \to [\![\tau']\!]$ *and*

$$\boxed{[\![M'[M/x]]\!] = [\![x \mapsto \tau \vdash M']\!]([\![M]\!])}$$

Slide 57

The substitution property stated on Slide 57 gives another aspect of the compositional nature of the denotational semantics of PCF. It can be proved by induction on the structure of the term M'.

6.4 Soundness

The second of the aims mentioned on Slide 43 is to show that if a closed PCF term M evaluates to a value V in the operational semantics, then M and V have the same denotation.

Theorem 6.4.1. *For all PCF types τ and all closed terms $M, V \in \mathrm{PCF}_\tau$ with V a value, if $M \Downarrow_\tau V$ is derivable from the axioms and rules in Figure* 3 *then $[\![M]\!]$ and $[\![V]\!]$ are equal elements of the domain $[\![\tau]\!]$.*

Proof. One uses Rule Induction for the inductively defined relation $\Downarrow$. Specifically, defining

$$\Phi(M, \tau, V) \overset{\mathrm{def}}{\Leftrightarrow} [\![M]\!] = [\![V]\!] \in [\![\tau]\!]$$

one shows that the property $\Phi(M, \tau, V)$ is closed under the axioms and rules in Figure 3. We give the argument for rules ($\Downarrow_{\mathrm{cbn}}$) and ($\Downarrow_{\mathrm{fix}}$), and leave the others as easy exercises.

Case ($\Downarrow_{\mathrm{cbn}}$). Suppose

$$\llbracket M_1 \rrbracket = \llbracket \mathbf{fn}\, x : \tau \,.\, M_1' \rrbracket \in \llbracket \tau \to \tau' \rrbracket \tag{10}$$

$$\llbracket M_1'[M_2/x] \rrbracket = \llbracket V \rrbracket \in \llbracket \tau' \rrbracket. \tag{11}$$

We have to prove that $\llbracket M_1\, M_2 \rrbracket = \llbracket V \rrbracket \in \llbracket \tau' \rrbracket$. But

$$\begin{aligned}
\llbracket M_1\, M_2 \rrbracket &= \llbracket M_1 \rrbracket(\llbracket M_2 \rrbracket) && \text{by Slide 52}\\
&= \llbracket \mathbf{fn}\, x : \tau \,.\, M_1' \rrbracket(\llbracket M_2 \rrbracket) && \text{by (10)}\\
&= (\lambda d \in \llbracket \tau \rrbracket \,.\, \llbracket x \mapsto \tau \vdash M_1' \rrbracket(d))(\llbracket M_2 \rrbracket) && \text{by Slide 53}\\
&= \llbracket x \mapsto \tau \vdash M_1' \rrbracket(\llbracket M_2 \rrbracket) &&\\
&= \llbracket M_1'[M_2/x] \rrbracket && \text{by Slide 57}\\
&= \llbracket V \rrbracket && \text{by (11).}
\end{aligned}$$

Case ($\Downarrow_{\mathrm{fix}}$). Suppose

$$\llbracket M\, \mathbf{fix}(M) \rrbracket = \llbracket V \rrbracket \in \llbracket \tau \rrbracket. \tag{12}$$

We have to prove that $\llbracket \mathbf{fix}(M) \rrbracket = \llbracket V \rrbracket \in \llbracket \tau \rrbracket$. But

$$\begin{aligned}
\llbracket \mathbf{fix}(M) \rrbracket &= \mathit{fix}(\llbracket M \rrbracket) && \text{by Slide 53}\\
&= \llbracket M \rrbracket(\mathit{fix}(\llbracket M \rrbracket)) && \text{by fixed point property of } \mathit{fix}\\
&= \llbracket M \rrbracket\, \llbracket \mathbf{fix}(M) \rrbracket && \text{by Slide 53}\\
&= \llbracket M\, \mathbf{fix}(M) \rrbracket && \text{by Slide 52}\\
&= \llbracket V \rrbracket && \text{by (12).}
\end{aligned}$$

□

We have now established two of the three properties of the denotational semantics of PCF stated on Slide 43 (and which in particular are needed to use denotational equality to prove PCF contextual equivalences). The third property, *adequacy*, is not so easy to prove as are the first two. We postpone the proof until we have introduced a useful principle of induction tailored to reasoning about least fixed points. This is the subject of the next section.

6.5 Exercises

Exercise 6.5.1. Prove the Proposition on Slide 57.

Exercise 6.5.2. Defining $\Omega_\tau \overset{\text{def}}{=} \mathbf{fix}(\mathbf{fn}\, x : \tau \,.\, x)$, show that $\llbracket \Omega_\tau \rrbracket$ is the least element $\bot$ of the domain $\llbracket \tau \rrbracket$. Deduce that $\llbracket \mathbf{fn}\, x : \tau \,.\, \Omega_\tau \rrbracket = \llbracket \Omega_{\tau \to \tau} \rrbracket$.

7 Relating Denotational and Operational Semantics

We have already seen (in Section 6.4) that the denotational semantics of PCF given in Section 6 is *sound* for the operational semantics, in the sense defined on Slide 43. Here we prove the property of *adequacy* defined on that slide. So we have to prove for any closed PCF terms M and V of type $\tau = \mathit{nat}$ or bool with V a value, that

$$[\![M]\!] = [\![V]\!] \ \Rightarrow\ M \Downarrow_\tau V.$$

Perhaps surprisingly, this is not easy to prove. We will employ a method due to Plotkin (although not quite the one used in his original paper on PCF, Plotkin 1977) and Mulmuley (1987) making use of the following notion of 'formal approximation' relations.

