Preface

I wrote these notes between about 1984 and 1989. Not too long before I had been lucky enough to attend Joe Stoy's lectures on domain theory and denotational semantics in 1977/8 (my final undergraduate year) and to be a graduate student in Oxford as Dana Scott developed the notion of an *information system* through two later iterations of the first half of the same course. I vividly remember how the second version, on which the present notes are fairly closely based, were a blessed relief to one who had taught people the first version!

I'd like to express my thanks to both Joe and Dana for the education they gave me.

In my turn I gave the domain theory course in Oxford for many years and I wrote these notes to accompany that course, and they were used until quite recently by subsequent lecturers. The version of information systems in these notes was simplified from Dana's version by the removal of a null token and the change from inferences of the form $X \vdash Y$ to ones of the form $X \vdash a$, where a is a single token and Y is a set of them.

What appealed to me about information systems was the way they turned an abstract axiomatisation of how propositions might describe an object into a fairly concrete representation of domain theory. There is not much category theory in these notes, and what there is is largely camouflaged. This was because category theory did not seem to me to be necessary to achieve what I wanted, and I did not want to add a further level of abstraction to what could be taught in an 8-week course at two lectures a week. (Oxford final-year undergraduates came to the course without any background in categories.) In that sense at least these notes now seem quite out of date, and it would be wrong to try to give the impression that they give a reasonable introduction to the subject of domain theory as it is now generally understood.

My course would consist of these notes with the exceptions of Section 2.4 (advanced topics in partial orders), Chapter 6 (powerdomains) and sometimes Section 5.2.

I have wanted to put these notes on line for a number of years, but was

held back until now by the way my 1980's LaTeX had grown incompatible with modern versions and font selections. I'm very grateful to my secretary Sue Taylor for revising the LaTeX as well as tidying them up in various ways.

Bill Roscoe
November 2007

Notes on Domain Theory

Bill Roscoe
Oxford University Computing Laboratory

January 18, 2007

Chapter 0

Introduction

I wrote these notes in the 1980s for a course on domain theory that I gave at Oxford. They were used (by me and others) as the text for that case for many years after. They were originally intended to form half of a co-authored book on domain theory and semantics, but my co-author never finished his half.

I had done a course on Scott-Strachey semantics in 1977/8 as a final year undergraduate. That course was given by Joe Stoy, based on his (then new) book. The domain theory used there was the $\mathbf{P}\omega$ model in which domains were sublattices of the powerset of the natural numbers.

While that theory provided excellent connections with recursion theory, it had the disadvantage that the codings of domains within the natural numbers were a bit artificial, and that all domains had tops. By about 1980, Dana Scott was developing the concept of an "information system" in successors to the course I had taken and which I now found myself helping with the teaching for. After a couple of years his development culminated in the paper *Domains for denotational semantics* (ICALP 1982).

I slightly reworked the structure of an information system for these notes, but these remain very close to what Scott had proposed. They were my attempt to present a course on domain theory appropriate for final-year Oxford undergraduates based on Scott's ideas. The undergraduate course never covered the chapter on powerdomains: essentially this was written for my own entertainment.

Information systems are based on concrete representations of domains: there is no explicit mention of category theory here at all. Obviously the topic of domain theory has moved on a great deal since I wrote these notes: they should be thought of as a product of their era, a kind of valedictory conclusion to Scott's time in Oxford.

Bill Roscoe

Oxford, January 2007.

Chapter 1

The motivation for domain theory

Computer programs, programming languages and the machines we run them on are among the most complex creations of mankind. In everyday life we often need to rely on them working correctly. There are therefore very good reasons why we should develop methods of understanding these objects, constructing them properly, and proving them correct. We would like to be able to use mathematics to reason about computer programs, just as engineers use mathematical methods to reason about their structures.

We need to develop the right sort of mathematics to do this. A mathematical model should reflect the detail we are interested in: at each level we should be able to manipulate the concepts that are important without being cluttered by irrelevant or misleading information. In other words it should be at the right level of abstraction. For example, let us consider what is required of a mathematical model for programs in a language which includes variables and assignment.

- The model should almost certainly keep some record of the final values of a program's free variables. This is because we are likely to be able to use these values outside the program, for example by sequentially composing our program with another one that uses its variables.
- On the other hand we are very unlikely at this level to want to know about the electrical impulses which make up the computation. Not only would such details clutter up the model but they are inappropriate: provided the language implementation we are using is correct we should not have to worry about precisely how the implementation works, or even what sort of machine it is running on.
- Many things are not so clear cut. Consider the *intermediate* values of a program's free variables. In most languages there is no reason to want to

know about these, because there is no way of inspecting a program's variables while it is running. However, imagine a language where it is possible to compose two subprograms in *parallel*, so that they run simultaneously. Now, if one parallel program has the power to inspect the other's variables at any time, it is obviously necessary to keep track of *all* the values of the latter's variables, not only the final ones. In such a language, if we run $y := x$ in parallel with $x := 3$ then the final value of y is either 3 or the initial value of x. If, on the other hand, we run $y := x$ in parallel with $x := 1;\ x := 3$ then we have the additional possibility that y might have final value 1. Thus the programs $x := 3$ and $x := 1;\ x := 3$ are not equivalent in this language, though they would be in most.

It seems likely that we will need a good deal of flexibility in our theory: for any particular system or language we must have the ability to choose the features that are important there, and build a model for them.