Adequacy

For any closed PCF terms M and V of *ground* type $\gamma \in \{\mathit{nat}, \mathit{bool}\}$ with V a value

$$[\![M]\!] = [\![V]\!] \in [\![\gamma]\!] \implies M \Downarrow_\gamma V\,.$$

NB. Adequacy does not hold at function types:

$$[\![\mathbf{fn}\, x : \tau .\, (\mathbf{fn}\, y : \tau .\, y)\, x]\!] \quad = \quad [\![\mathbf{fn}\, x : \tau .\, x]\!] \quad : [\![\tau]\!] \to [\![\tau]\!]$$

but

$$\mathbf{fn}\, x : \tau .\, (\mathbf{fn}\, y : \tau .\, y)\, x \not\Downarrow_{\tau \to \tau} \mathbf{fn}\, x : \tau .\, x$$

Slide 58

7.1 Formal approximation relations

We define a certain family of binary relations

$$\lhd_\tau \ \subseteq\ [\![\tau]\!] \times \mathrm{PCF}_\tau$$

indexed by the PCF types, τ. Thus each $\lhd_\tau$ relates elements of the domain $[\![\tau]\!]$ to closed PCF terms of type τ. We use infix notation and write $d \lhd_\tau M$ instead of $(d, M) \in \lhd_\tau$. The definition of these relations $\lhd_\tau$ proceeds *by induction on the* *structure of the type* τ and is given on Slide 59. (Read the definition in conjunction with the definition of $[\![\tau]\!]$ given on Slide 47.)

The key property of the relations $\lhd_\tau$ is that they are respected by the various syntax-forming operations of the PCF language. This is summed up by the Proposition on Slide 60 which makes use of the following terminology.

Definition 7.1.1. For each typing environment Γ (= a finite partial function from variables to PCF types), a Γ-*substitution* σ is a function mapping each variable $x \in dom(\Gamma)$ to a closed PCF term $\sigma(x)$ of type $\Gamma(x)$. Recall from Section 6.2 that a Γ-environment ρ is a function mapping each variable $x \in dom(\Gamma)$ to an element $\rho(x)$ of the domain $[\![\Gamma(x)]\!]$. We define

$$\rho \lhd_\Gamma \sigma \overset{\text{def}}{\Leftrightarrow} \forall x \in dom(\Gamma) \,.\, \rho(x) \lhd_{\Gamma(x)} \sigma(x).$$

Definition of $d \lhd_\tau M$ **(**$d \in [\![\tau]\!], M \in \mathrm{PCF}_\tau$**)**

$$d \lhd_{nat} M \overset{\text{def}}{\Leftrightarrow} (d \in \mathbb{N} \;\Rightarrow\; M \Downarrow_{nat} \mathbf{succ}^d(\mathbf{0}))$$

$$\begin{aligned} d \lhd_{bool} M \overset{\text{def}}{\Leftrightarrow} & (d = true \;\Rightarrow\; M \Downarrow_{bool} \mathbf{true}) \\ & \& \; (d = false \;\Rightarrow\; M \Downarrow_{bool} \mathbf{false}) \end{aligned}$$

$$d \lhd_{\tau \to \tau'} M \overset{\text{def}}{\Leftrightarrow} \forall e, N \; (e \lhd_\tau N \;\Rightarrow\; d(e) \lhd_{\tau'} M\, N)$$

Slide 59

Fundamental property of the relations $\lhd_\tau$

Proposition. *If $\Gamma \vdash M : \tau$ is a valid PCF typing, then for all Γ-environments ρ and all Γ-substitutions σ*

$$\rho \lhd_\Gamma \sigma \;\Rightarrow\; [\![\Gamma \vdash M]\!](\rho) \lhd_\tau M[\sigma]$$

- $\rho \lhd_\Gamma \sigma$ means that $\rho(x) \lhd_{\Gamma(x)} \sigma(x)$ holds for each $x \in dom(\Gamma)$.
- $M[\sigma]$ is the PCF term resulting from the simultaneous substitution of $\sigma(x)$ for x in M, each $x \in dom(\Gamma)$.

Slide 60

Note that the Fundamental Property of $\lhd_\tau$ given on Slide 60 specialises in case $\Gamma = \emptyset$ to give

$$[\![M]\!] \lhd_\tau M$$

for all types τ and all closed PCF terms $M : \tau$. (Here we are using the notation for denotations of closed terms introduced on Slide 56.) Using this, we can complete the proof of the adequacy property, as shown on Slide 61.

Proof of $[\![M]\!] = [\![V]\!] \;\Rightarrow\; M \Downarrow_\tau V \quad (\tau = nat, bool)$

Case $\tau = nat$**.**

$V = \mathbf{succ}^n(\mathbf{0})$ for some $n \in \mathbb{N}$ and hence

$$\begin{array}{ll} [\![M]\!] = [\![\mathbf{succ}^n(\mathbf{0})]\!] & \\ \quad \Rightarrow n = [\![M]\!] \lhd_\tau M & \text{by Fundamental Property (Slide 60)} \\ \quad \Rightarrow M \Downarrow \mathbf{succ}^n(\mathbf{0}) & \text{by definition of } \lhd_{nat} \end{array}$$

Case $\tau = bool$ is similar.

Slide 61

7.2 Proof of the Fundamental Property of $\lhd$

To prove the Proposition on Slide 60 we need the following properties of the formal approximation relations.

Lemma 7.2.1.

(i) $\perp \lhd_\tau M$ *holds for all* $M \in \mathrm{PCF}_\tau$.

(ii) For each $M \in \mathrm{PCF}_\tau$*,* $\{d \mid d \lhd_\tau M\}$ *is a chain-closed subset of the domain* $[\![\tau]\!]$*. Hence by (i), it is also an admissible subset (cf. Slide* 34*).*

(iii) If $d_2 \sqsubseteq d_1$*,* $d_1 \lhd_\tau M_1$*, and* $\forall V\ (M_1 \Downarrow_\tau V \;\Rightarrow\; M_2 \Downarrow_\tau V)$*, then* $d_2 \lhd_\tau M_2$*.*

Proof. Each of these properties follows easily by induction on the structure of τ, using the definitions of $\lhd_\tau$ and of the evaluation relation $\Downarrow_\tau$. □

Proof of the Proposition on Slide 60 **[NON-EXAMINABLE]**. We use Rule Induction for the inductively defined typing relation $\Gamma \vdash M : \tau$. Define

$$\Phi(\Gamma, M, \tau) \overset{\text{def}}{\Leftrightarrow} \Gamma \vdash M : \tau \;\;\&\;\; \forall \rho, \sigma\ (\rho \lhd_\Gamma \sigma \;\;\Rightarrow\;\; [\![\Gamma \vdash M]\!](\rho) \lhd_\tau M[\sigma])$$

Then it suffices to show that Φ is closed under the axioms and rules in Figure 2 inductively defining the typing relation.