Indeed, it seems only natural that once we have developed a reasonable mathematical theory of programming languages, we should use it to provide a standard, or specification, by which to judge particular implementations. Therefore it is important that we should develop a theory which allows us to capture the 'essence' of languages and the programs written in them.

There are several approaches to the mathematics of programming languages: some are based on associating logical formulae with programs; others are based on developing an algebra of programs. The approach described in this book is that of Scott and Strachey: given a language $\mathbf{L}$ we will attempt to construct an appropriate mathematical model $\mathcal{M}$ and an *abstraction* or *semantic* function $\mathcal{S} : \mathbf{L} \to \mathcal{M}$. For each program $P \in \mathbf{L}$, SSP should represent the 'meaning' of P. This is known as *denotational* semantics; for a summary of other approaches see Chapter ??.

Our aim will be to construct models and semantic functions in such a way that the aims set out above are achieved. The function $\mathcal{S}$ should be such that $SSP = SSQ$ if and only if P is naturally equivalent to Q. We will also hope to be able to prove the correctness of a program by inspecting the value it is mapped to in M.

In Sections 1.1 and 1.2 we look at two very different programming languages. On the one hand this serves to introduce the notation and basic ideas of programming language semantics. On the other, we quickly meet the problem which much of this book is devoted to solving: the fact that it is by no means obvious that the mathematical models we need even exist.

1.1 Some languages are easy

Whenever we are presenting a language we will need to describe its syntax. We will do this by describing a number of *syntactic domains*. These will be the various classes of syntactic objects from which programs are constructed.

Consider a very simple language in which there are variables, expressions that can be assigned to them, and in which the only way of building large programs is by running two smaller ones in sequence. The expressions include numerals representing the natural numbers, variables and the sum and product of all pairs of expressions. We will use the following syntactic domains to represent this language.

SYNTACTIC DOMAINS

I	$\in$	Var	variables
N	$\in$	Num	numerals
E	$\in$	Exp	expressions
Γ	$\in$	Cmd	commands

Here we are giving names to a number of classes of syntactic objects, and giving a notation for the typical element of each one. Thus, while we are considering this language, x will always represent a variable, E, E' will be expressions, and so on. Of course, giving names to the syntactic classes does not determine the form of the objects they contain. Let us assume that we already know the syntax of Var and Num. The syntax of expressions can be described by the formula:

$$E ::= N \mid I \mid E_1 + E_2 \mid E_1 * E_2 \quad .$$

This simply says that an expression is always either a numeral, or a variable, or the sum of two smaller expressions, or the product of two smaller expressions. We formally define the set of all expressions to be the smallest set satisfying the above 'equation': in other words the smallest set which, whenever it contains E_1, E_2 and E, also contains all the objects on the right hand side above. This very convenient way of describing syntax is known as *Backus-Naur Form* or BNF for short[1]. The

[1]The above actually varies a little from standard BNF, because we have labelled the subcomponents: for example E_1 and E_2, imagining these to stand for particular expressions. The standard method would use the unlabelled name E, with no implication that different uses of it represent the same expression. Provided one keeps to the rule that no label is used more than once in any term (and we will keep to it), ours is equivalent to the standard form, and perhaps a little clearer. By using a label more than once – with the obvious meaning that expressions with the same label are the same – one can describe syntaxes not obtainable from standard BNF (and harder to parse and store). A good example is *palindromes* (sequences that read the same backwards and forwards), which are generated by $PI ::= B \mid B_1.B_1 \mid B_1.PI.B_1$, where B ranges (say) over the letters of the alphabet.

syntax of commands can be similarly described:

$$\Gamma ::= \ I := E \ \mid \ \Gamma_1; \ \Gamma_2 \quad .$$

An important assumption in describing a BNF syntax is that the various clauses contained in it are disjoint: for example no object can both be of the form $E_1 + E_2$ *and* of the form $E_1 * E_2$. In other words, given an object in one of these syntaxes, we know exactly how it is put together, or *parsed*. They can be termed objects in an *abstract syntax*, and in some ways it is more accurate to think of them as parse trees than as strings of symbols, which may be termed *concrete syntax*. (Consider, for example, the expression $1 + 2 * 3$, which can either be read as $1 + (2 * 3)$ or as $(1 + 2) * 3$.) It is the duty of anyone writing down syntactic objects as strings of symbols to introduce enough bracketing (even when this is not a formal part of the syntax) or parsing conventions to make their abstract syntax unambiguous.

Having completed the description of our language's syntax, we can now start thinking about its *semantics*. To do this we must decide on appropriate mathematical models for the various syntactic classes.

We will assume the existence of a semantic function $\mathcal{N} : \mathit{Num} \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers (non-negative integers). For each $N \in \mathit{Num}$, $\mathcal{N}[\![N]\!]$ is the number that N denotes. We will always use the 'semantic brackets' $[\![\cdot]\!]$ around a piece of syntax to which we are applying a semantic function.

The syntactic domain *Var* is, so far as we are concerned, just a set of distinct names. Any sensible semantic model for it would also just be a set of one distinct object for each – so it is simplest to use *Var* 'as its own model' – in other words to use variables directly in the semantics. We do, however, need a mechanism for recording the values stored in the variables: recalling that all expressions are to take values in $\mathbb{N}$, we define S, the set of *states* to be $\mathit{Var} \to \mathbb{N}$, the set of functions from variables to their possible values. Given the variable[2] x and state σ, $\sigma[\![x]\!]$ is the value of x in σ.