Case ($:_0$). $\Phi(\Gamma, \mathbf{0}, nat)$ holds because $0 \lhd_{nat} \mathbf{0}$.

Case ($:_{\text{succ}}$). We have to prove that $\Phi(\Gamma, M, nat)$ implies $\Phi(\Gamma, \mathbf{succ}(M), nat)$. But this follows from the easily verified fact that

$$d \lhd_{nat} M \;\Rightarrow\; s_\perp(d) \lhd_{nat} \mathbf{succ}(M)$$

where $s_\perp : \mathbb{N}_\perp \to \mathbb{N}_\perp$ is the continuous function used in Section 6.2 to describe the denotation of successor terms, $\mathbf{succ}(M)$.

Cases ($:_{\text{pred}}$) and ($:_{\text{zero}}$) are similar to the previous case.

Case ($:_{\text{bool}}$). $\Phi(\Gamma, \mathbf{true}, bool)$ holds because $true \lhd_{bool} \mathbf{true}$. Similarly for $\Phi(\Gamma, \mathbf{false}, bool)$.

Case ($:_{\text{if}}$). It suffices to show that if $d_1 \lhd_{bool} M_1$, $d_2 \lhd_\tau M_2$, and $d_3 \lhd_\tau M_3$, then

(13) $$\mathit{if}(d_1, (d_2, d_3)) \lhd_\tau \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3$$

where if is the continuous function : $\mathbb{B}_\perp \times ([\![\tau]\!] \times [\![\tau]\!]) \to [\![\tau]\!]$ of Proposition 3.2.2 that was used in Section 6.2 to describe the denotation of conditional terms. If $d_1 = \perp \in \mathbb{B}_\perp$, then $\mathit{if}(d_1, (d_2, d_3)) = \perp$ and (13) holds by Lemma 7.2.1(i). So we may assume $d_1 \neq \perp$, in which case either $d_1 = true$ or $d_1 = false$. We consider the case $d_1 = true$; the argument for the other case is similar.

Since $true = d_1 \lhd_{bool} M_1$, by the definition of $\lhd_{bool}$ (Slide 59) we have $M_1 \Downarrow_{bool} \mathbf{true}$. It follows from rule ($\Downarrow_{\text{if1}}$) in Figure 3 that

$$\forall V\ (M_2 \Downarrow_\tau V \;\Rightarrow\; \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \Downarrow_\tau V).$$

So Lemma 7.2.1(iii) applied to $d_2 \lhd_\tau M_2$ yields that

$$d_2 \lhd_\tau \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3$$

and then since $d_2 = \mathit{if}(true, (d_2, d_3)) = \mathit{if}(d_1, (d_2, d_3))$, we get (13), as required.

Case ($:_{\text{var}}$). $\Phi(\Gamma, x, \Gamma(x))$ holds because if $\rho \lhd_\Gamma \sigma$, then for all $x \in dom(\Gamma)$ we have $[\![\Gamma \vdash x]\!](\rho) \overset{\text{def}}{=} \rho(x) \lhd_{\Gamma(x)} \sigma(x) \overset{\text{def}}{=} x[\sigma]$.

Case ($:_{\text{fn}}$). Suppose $\Phi(\Gamma[x \mapsto \tau], M, \tau')$ and $\rho \lhd_\Gamma \sigma$ hold. We have to show that $[\![\Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M]\!](\rho) \lhd_{\tau \to \tau'} (\mathbf{fn}\, x : \tau \,.\, M)[\sigma]$, i.e. that $d \lhd_\tau N$ implies

(14) $$[\![\Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M]\!](\rho)(d) \lhd_{\tau'} ((\mathbf{fn}\, x : \tau \,.\, M)[\sigma])N.$$

From Slide 53 we have

(15) $$[\![\Gamma \vdash \mathbf{fn}\, x : \tau \,.\, M]\!](\rho)(d) = [\![\Gamma[x \mapsto \tau] \vdash M]\!](\rho[x \mapsto d]).$$

Since $(\mathbf{fn}\, x : \tau \,.\, M)[\sigma] = \mathbf{fn}\, x : \tau \,.\, M[\sigma]$ and $(M[\sigma])[N/x] = M[\sigma[x \mapsto N]]$, by rule ($\Downarrow_{\mathrm{cbn}}$) in Figure 3 we have

(16) $$\forall V\ (M[\sigma[x \mapsto N]] \Downarrow_{\tau'} V \ \Rightarrow\ ((\mathbf{fn}\, x : \tau \,.\, M)[\sigma])N \Downarrow_{\tau'} V).$$

Since $\rho \lhd_\Gamma \sigma$ and $d \lhd_\tau N$, we have $\rho[x \mapsto d] \lhd_{\Gamma[x \mapsto \tau]} \sigma[x \mapsto N]$; so by $\Phi(\Gamma[x \mapsto \tau], M, \tau')$ we have

$$[\![\Gamma[x \mapsto \tau] \vdash M]\!](\rho[x \mapsto d]) \lhd_{\tau'} M[\sigma[x \mapsto N]].$$

Then (14) follows from this by applying Lemma 7.2.1(iii) to (15) and (16).

Case ($:_{\mathrm{app}}$). It suffices to show that if $d_1 \lhd_{\tau \to \tau'} M_1$ and $d_2 \lhd_\tau M_2$, then $d_1(d_2) \lhd_{\tau'} M_1\, M_2$. But this follows immediately from the definition of $\lhd_{\tau \to \tau'}$.