Clearly the value of a general expression can depend on the values of any variables it contains. Therefore the semantic function for expressions has the form

$$\mathcal{E} : \mathit{Exp} \to (S \to \mathbb{N}) \quad .$$

In other words $\mathcal{E}[\![E]\!]$ is a function from states to natural numbers; $\mathcal{E}[\![E]\!]\sigma$ is the

[2]Notice that we are using 'x' as a typical variable rather than I. The distinction between these two ways of writing down the same thing is that one, I, is a *syntactic metavariable* representing any variable, whereas x is one specific variable, whose identity is the name 'x'. Thus, I_1 and I_2 are sometimes equal and sometimes not (depending on whether the variables they denote are the same; while x and y are always different. The reason why this is a little confusing is that the set *Var* contains only one class of objects: names. In fact x is to I what $x := 3; \ y := x$ is to Γ.

value of the expression E in the state σ.

The value of $\mathcal{E}[\![E]\!]$ can be defined for every expression E by giving one clause for each case in the BNF syntax of *Exp*. In each case, the value of the expression will be computed from those of the simpler ones from which it is built, a principle which we will discuss shortly, and thus:

$$\begin{aligned}
\mathcal{E}[\![N]\!]\sigma &= \mathcal{N}[\![N]\!] \\
\mathcal{E}[\![I]\!]\sigma &= \sigma[\![I]\!] \\
\mathcal{E}[\![E_1 + E_2]\!]\sigma &= \mathcal{E}[\![E_1]\!]\sigma + \mathcal{E}[\![E_2]\!]\sigma \\
\mathcal{E}[\![E_1 * E_2]\!]\sigma &= \mathcal{E}[\![E_1]\!]\sigma \times \mathcal{E}[\![E_2]\!]\sigma
\end{aligned}$$

Note that in each case we have defined the function $\mathcal{E}[\![E]\!]$ by giving its value at every element σ of its domain S. This style of definition depends on the principle of *extensionality* which states that two functions on a given domain X are equal if and only if they give the same result for every element of X. We will appeal to this principle many times in this book.

The definitions for $E_1 + E_2$ and $E_1 * E_2$ may appear trivial or circular, but it is important to remember that the 'operators' on the left hand side above are simply symbols, or syntactic objects, whereas + and × on the right hand side are just the usual operators over $\mathbb{N}$.

It is intuitively clear that, by giving the four clauses above, we have defined the function $\mathcal{E}$ for all $E \in \mathit{Exp}$. For example we can work out the value of $E = (\underline{1} + \underline{1}) * x$ in the state σ by first working out $\mathcal{E}[\![\underline{1}]\!]\sigma = \mathcal{N}[\![\underline{1}]\!]$ (= 1, presumably). (Numerals, such as $\underline{1}$, are underlined to distinguish them from numbers, such as 1.) We can then deduce that $\mathcal{E}[\![\underline{1} + \underline{1}]\!]\sigma = \mathcal{E}[\![\underline{1}]\!]\sigma + \mathcal{E}[\![\underline{1}]\!]\sigma = 1 + 1 = 2$, and finally obtain $\mathcal{E}[\![E]\!]\sigma = 2 \times \mathcal{E}[\![x]\!]\sigma = 2 \times \sigma[\![x]\!]$. Shortly we will see a proof that $\mathcal{E}$ is indeed defined on all E.

We can give a semantics for the commands in our language into the model $S \to S$: a command represents a function which maps initial to final states. The semantic function thus has the form

$$\mathcal{C} : \mathit{Cmd} \to (S \to S) \quad .$$

It is defined by the following clauses:

$$\begin{aligned}
\mathcal{C}[\![I := E]\!]\sigma &= \sigma[\mathcal{E}[\![E]\!]\sigma / I] \\
\mathcal{C}[\![\Gamma_1;\ \Gamma_2]\!]\sigma &= \mathcal{C}[\![\Gamma_2]\!](\mathcal{C}[\![\Gamma_1]\!]\sigma) \,.
\end{aligned}$$

$\sigma[n/x]$ represents the state which is the same as σ except that it maps x to n:

$$\sigma[n/x][\![x']\!] = \begin{cases} n & \text{if} \quad x = x' \\ \sigma[\![x']\!] & \text{if} \quad x \neq x' \end{cases} \quad .$$

The effect of the command $I := E$ is simply to evaluate the expression E and store the resulting natural number in I. The effect of the command $\Gamma_1;\ \Gamma_2$ on the state σ is the same as that of the command Γ_2 on the final state produced when Γ_1 is given σ as its initial state.

Now that we have defined the semantics of our language it is possible to prove theorems about it. For example, for all commands Γ_1, Γ_2 and Γ_3 we have $\mathcal{C}[\![\Gamma_1;\ (\Gamma_2;\ \Gamma_3)]\!] = \mathcal{C}[\![(\Gamma_1;\ \Gamma_2);\ \Gamma_3]\!]$ because, for all $\sigma \in S$,

$$\begin{aligned}
\mathcal{C}[\![\Gamma_1;\ (\Gamma_2;\ \Gamma_3)]\!]\sigma &= \mathcal{C}[\![\Gamma_2;\ \Gamma_3]\!](\mathcal{C}[\![\Gamma_1]\!]\sigma) \\
&= \mathcal{C}[\![\Gamma_3]\!](\mathcal{C}[\![\Gamma_2]\!](\mathcal{C}[\![\Gamma_1]\!]\sigma)) \\
&= \mathcal{C}[\![\Gamma_3]\!](\mathcal{C}[\![\Gamma_1;\ \Gamma_2]\!]\sigma) \\
&= \mathcal{C}[\![(\Gamma_1;\ \Gamma_2);\ \Gamma_3]\!]\sigma
\end{aligned}$$

Each line of this derivation is justified by the semantics of sequential composition, in one direction or the other.