Case ($:_{\mathrm{fix}}$). Suppose $\Phi(\Gamma, M, \tau \to \tau)$ holds. For any $\rho \lhd_\Gamma \sigma$, we have to prove that

(17) $$[\![\Gamma \vdash \mathbf{fix}(M)]\!](\rho) \lhd_\tau \mathbf{fix}(M)[\sigma].$$

Referring to Slide 53, we have $[\![\Gamma \vdash \mathbf{fix}(M)]\!](\rho) = \mathit{fix}(f)$, where $f \overset{\mathrm{def}}{=} [\![\Gamma \vdash M]\!](\rho)$. By Lemma 7.2.1(ii)

$$S \overset{\mathrm{def}}{=} \{d \mid d \lhd_\tau \mathbf{fix}(M)[\sigma]\}$$

is an admissible subset of the domain $[\![\tau]\!]$. So by Scott's Fixed Point Induction Principle (Slide 35) to prove (17) it suffices to prove

$$\forall d \in [\![\tau]\!]\ (d \in S \ \Rightarrow\ f(d) \in S).$$

Now since $\rho \lhd_\Gamma \sigma$, by $\Phi(\Gamma, M, \tau \to \tau)$ and by definition of f we have $f \lhd_{\tau \to \tau} M[\sigma]$. So if $d \in S$, i.e. $d \lhd_\tau \mathbf{fix}(M)[\sigma]$, then by definition of $\lhd_{\tau \to \tau}$, it is the case that

(18) $$f(d) \lhd_\tau (M[\sigma])(\mathbf{fix}(M)[\sigma]).$$

Rule ($\Downarrow_{\mathrm{fix}}$) in Figure 3 implies

(19) $$\forall V\ ((M[\sigma])(\mathbf{fix}(M)[\sigma]) \Downarrow_\tau V \ \Rightarrow\ \mathbf{fix}(M)[\sigma] \Downarrow_\tau V).$$

Then applying Lemma 7.2.1(iii) to (18) and (19) yields $f(d) \lhd_\tau \mathbf{fix}(M)[\sigma]$, i.e. $f(d) \in S$, as required. □

7.3 Extensionality

Recall the notion of contextual equivalence of PCF terms from Slide 42. The *contextual preorder* is the one-sided version of this relation defined on Slide 62. Clearly

$$\Gamma \vdash M_1 \cong_{\mathrm{ctx}} M_2 : \tau \;\Leftrightarrow\; (\Gamma \vdash M_1 \leq_{\mathrm{ctx}} M_2 : \tau \;\;\&\;\; \Gamma \vdash M_2 \leq_{\mathrm{ctx}} M_1 : \tau).$$

As usual we write $M_1 \leq_{\mathrm{ctx}} M_2 : \tau$ for $\emptyset \vdash M_1 \leq_{\mathrm{ctx}} M_2 : \tau$ in case M_1 and M_2 are closed terms.

The formal approximation relations $\lhd_\tau$ actually characterise the PCF contextual preorder between closed terms, in the sense shown on Slide 63.

Contextual preorder between PCF terms

Given PCF terms M_1, M_2, PCF type τ, and a type environment Γ, the relation $\boxed{\Gamma \vdash M_1 \leq_{\mathrm{ctx}} M_2 : \tau}$ is defined to hold iff

- Both the typings $\Gamma \vdash M_1 : \tau$ and $\Gamma \vdash M_2 : \tau$ hold.
- For all PCF contexts $\mathcal{C}$ for which $\mathcal{C}[M_1]$ and $\mathcal{C}[M_2]$ are closed terms of type γ, *where* $\gamma = nat$ *or* $\gamma = bool$, and for all values $V : \gamma$,

$$\mathcal{C}[M_1] \Downarrow_\gamma V \;\Rightarrow\; \mathcal{C}[M_2] \Downarrow_\gamma V.$$

Slide 62

Contextual preorder from formal approximation

Proposition. *For all PCF types τ and all closed terms* $M_1, M_2 \in \mathrm{PCF}_\tau$

$$M_1 \leq_{\mathrm{ctx}} M_2 : \tau \;\Leftrightarrow\; [\![M_1]\!] \lhd_\tau M_2.$$

Slide 63

Proof of the Proposition on Slide 63. It is not hard to prove that for closed terms $M_1, M_2 \in \mathrm{PCF}_\tau$, $M_1 \leq_{\mathrm{ctx}} M_2 : \tau$ holds if and only if for all $M \in \mathrm{PCF}_{\tau \to bool}$

$$M\,M_1 \Downarrow_{bool} \mathbf{true} \;\Rightarrow\; M\,M_2 \Downarrow_{bool} \mathbf{true}.$$

Now if $[\![M_1]\!] \lhd_\tau M_2$, then for any $M \in \mathrm{PCF}_{\tau \to bool}$ since by the Fundamental Property of $\lhd$ we have $[\![M]\!] \lhd_{\tau \to bool} M$, the definition of $\lhd_{\tau \to bool}$ implies that

$$[\![M\,M_1]\!] = [\![M]\!]([\![M_1]\!]) \lhd_{bool} M\,M_2. \tag{20}$$

So if $M\,M_1 \Downarrow_{bool} \mathbf{true}$, then $[\![M\,M_1]\!] = \mathit{true}$ (by the Soundness property) and hence by definition of $\lhd_{bool}$ from (20) we get $M\,M_2 \Downarrow_{bool} \mathbf{true}$. Thus using the characterisation of $\leq_{\mathrm{ctx}}$ mentioned above, we have $M_1 \leq_{\mathrm{ctx}} M_2 : \tau$.

This establishes the right-to-left implication on Slide 63. For the converse, it is enough to prove

$$(d \lhd_\tau M_1 \;\&\; M_1 \leq_{\mathrm{ctx}} M_2 : \tau) \;\Rightarrow\; d \lhd_\tau M_2. \tag{21}$$

For then if $M_1 \leq_{\mathrm{ctx}} M_2 : \tau$, since $[\![M_1]\!] \lhd_\tau M_1$ (by the Fundamental Property), (21) implies $[\![M_1]\!] \lhd_\tau M_2$. Property (21) follows by induction on the structure of the type τ, using the following easily verified properties of $\leq_{\mathrm{ctx}}$:

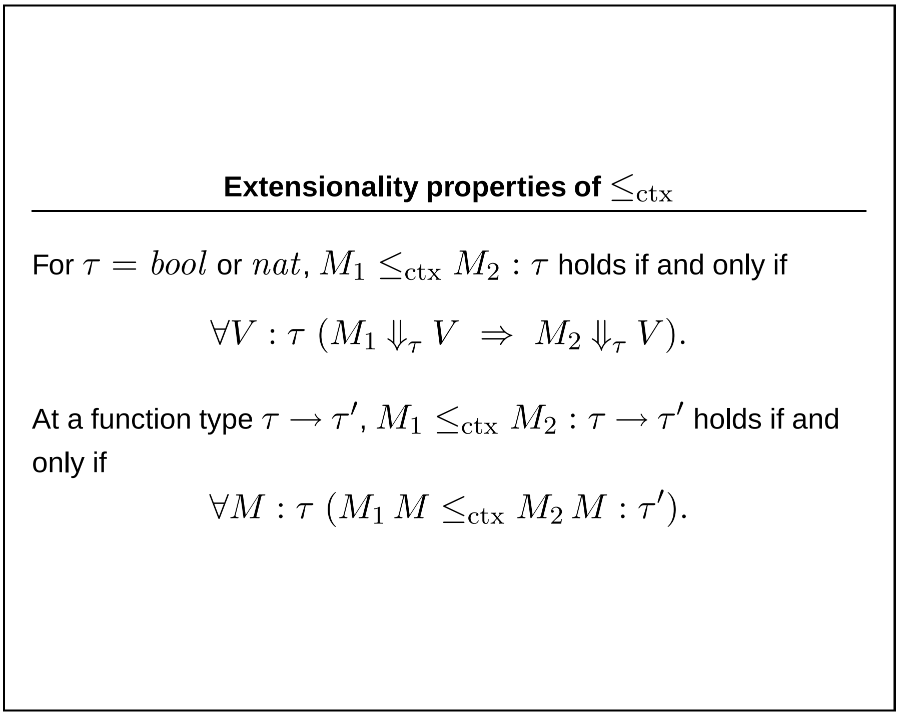
- If $\tau = nat$ or $bool$, then $M_1 \leq_{\mathrm{ctx}} M_2 : \tau$ implies

$$\forall V : \tau \, (M_1 \Downarrow_\tau V \;\Rightarrow\; M_2 \Downarrow_\tau V).$$

- If $M_1 \leq_{\mathrm{ctx}} M_2 : \tau \to \tau'$, then $M_1 \, M \leq_{\mathrm{ctx}} M_2 \, M : \tau'$, for all $M : \tau$.

□

The bi-implication on Slide 63 allows us to transfer the extensionality properties enjoyed by the domain partial orders $\sqsubseteq$ to the contextual preorder, as shown on Slide 64. (These kind of properties of PCF were first proved by Milner 1977, First Context Lemma, page 6.)



Extensionality properties of $\leq_{\mathrm{ctx}}$

For $\tau = bool$ or nat, $M_1 \leq_{\mathrm{ctx}} M_2 : \tau$ holds if and only if

$$\forall V : \tau \; (M_1 \Downarrow_\tau V \;\Rightarrow\; M_2 \Downarrow_\tau V).$$

At a function type $\tau \to \tau'$, $M_1 \leq_{\mathrm{ctx}} M_2 : \tau \to \tau'$ holds if and only if

$$\forall M : \tau \; (M_1 \, M \leq_{\mathrm{ctx}} M_2 \, M : \tau').$$

Slide 64

Proof of the properties on Slide 64. The ‘only if’ directions are easy consequences of the definition of $\leq_{\mathrm{ctx}}$.

For the ‘if’ direction in case $\tau = bool$ or nat, we have

$$\begin{aligned} [\![M_1]\!] = [\![V]\!] &\Rightarrow M_1 \Downarrow_\tau V && \text{by the adequacy property} \\ &\Rightarrow M_2 \Downarrow_\tau V && \text{by assumption} \end{aligned}$$

and hence $[\![M_1]\!] \lhd_\tau M_2$ by definition of $\lhd$ at these ground types. Now apply the Proposition on Slide 63.

For the ‘if’ direction in case of a function type $\tau \to \tau'$, we have

$$\begin{array}{rll} d \lhd_\tau M & \Rightarrow [\![M_1]\!](d) \lhd_{\tau'} M_1\, M & \text{since } [\![M_1]\!] \lhd_\tau M_1 \\ & \Rightarrow [\![M_1]\!](d) \lhd_{\tau'} M_2\, M & \text{by (21), since } M_1\, M \leq_{\text{ctx}} M_2\, M : \tau' \\ & & \text{by assumption} \end{array}$$

and hence $[\![M_1]\!] \lhd_{\tau\to\tau'} M_2$ by definition of $\lhd$ at type $\tau \to \tau'$. So once again we can apply the Proposition on Slide 63 to get the desired conclusion. □

7.4 Exercises

Exercise 7.4.1. For any PCF type τ and any closed terms $M_1, M_2 \in \mathrm{PCF}_\tau$, show that

(22) $$\forall V : \tau\ (M_1 \Downarrow_\tau V \ \Leftrightarrow\ M_2 \Downarrow_\tau V) \ \Rightarrow\ M_1 \cong_{\text{ctx}} M_2 : \tau.$$

[Hint: combine the Proposition on Slide 63 with Lemma 7.2.1(iii).]

Exercise 7.4.2. Use (22) to show that β-conversion is valid up to contextual equivalence in PCF, in the sense that for all $\mathbf{fn}\, x : \tau\, .\, M_1 \in \mathrm{PCF}_{\tau\to\tau'}$ and $M_2 \in \mathrm{PCF}_\tau$

$$(\mathbf{fn}\, x : \tau\, .\, M_1)\, M_2 \ \cong_{\text{ctx}}\ M_1[M_2/x] : \tau'.$$

Exercise 7.4.3. Is the converse of (22) valid at all types? [Hint: recall the extensionality property of $\leq_{\text{ctx}}$ at function types (Slide 64) and consider the terms $\mathbf{fix}(\mathbf{fn}\, f : (nat \to nat)\, .\, f)$ and $\mathbf{fn}\, x : nat\, .\, \mathbf{fix}(\mathbf{fn}\, x' : nat\, .\, x')$ of type $nat \to nat$.]