The functions $\mathcal{C}[\![\Gamma_1;\ (\Gamma_2;\ \Gamma_3)]\!]$ and $\mathcal{C}[\![(\Gamma_1;\ \Gamma_2);\ \Gamma_3]\!]$ are thus equal, because they are equal on all arguments. (There is another rather more elegant proof of this result: since sequential composition simply has the effect of composing the values of its arguments as functions from S to S (i.e., $\mathcal{C}[\![\Gamma_1;\ \Gamma_2]\!] = \mathcal{C}[\![\Gamma_2]\!] \circ \mathcal{C}[\![\Gamma_1]\!]$), the result follows from the associativity of functional composition. However the proof given above illustrates a methodical style that is widely applicable in denotational semantics.) Other properties of this language are included as exercises at the end of the section.

The above semantics is adequate for our language because it tells us everything we want to know about the effects of the programs we can write in it (assuming that we are not interested in, for example, the intermediate values of variables). It also has the important property that the value of a compound program (such as $\Gamma_1;\ \Gamma_2$) or expression can be determined from the values of its immediate syntactic subcomponents. Suppose that the only initial state that we will ever want to give a program is σ_1, which maps all variables to 1. We might then attempt to define a simpler semantic function than $\mathcal{C}$ (which apparently is now too general): $\mathcal{C}'[\![P]\!]$ gives the final state of P when started in state σ_1. Even though $\mathcal{C}'$ tells us everything we want to know about programs, it is not a reasonable semantic function: in general it is impossible to tell the value of $\Gamma_1;\ \Gamma_2$ from those of Γ_1 and Γ_2, because Γ_1 is likely not to have σ_1 as its final state.

The term *denotational semantics* is usually reserved for situations where, in addition to the fact that programs are mapped to values in mathematical models, the value of a compound semantic object can be deduced from the those of its parts in the manner discussed above.

Let us now turn to the subject of proving things about syntactic objects. We asserted above that semantic functions such as $\mathcal{E}$ are defined on the whole of

their domain if one gives a clause for each case of the syntax, defining the value of that case in terms of its subcomponents. The following principle is very useful in proving this and other, less trivial, properties of objects in a BNF syntax.

PRINCIPLE OF STRUCTURAL INDUCTION
Suppose the property $\mathcal{R}$ is such that it can be proved for every object in a syntax on the assumption that it holds of all of that object's subcomponents. Then $\mathcal{R}$ holds of every object in the syntax.

The validity of this principle in BNF syntaxes is proved by the observation that, under its assumptions, the set of all objects in the syntax satisfying property $\mathcal{R}$ must satisfy the 'defining equation' of the syntax (because whenever all of an object's subcomponents are in the set, so is the object). But the syntactic domain is defined to be the *smallest* set satisfying its defining equation[3].

Our assertion about the definedness of semantic functions follows easily from this principle: $\mathcal{R}$ becomes, for example, '$\mathcal{E}$ is defined'. We already have, by construction, that $\mathcal{E}$ is defined on $E \in \mathit{Exp}$ whenever it is defined on E's subcomponents.

Readers familiar with mathematical logic will probably recognise structural induction as a technique they have used there. We will see it used to prove a wide variety of results about programs. The exercises at the end of this section include several applications.

The notation of functions

The reader will no doubt already have noticed that, in denotational semantics, we make frequent use of functions and function spaces. It is useful to develop a few conventions to make our mathematics easier to read.

Firstly we will usually *Curry* our functions. A Curried function is a function of several arguments which is modified to take its various arguments one at a time. This technique is named after H.B. Curry, one of the architects of the λ-calculus (see Chapter 7). For example, we can Curry the binary function '+' to obtain $\mathit{plus} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$; so that $(\mathit{plus}(3))(4) = 7$, and $\mathit{plus}(3)$ is the function that adds 3 to any natural number. Notice that $\mathcal{E}$ was of type $\mathit{Exp} \to (S \to \mathbb{N})$ rather than $(\mathit{Exp} \times S) \to \mathbb{N}$. Thus instead of $\mathcal{E}$ demanding both an expression and a state at the same time, if we wish we can feed it *only* an expression. Our reason for doing this is that 'intermediate' results (such as $\mathcal{E}[\![E]\!]$) are often interesting and/or useful. Note that in Currying functions we are taking advantage of the following general

[3]This demonstration of the validity of structural induction is highly reminiscent of the justification of ordinary mathematical induction from Peano's axioms. The last of these states that the natural numbers is the smallest set satisfying the preceding ones, and it is this which justifies induction.

isomorphism[4] between function spaces.

$$(A_1 \times A_2 \times \cdots \times A_n) \to B \quad \cong \quad A_1 \to (A_2 \to \cdots (A_n \to B) \ldots)$$

Notice that we can Curry a function of 2 arguments in two ways depending on which argument is taken first: for example the arithmetic subtraction operator $-$ could be Curried as

$$from\, n\, m = n - m \qquad \text{or} \qquad take\, n\, m = m - n\ .$$

So $from(10)$ is the function that subtracts its argument from 10, while $take(10)$ takes 10 from its argument. For a function of k arguments there are, similarly, $k!$ ways of Currying. If you are going to be interested in intermediate results it is vitally important to choose the correct order. For example it would have been wrong to Curry the semantic function $\mathcal{C}$ above as $\mathcal{C}\sigma[\![\Gamma]\!]$, since this would not allow us conveniently to write down the semantic value of a command.

There are two notational consequences of the habit of Currying functions. We often have to define iterated function spaces such as the one on the right above, and we often have to supply a function with several arguments, one at a time, as in $(((f(a))(b))(c))(d)$. To avoid cluttering up expressions with brackets, we adopt two conventions. The *function space* constructor will always associate to the *right*, so that the space on the right hand above can be rewritten:

$$A_1 \to A_2 \to \cdots A_n \to B\quad .$$

Functional *application* will associate to the *left*. We will also generally omit meaningless brackets around the arguments to functions. Thus we can rewrite $(((f(a))(b))(c))(d)$ as $f\ a\ b\ c\ d$. We can, of course, override these conventions by the use of brackets, as in

$$\begin{array}{rcl} A \to (B \to C) \to D & = & A \to ((B \to C) \to D) \\ f(g\,x)y & = & (f(g(x)))(y)\,. \end{array}$$

Notice that in writing such things as $\mathcal{E}[\![E]\!]\sigma$ we have already implicitly followed the convention concerning application.

Exercises

[4] Readers might be more familiar with one of the consequences of this isomorphism, which is that $a^{b \cdot c} = (a^b)^c$ for natural numbers a, b and c. For these are respectively the sizes of the sets $(B \times C) \to A$ and $C \to (B \to A)$ respectively, where A has size a and so on.

1.1.1 The syntax of binary numerals is as follows:

$N \in Num$ the set of binary numerals

$N ::= \underline{0} \mid \underline{1} \mid N\underline{0} \mid N\underline{1}$

Typical binary numerals are $\underline{001}$, $\underline{1}$, $\underline{100}$ and $\underline{00}$.
Define the semantic function $\mathcal{N} : Num \to \mathbb{N}$ that maps each binary numeral to the expected natural number. (Your definition should be by recursion on syntax, in the style illustrated in the definitions of $\mathcal{E}$ and $\mathcal{C}$ above.)

1.1.2 A 'non-standard' semantic function for Num is given by $\mathcal{Z} : Num \to \mathbb{Z}$. ($\mathbb{Z}$ is the set of all integers: positive, negative and zero.)

$$\begin{aligned} \mathcal{Z}[\![\underline{0}]\!] &= 0 \\ \mathcal{Z}[\![\underline{1}]\!] &= 1 \\ \mathcal{Z}[\![N\underline{0}]\!] &= -2 \times \mathcal{Z}[\![N]\!] \\ \mathcal{Z}[\![N\underline{1}]\!] &= 1 - (2 \times \mathcal{Z}[\![N]\!]) \end{aligned}$$

(a) Prove by structural induction on Num that, if $N = b_n b_{n-1} \ldots b_1 b_0$ is any numeral (where each b_i is $\underline{0}$ or $\underline{1}$), then

$$\mathcal{Z}[\![N]\!] = \textstyle\sum_{i=0}^{n} (-2)^i \hat{b_i}$$

where $\hat{\underline{0}} = 0$ and $\hat{\underline{1}} = 1$.

(b) Show that the function $\mathcal{Z}$ is onto.
[Hint: show by induction that the numerals of length n or less are mapped onto a contiguous set of exactly 2^n integers.]

(c) Show that $\mathcal{Z}[\![N]\!] = \mathcal{Z}[\![N']\!]$ if and only if *either* N can be obtained from N' by adding leading zeros *or vice-versa*.

1.1.3 Use the principle of extensionality to prove the following results about substitution in states $\sigma \in S$:

(a) If $x \neq y$ then $\sigma[a/x][b/y] = \sigma[b/y][a/x]$, for all $a, b \in \mathbb{N}$.
(b) For all $a, b \in \mathbb{N}$, $\sigma[a/x][b/x] = \sigma[b/x]$.

1.1.4 Recall the definition of, and semantics for, the set *Exp* of expressions on pages 7 and 9. Prove, by structural induction, that, if the expression E contains no occurrence of the variable x, then $\mathcal{E}[\![E]\!]\sigma = \mathcal{E}[\![E]\!](\sigma[n/x])$ for all $n \in \mathbb{N}$. Deduce

(a) that if Γ is a program in our first example language which does not contain x, then for all $n \in \mathbb{N}$,

$$(\mathcal{C}[\![\Gamma]\!]\sigma)[n/x] \;=\; \mathcal{C}[\![\Gamma]\!](\sigma[n/x]) \quad \text{and}$$

(b) if E_2 does not contain x then

$$\mathcal{C}[\![x := E_1;\;\; x := E_2]\!] \;=\; \mathcal{C}[\![x := E_2]\!]\,.$$

1.2 Some languages cause problems

The simple language treated above caused us very few difficulties. Its main function was as vehicle to introduce the notation of denotational semantics. In particular there was no problem in devising and constructing the appropriate semantic model.