8 Full Abstraction

8.1 Failure of full abstraction

As we saw on Slide 44, the adequacy property implies that contextual equivalence of two PCF terms can be proved by showing that they have equal denotations: $[\![M_1]\!] = [\![M_2]\!] \in [\![\tau]\!] \quad \Rightarrow \quad M_1 \cong_{\mathrm{ctx}} M_2 : \tau$. Unfortunately the converse is false: *there are contextually equivalence PCF terms with unequal denotations.*

Proof principle

For all types τ and closed terms $M_1, M_2 \in \mathrm{PCF}_\tau$,

$$[\![M_1]\!] = [\![M_2]\!] \text{ in } [\![\tau]\!] \implies M_1 \cong_{\mathrm{ctx}} M_2 : \tau \ .$$

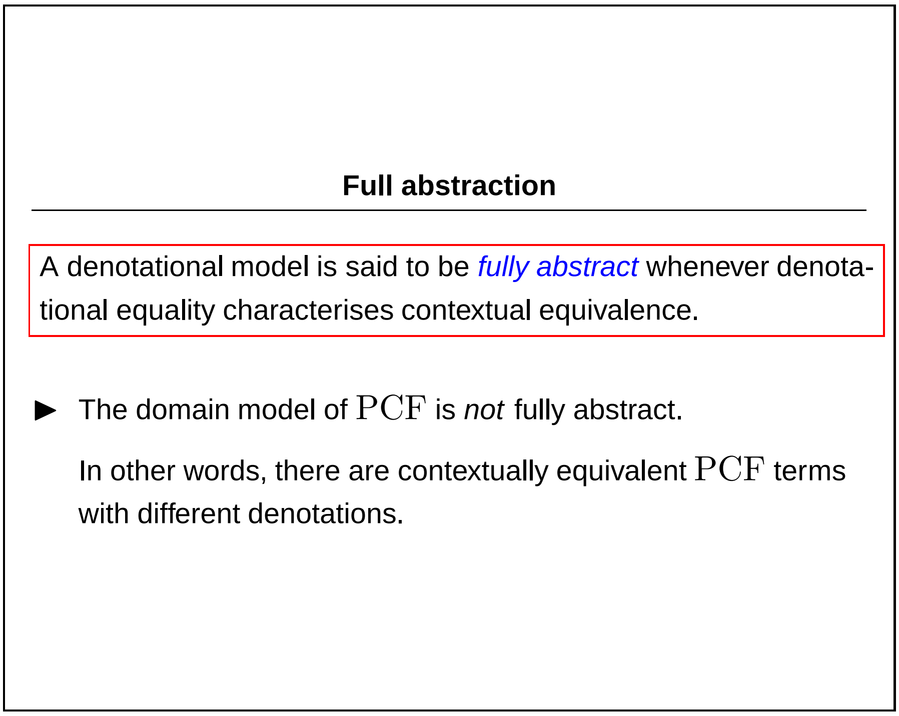
Hence, to prove

$$M_1 \cong_{\mathrm{ctx}} M_2 : \tau$$

it suffices to establish

$$[\![M_1]\!] = [\![M_2]\!] \text{ in } [\![\tau]\!] \ .$$

Slide 65



Full abstraction

A denotational model is said to be *fully abstract* whenever denotational equality characterises contextual equivalence.

- The domain model of PCF is *not* fully abstract.

  In other words, there are contextually equivalent PCF terms with different denotations.

Slide 66

In general one says that a denotational semantics is *fully abstract* if contextual equivalence coincides with equality of denotation. Thus the denotational semantics of PCF using domains and continuous functions fails to be fully abstract. The classic example demonstrating this failure is due to Plotkin (1977) and involves the *parallel-or* function shown on Slide 67.

Parallel-or function

is the continuous function $por : \mathbb{B}_\perp \to (\mathbb{B}_\perp \to \mathbb{B}_\perp)$ defined by

por	$true$	$false$	$\perp$
$true$	$true$	$true$	$true$
$false$	$true$	$false$	$\perp$
$\perp$	$true$	$\perp$	$\perp$

Slide 67

Contrast por with the 'sequential-or' function shown on Slide 68. Both functions give the usual boolean 'or' function when restricted to $\{true, false\}$, but differ in their behaviour at arguments involving the element $\perp$ denoting 'non-termination'. Note that $por(d_1, d_2) = true$ if *either* of d_1 or d_2 is $true$, even if the other argument is $\perp$; whereas $orelse(d_1, d_2) = true$ implies $d_1 \neq \perp$.

Left *sequential-or* function

The function $orelse : \mathbb{B}_\perp \to (\mathbb{B}_\perp \to \mathbb{B}_\perp)$ defined by

$orelse$	$true$	$false$	$\perp$
$true$	$true$	$true$	$true$
$false$	$true$	$false$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

is the denotation of the PCF term

$$\mathbf{fn}\, x : bool\,.\, \mathbf{fn}\, x' : bool\,.\, \mathbf{if}\ x\ \mathbf{then}\ \mathbf{true}\ \mathbf{else}\ x'$$

of type $bool \to (bool \to bool)$.

Slide 68

As noted on Slide 68, $orelse$ can be defined in PCF, in the sense that there is a closed PCF term $M : bool \to (bool \to bool)$ with $[\![M]\!] = orelse$. This term M tests whether its first argument is $\mathbf{true}$ or $\mathbf{false}$ (and so diverges if that first argument diverges), in the first case returning $\mathbf{true}$ (leaving the second argument untouched) and in the second case returning the second argument. By contrast, for por we have the Proposition stated on Slide 69. We will not give the proof of this proposition here. Plotkin (1977) proves it via an 'Activity Lemma', but there are alternative approaches using 'stable' continuous functions (Gunter 1992, p 181), or using 'sequential logical relations' (Sieber 1992). The key idea is that evaluation in PCF proceeds *sequentially*. So whatever P is, evaluation of $P\, M_1\, M_2$ must at some point involve full evaluation of either M_1 or M_2 (P cannot ignore its arguments if it is to return $\mathbf{true}$ in some cases and $\mathbf{false}$ in others); whereas an algorithm to compute por at a pair of arguments must compute the values of those arguments 'in parallel' in case one diverges whilst the other yields the value $true$.

One can exploit the undefinability of por in PCF to manufacture a pair of contextually equivalent closed terms in PCF with unequal denotations. Such a pair is given on Slide 70.