Consider the following simple functional language.

SYNTACTIC DOMAINS

N	$\in$	Nml	numerals
B	$\in$	Bas	basic constants
I	$\in$	Ide	identifiers
E	$\in$	Exp	expressions
Δ	$\in$	Dec	declarations

SYNTAX[5]

$$\begin{aligned}
B &::= N \mid * \mid + \mid - \mid \mathsf{div} \mid \ldots \mid \underline{true} \mid \underline{false} \ldots \\
E &::= B \mid I \mid \underline{if}\;\; E_0\;\; \underline{then}\;\; E_1\;\; \underline{else}\;\; E_2 \mid E_0 E_1 \mid \Delta \mathbf{in}\; E \\
\Delta &::= \underline{define}\;\; I \;=\; E \mid \underline{define}\;\; (I\; I_1\; I_2 \ldots I_n) \;=\; E \mid \\
&\quad\;\; \underline{rec}\;\; I \;=\; E \mid \underline{rec}\;\; (I\; I_1\; I_2 \ldots I_n) \;=\; E
\end{aligned}$$

The syntax of this language looks like that of a small purely functional subset of a dialect of Lisp. It bears some resemblance to the language described by Abelson

[5]The BNF clause for declarations here is non-standard because it contains, in the two clauses defining functions, an n which ranges over $\mathbb{N}$ in such a way that the syntactic construction varies. In fact, each of these 'clauses' really consists of an infinite family of very similar ones. If one wanted to define the same thing without infinitely many clauses, one could introduce an extra syntactic domain of parameter lists: defined using BNF to be any nonempty list of identifiers. The way we have described the syntax here is probably clearer, to the human eye, at least.

and Sussman in their text [] and also to Peter Henderson's Lispkit language []. Our interpretation of the language will be similar to theirs, but the language has deliberately been chosen not to coincide with either (or any other language we know of), so that we are not bound to place *exactly* the same interpretation on its constructs as was done before. An informal description of the meaning of the language constructs is given below. There are a number of example programs on the following pages that illustrate all of these.

In this language the terms 'program' and 'expression' are interchangeable, for the only way to compute anything is by evaluating an expression. Programs are made up from constants and identifiers with three constructs: a conditional expression $\underline{if}\ E_0\ \underline{then}\ E_1\ \underline{else}\ E_2$ takes value E_1 or E_2 depending on whether E_0 takes value $\underline{true}$ or $\underline{false}$. E_0E_1 is the result of applying the function E_0 to E_1, and $\Delta\mathbf{in}\ E$ evaluates E using the definitions created by the declaration Δ. Declarations can be of a form which defines an identifier to be the same as an expression $\underline{define}\ I\ =\ E$ or $\underline{rec}\ I\ =\ E$, or which define it to be a (Curried) function $\underline{define}\ (I\ I_1\ I_2 \ldots I_n)\ =\ E$ or $\underline{rec}\ (I\ I_1\ I_2 \ldots I_n)\ =\ E$ of any number of arguments which returns the value of the expression E with the names of the arguments replaced by the arguments themselves. The difference between the $\underline{define}$ and the $\underline{rec}$ forms is that the first is non-recursive (the meaning of any occurrence of I in the expression E has the same meaning as outside the scope of the declaration, namely the value 'before' the declaration) while the second is recursive (any occurrence of I within E has the meaning which is currently being defined).

Bas, the domain of basic constants, contains all the numerals, truth values, and also basic operators. Identifiers are used to denote objects within programs. They are not *variables*, because there is no assignment which can ever vary their values. All usual basic operators such as $+$, $<$ *and* etc. are treated as functions like any other and are thus used in prefix rather than the more usual infix form. Thus we write $(+\,5\,4)$ and $(<\,3\,4)$ rather than $5+4$ and $3<4$.

In case the reader has not met a language like this one before, the following are a few simple examples. The declaration Δ_1 defines a function that returns the greater of its two arguments:

$$\Delta_1\ \equiv\ \underline{define}\ (max\,n\,m)\ =\ \underline{if}\ (>\ n\,m)\ \underline{then}\ n\ \underline{else}\ m$$

Thus, $\Delta_1\mathbf{in}\ max(max\,3\,4)\,5$ would have value 5 and

$$\underline{define}\ (nonneg\,x)\ =\ (\Delta_1\mathbf{in}\ max\,x\,0)$$

defines a function which returns its argument, unless that was negative, in which case it returns 0.

A recursive definition can be used where looping constructs would be used

in a more traditional language. For example,

$$\underline{rec}\,(fib\,n) \;=\; \underline{if} \leq\; n\,2\,\underline{then}\,1 \\ \underline{else}\;(+\,(fib(-\,n\,1))(fib(-\,n\,2)))$$

defines a function which computes the nth Fibonacci number, where each is calculated as the sum of the previous two (after setting $fib\,1 = 1$ and $fib\,2 = 1$).

One of the most useful features is the way in which one can use *higher-order* functions, namely functions which are applied to other functions. For example, Δ_2 defines a function, *iterate*, that takes a function f, a natural number n and a starting value x and computes the result of applying f to x n times (i.e., $f^n(x)$).

$$\Delta_2 \;\equiv\; \underline{rec}\,(iterate\,f\,n\,x) \;=\; \underline{if}\,(=\;n\,0)\,\underline{then}\,x \\ \underline{else}\,(iterate\,f\,(-\,n\,1)\,(f\,x))$$

Higher order functions such as this one can have a wide variety of uses. For example,

$$\underline{define}\;\;(poweroftwo\,n) \;=\; (\Delta_2\mathbf{in}\;iterate\,(*\,2)\,n\,1)$$

defines a function for computing 2^n – notice how we have made use of the Curried function $*\,2$. And we could also use *iterate* in, for example, iterative numerical algorithms.

We will soon find that higher-order functions are a major difficulty when we tackle the semantics of this language. For it is, of course, our objective to define a suitable semantics for it. To do this we need suitable mathematical models. We will now spend some time discovering what properties these models must have.

Let us postulate a mathematical model M for all *closures* (programs without free identifiers). Clearly we can design such programs to compute basic things such as natural numbers and truth values, so all of these must belong to M. We will soon see that it needs to be much bigger.

Clearly the value of an expression with free identifiers (that is, identifiers used at points where no declaration binds them) will depend on the values of those identifiers. For example, the value of the expression $I_1\,I_2$ can only be determined once we know what I_1 and I_2 denote. We need something to play the rôle that states did in our first language. We therefore introduce the space *Env* of *environments*: each element ρ of *Env* will be a function from *Ide* to M, mapping each identifier to the object which it denotes. We use the name 'environment' rather than 'state' because there is no assignment in the functional language: the value of each identifier is determined purely by its declaration. (The term 'state' is usually reserved in semantics for something that can change dynamically during the execution of a program; 'environment' is used when the bindings are not changed except by declarations.)

Environments are created by declarations. The *scope* of a declaration is the part of the program in which it applies. In the programs

$$\underline{define}\ \ I = E_1 \mathbf{in}\ E_2 \quad \text{and}$$
$$\underline{rec}\ \ I = E_1 \mathbf{in}\ E_2$$

E_2 is evaluated in the external environment modified to map I to the value of E_1. Which environment is chosen for the evaluation of E_1 depends on whether '$\underline{define}$ ...' or '$\underline{rec}$ ...' is used. (In other words, on whether an occurrence of I within E_1 has the value presently being defined, or whatever value I might have in the outside world.) The semantic value of each declaration will be a function from *Env* to *Env*, because a declaration determines its 'internal' environment from the one in force 'outside'.

Let us return to the task of deciding what shape our main semantic domain, M, must have. There will be a domain B of *basic values*, such as 42, *true*, or *error*, that a closure might have as its value. Each of these values is complete in itself: it is not a function. Unfortunately we will need M to be rather larger than B. We want to be able to discover the values of compound programs from the values of their parts, and it is certainly possible for parts of reasonable programs to be *functions*. It is thus inevitable that M must contain some functions.

Some of these functions can be thought of as mappings from B to itself, for example

$$\begin{aligned} Fact \ \equiv\ & \underline{rec}\ (fact\, n) = \underline{if}\ (=\ n\, 0)\ \underline{then}\ 1 \\ & \qquad\qquad\qquad\qquad \underline{else}\ (\times\, n\, (fact(-\, n\, 1))) \\ & \mathbf{in}\ fact \end{aligned}$$

We would intuitively expect $Fact$ to represent a function from $\mathbb{N}$ to itself so that, for example, the expression $Fact\, 4$ has value 24. Consider next the expression *Loop*, which an inexperienced programmer might write in an attempt to produce a shorter factorial program than $Fact$:

$$\begin{aligned} Loop \ \equiv\ & \underline{rec}\ (sfact\ n) = (\mathsf{div}\ (sfact(+\ n\ 1))\ (+\ n\ 1)) \\ & \mathbf{in}\ sfact \qquad . \end{aligned}$$

He would expect this expression to have a value in $\mathbb{N} \to \mathbb{N}$, but we might suspect that $Loop\, E$ will, provided E represents a natural number, always loop infinitely and so not produce a value. Our domain will therefore need to be able to cope with functions which, on some arguments, give a completely undefined result. As discussed above, we will want to deal with higher-order functions, which take functions

as arguments. Consider, for example, the expression

$$H \equiv \underline{define}\ (double\ f\ n) = (*\ 2\ (f\ n))$$

$$\textbf{in}\ double \quad .$$

Given a function f that maps some domain X to $\mathbb{N}$, we would expect $H\ f$ to be the function from X to $\mathbb{N}$ that always returns twice what f does. For example, $H\ Fact$ will be the function that maps each natural number to twice its factorial: $H\ Fact\, 3 = 12$. If this is not bad enough, consider the extremely simple expression

$$Id \equiv \underline{define}\ (identity\ x) = x$$

$$\textbf{in}\ identity \quad .$$

Given any expression E, we would expect $Id\, E$ to have exactly the same value as E. Thus the 'type' of the function Id is $M \to M$. Actually, all the functions we have defined are more properly thought of as elements of $M \to M$, because nowhere in their definitions was there any restriction on the type of their argument or result other than that which have we assumed is imposed by the basic constants of the language. It is better to think of them as elements of $M \to M$ that map some arguments to *error* (assumed to be an element of M).

Thus we now expect that the model M will look like $B + (M \to M)$ (at least), where '+' is an operator on domains rather like disjoint union. In fact, as we will see below, any set satisfying

$$M \cong B + (M \to M)$$

can be used as a model for our language. Unfortunately, if $B \neq \emptyset$, this congruence seems impossible to solve! We know by Cantor's theorem that there can never be a 1-1 correspondence (bijection) between M and the function space $M \to M$ when M has more than one element.