Undefinability of parallel-or

Proposition. *There is no closed PCF term*

$$P : \mathit{bool} \to (\mathit{bool} \to \mathit{bool})$$

satisfying

$$[\![P]\!] = \mathit{por} : \mathbb{B}_\perp \to (\mathbb{B}_\perp \to \mathbb{B}_\perp) \ .$$

Slide 69

Parallel-or test functions

For $i = 1, 2$ define

```
$T_i \overset{\text{def}}{=}$  $\mathbf{fn}\, f : \mathit{bool} \to (\mathit{bool} \to \mathit{bool})\,.$
      $\mathbf{if}\ (f\ \mathbf{true}\ \Omega)\ \mathbf{then}$
        $\mathbf{if}\ (f\ \Omega\ \mathbf{true})\ \mathbf{then}$
          $\mathbf{if}\ (f\ \mathbf{false}\ \mathbf{false})\ \mathbf{then}\ \Omega\ \mathbf{else}\ B_i$
        $\mathbf{else}\ \Omega$
      $\mathbf{else}\ \Omega$
```

where $B_1 \overset{\text{def}}{=} \mathbf{true}$, $B_2 \overset{\text{def}}{=} \mathbf{false}$,
and $\Omega \overset{\text{def}}{=} \mathbf{fix}(\mathbf{fn}\, x : \mathit{bool}\,.\, x)$.

Slide 70

Failure of full abstraction

Proposition.

$$T_1 \cong_{\mathrm{ctx}} T_2 : (\mathit{bool} \to (\mathit{bool} \to \mathit{bool})) \to \mathit{bool}$$

$$[\![T_1]\!] \neq [\![T_2]\!] \in (\mathbb{B}_\bot \to (\mathbb{B}_\bot \to \mathbb{B}_\bot)) \to \mathbb{B}_\bot$$

Slide 71

Proof of the Proposition on slide 71. From the definition of por on Slide 67 and the definition of $[\![-]\!]$ in Section 6.2, it is not hard to see that

$$[\![T_i]\!](por) = \begin{cases} \mathit{true} & \text{if } i = 1 \\ \mathit{false} & \text{if } i = 2. \end{cases}$$

Thus $[\![T_1]\!](por) \neq [\![T_2]\!](por)$ and therefore $[\![T_1]\!] \neq [\![T_2]\!]$.

To see that $T_1 \cong_{\mathrm{ctx}} T_2 : (\mathit{bool}\to(\mathit{bool}\to\mathit{bool}))\to\mathit{bool}$ we use the extensionality results on Slide 64. Thus we have to show for all $M : \mathit{bool} \to (\mathit{bool} \to \mathit{bool})$ and $V \in \{\mathbf{true}, \mathbf{false}\}$ that

$$T_1\, M \Downarrow_{\mathit{bool}} V \;\Leftrightarrow\; T_2\, M \Downarrow_{\mathit{bool}} V. \tag{23}$$

But the definition of T_i is such that $T_i\, M \Downarrow_{\mathit{bool}} V$ only holds if

$$M\,\mathbf{true}\,\Omega \Downarrow_{\mathit{bool}} \mathbf{true},\; M\,\Omega\,\mathbf{true} \Downarrow_{\mathit{bool}} \mathbf{true},\; M\,\mathbf{false}\,\mathbf{false} \Downarrow_{\mathit{bool}} \mathbf{false}.$$

By the soundness property of Slide 43 this means that

$$[\![M]\!](\mathit{true})(\bot) = \mathit{true},\; [\![M]\!](\bot)(\mathit{true}) = \mathit{true},\; [\![M]\!](\mathit{false})(\mathit{false}) = \mathit{false}.$$

(Recall from Exercise 6.5.2 that $[\![\Omega]\!] = \bot$.) It follows in that case that the continuous function $[\![M]\!] : (\mathbb{B}_\bot \times \mathbb{B}_\bot) \to \mathbb{B}_\bot$ coincides with por (see Exercise 8.4.1). But this is impossible, by the Proposition on Slide 69. Therefore (23) is trivially satisfied for all M, and thus T_1 and T_2 are indeed contextually equivalent. □

8.2 PCF+por

The failure of full abstraction for the denotational semantics of PCF can be repaired by extending PCF with extra terms for those elements of the domain-theoretic model that are not definable in the language as originally given. We have seen that por is one such element 'missing' from PCF, and one of the remarkable results in (Plotkin 1977) is that this is the only thing we need add to PCF to obtain full abstraction. This is stated without proof on Slides 72 and 73.

PCF+por

Expressions $\quad M ::= \cdots \mid \mathbf{por}(M, M)$

Typing $\quad \dfrac{\Gamma \vdash M_1 : \mathit{bool} \quad \Gamma \vdash M_2 : \mathit{bool}}{\Gamma \vdash \mathbf{por}(M_1, M_2) : \mathit{bool}}$

Evaluation

$$\frac{M_1 \Downarrow_{bool} \mathbf{true}}{\mathbf{por}(M_1, M_2) \Downarrow_{bool} \mathbf{true}} \qquad \frac{M_2 \Downarrow_{bool} \mathbf{true}}{\mathbf{por}(M_1, M_2) \Downarrow_{bool} \mathbf{true}}$$

$$\frac{M_1 \Downarrow_{bool} \mathbf{false} \quad M_2 \Downarrow_{bool} \mathbf{false}}{\mathbf{por}(M_1, M_2) \Downarrow_{bool} \mathbf{false}}$$

Slide 72

Plotkin's full abstraction result

The denotational semantics of PCF+por terms is given by extending the definition on Slides 50–54 with the clause

$$\begin{aligned}\llbracket \Gamma \vdash \mathbf{por}(M_1, M_2) \rrbracket(\rho) &\stackrel{\text{def}}{=} \\ &\quad por(\llbracket \Gamma \vdash M_1 \rrbracket(\rho))(\llbracket \Gamma \vdash M_2 \rrbracket(\rho))\end{aligned}$$

where $por : \mathbb{B}_\perp \to (\mathbb{B}_\perp \to \mathbb{B}_\perp)$ is as on Slide 67.

This denotational semantics is fully abstract for contextual equivalence of PCF+por terms:

$$\Gamma \vdash M_1 \cong_{\text{ctx}} M_2 : \tau \;\Leftrightarrow\; \llbracket \Gamma \vdash M_1 \rrbracket = \llbracket \Gamma \vdash M_2 \rrbracket.$$

Slide 73

8.3 Fully abstract semantics for PCF

The evaluation of PCF terms involves a form of 'sequentiality' which is not reflected in the denotational semantics of PCF using domains and continuous functions: the continuous function por does not denote any PCF term and this results in a mismatch between denotational equality and contextual equivalence. But what precisely does 'sequentiality' mean in general? Can we characterise it in an abstract way, independent of the particular syntax of PCF terms, and hence give a more refined form of denotational semantics that *is* fully abstract for contextual equivalence for PCF (and for other types of language besides the simple, pure functional language PCF)? These questions have motivated the development much domain theory and denotational semantics since the appearance of (Plotkin 1977): see the survey by Ong (1995), for example.

It is only recently that definitive answers have emerged even for such an apparently simple language as PCF. O'Hearn and Riecke (1995) construct a fully abstract model of PCF by using certain kinds of 'logical relation' to repair the deficiencies of the standard model we have described here. Although this does provide a solution, it does not seem to give much insight into the nature of sequential computation. By contrast, Abramsky, Jagadeesan, and Malacaria (2000) and Hyland and Ong (2000) solve the problem by introducing what appears to be a radically different approach to giving semantics to programming languages (not just PCF), based upon certain kinds of two-player game: see (Abramsky 1997) and (Hyland 1997) for introductions to this 'game semantics'.

Finally, a negative result by Loader should be mentioned. Note that the material in Section 8.1 does not depend upon the presence of numbers and arithmetic in PCF. Let PCF$_2$ denote the fragment of PCF only involving *bool* and the function types formed from it, $\mathbf{true}$, $\mathbf{false}$, conditionals, variables, function abstraction and application, and a divergent term $\Omega_{bool} : bool$. Since $\mathbb{B}_\perp$ is a finite domain and since the function domain formed from finite domains is again finite, the domain associated to each PCF$_2$ type is finite.[1] The domain model is adequate for PCF$_2$ and hence there are only finitely many different PCF$_2$ terms of each type, up to contextual equivalence. Given these finiteness properties, and the terribly simple nature of the language, one might hope that the following questions are decidable (uniformly in the PCF$_2$ type τ):

- Which elements of $[\![\tau]\!]$ are definable by PCF$_2$ terms?
- When are two PCF$_2$ of type τ contextually equivalent?

Quite remarkably Loader (2001) shows that these are recursively undecidable questions.

8.4 Exercises

Exercise 8.4.1. Suppose that a monotonic function $p : (\mathbb{B}_\perp \times \mathbb{B}_\perp) \to \mathbb{B}_\perp$ satisfies

$$p(true, \perp) = true, \quad p(\perp, true) = true, \quad \text{and} \quad p(false, false) = false.$$

Show that p coincides with the parallel-or function on Slide 67 in the sense that $p(d_1, d_2) = por(d_1)(d_2)$, for all $d_1, d_2 \in \mathbb{B}_\perp$.

Exercise 8.4.2. Show that even though there are two evaluation rules on Slide 72 with conclusion $\mathbf{por}(M_1, M_2) \Downarrow_{bool} \mathbf{true}$, nevertheless the evaluation relation for PCF+por is still deterministic (in the sense of Proposition 5.4.1).

Exercise 8.4.3. Give the axioms and rules for an inductively defined transition relation for PCF+por. This should take the form of a binary relation $M \to M'$ between closed PCF+por terms. It should satisfy

$$M \Downarrow V \;\Leftrightarrow\; M \to^* V$$

(where $\to^*$ is the reflexive-transitive closure of $\to$).

[1] A further simplification arises from the fact that if the domains D and D' are finite, then they contain no non-trivial chains and hence the continuous functions $D \to D'$ are just the monotone functions.

References

Abramsky, S. (1997). Semantics of interaction: An introduction to game semantics. In A. M. Pitts and P. Dybjer (Eds.), *Semantics and Logics of Computation*, Publications of the Newton Institute, pp. 1–31. Cambridge University Press.

Abramsky, S., R. Jagadeesan, and P. Malacaria (2000). Full abstraction for PCF. *Information and Computation 163*, 409–470.

Fiore, M., A. Jung, E. Moggi, P. O'Hearn, J. Riecke, G. Rosolini, and I. Stark (1996). Domains and denotational semantics: History, accomplishments and open problems. *Bulletin of EATCS, 59*, 227–256.

Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press.

Hyland, J. M. E. (1997). Game semantics. In A. M. Pitts and P. Dybjer (Eds.), *Semantics and Logics of Computation*, Publications of the Newton Institute, pp. 131–184. Cambridge University Press.

Hyland, J. M. E. and C.-H. L. Ong (2000). On full abstraction for PCF. *Information and Computation 163*, 285–408.

Loader, R. (2001). Finitary PCF is not decidable. *Theoretical Computer Science 266*, 341–364.

Milner, R. (1977). Fully abstract models of typed lambda-calculi. *Theoretical Computer Science 4*, 1–22.

Mulmuley, K. (1987). *Full Abstraction and Semantic Equivalence*. MIT Press.

O'Hearn, P. W. and J. G. Riecke (1995). Kripke logical relations and PCF. *Information and Computation 120*, 107–116.

Ong, C.-H. L. (1995). Correspondence between operational and denotational semantics. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum (Eds.), *Handbook of Logic in Computer Science, Vol 4*, pp. 269–356. Oxford University Press.

Paulson, L. C. (1987). *Logic and Computation*. Cambridge University Press.

Pitts, A. M. (1996). Relational properties of domains. *Information and Computation 127*, 66–90.

Plotkin, G. D. (1977). LCF considered as a programming language. *Theoretical Computer Science 5*, 223–255.

Scott, D. S. (1993). A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science 121*, 411–440.

Sieber, K. (1992). Reasoning about sequential functions via logical relations. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts (Eds.), *Applications of Categories in Computer Science, Proceedings LMS Symposium, Durham, UK,* *1991*, Volume 177 of *LMS Lecture Note Series*, pp. 258–269. Cambridge University Press.

Tennent, R. D. (1991). *Semantics of Programming Languages*. Prentice Hall International (UK) Ltd.

Winskel, G. (1993). *The Formal Semantics of Programming Languages*. Foundations of Computing. Cambridge, Massachusetts: The MIT Press.