This example shows that it is not always obvious how to construct the mathematical models needed for denotational semantics. The problems arise particularly acutely here, but in truth we are at the moment ill-equipped to construct *any* domain D which, like M, is defined to satisfy some *domain equation* $D \cong F(D)$. Before we can do this, we need to have a much clearer idea of what a 'domain' is. Chapter 3 introduces a mathematical theory of domains, in which an element of a domain is identified with the set of all propositions (from some set) that are true of

it. This will allow us to define such terms as 'convergence' and 'continuity' in the world of domains. We will then find that it is possible to solve the above equation with $M \to M$ re-interpreted to mean the *continuous* functions from M to itself. (It will turn out that all the functions definable in our languages are continuous in a sense which we will define later.) This restriction to continuous functions removes the cardinality problems alluded to above. We will only be able to see this when we have defined the terms involved, but observe that the continuous functions from the real numbers to themselves can be put into 1-1 correspondence with the set $\mathbb{R}$ of real numbers, because a continuous function is completely determined by its values on the rationals (see Exercise 1.2.3 below).

As remarked above, given a solution to the 'domain equation'

$$M \;\cong\; B \,+\, (M \to M)$$

it is not hard to give a semantics to the functional language (at least without $\underline{rec}$ ...). Recalling that $Env = Ide \to M$ is the space of environments, we require the following semantic functions.

$$\begin{array}{l}\mathcal{B} : Bas \to M\\ \mathcal{D} : Dec \to Env \to Env\\ \mathcal{E} : Exp \to Env \to M\end{array}$$

We assume that we are given the function $\mathcal{B}$, which tells us the value of each of our basic constants (including all the numerals). The target of $\mathcal{B}$ is M rather than B because some of these constants (for example $+$, $=$, $\underline{and}$) are functions. The definitions of $\mathcal{E}$ and $\mathcal{D}$ are given below. Note that their definitions, like those of their syntactic domains *Exp* and *Dec*, are mutually recursive.

$$\begin{array}{rcll}
\mathcal{E}[\![B]\!]\rho &=& \mathcal{B}[\![B]\!] & \\
\mathcal{E}[\![I]\!]\rho &=& \rho[\![I]\!] & \\
\mathcal{E}[\![\underline{if}\ E_0\ \underline{then}\ E_1\ \underline{else}\ E_2]\!]\rho &=& \mathcal{E}[\![E_1]\!]\rho & \text{if } \mathcal{E}[\![E_0]\!]\rho = true\\
&=& \mathcal{E}[\![E_2]\!]\rho & \text{if } \mathcal{E}[\![E_0]\!]\rho = false\\
&=& error & \text{otherwise}\\
\mathcal{E}[\![E_0\ E_1]\!]\rho &=& \mathcal{E}[\![E_0]\!]\rho(\mathcal{E}[\![E_1]\!]\rho) & \text{if } \mathcal{E}[\![E_0]\!]\rho \text{ is in } M \to M\\
&=& error & \text{otherwise}\\
\mathcal{E}[\![\Delta \textbf{ in } E]\!]\rho &=& \mathcal{E}[\![E]\!](\mathcal{D}[\![\Delta]\!]\rho) &
\end{array}$$

Some of these definitions are not completely satisfactory, because we do not yet have a full understanding of all that is required. In particular, it would be better to pay

more attention to the possibilities of non-termination (such as that of the incorrect factorial function *Loop* above – if it were applied to a number we would expect to get an infinite loop, not the defined value *error*). For example, if the evaluation of E_0 in either the conditional or the application clause fails to terminate, it would be better if the whole clause failed to terminate also. Note that in the clause for declarations we are evaluating E in the environment created by the declaration Δ.

Remembering that we are considering only non-recursive declarations, the effect of a simple (non-functional) declaration is easy to calculate:

$$\mathcal{D}[\![\underline{define}\ \ I = E]\!]\rho \quad = \quad \rho[\mathcal{E}[\![E]\!]\rho/I]\ .$$

To calculate the result of a function of one variable, we work out the value of the body in the environment in which the argument to function has been substituted for the identifier which is the function's formal parameter.

$$\mathcal{D}[\![\underline{define}\ \ (I\ I_0) = E]\!]\rho \quad = \quad \rho[\psi/I],$$

where $\psi \in M \to M$ is such that

$$\psi\alpha \ = \ \mathcal{E}[\![E]\!]\rho[\alpha/I_0] \text{ for each } \alpha \in M.$$

A function which takes $n+1$ arguments produces a function from M to itself whose result is a function that takes n arguments.

$$\mathcal{D}[\![\underline{define}\ \ (I\ I_0\ \ldots I_n) = E]\!]\rho \quad = \quad \rho[\psi/I],$$

where $\psi \in M \to M$ is such that

$$\psi\alpha \ = \ (\mathcal{D}[\![\underline{define}\ \ (I\ I_1 \ldots I_n) = E]\!]\rho[\alpha/I_0])[\![I]\!] \text{ for each } \alpha \in M.$$

Notice that the recursion involved in the definition of $\mathcal{D}$ is slightly nonstandard, in that the definition for a function of $n+1$ arguments depends on the values of declarations which, though shorter, are not syntactically simpler in the usual sense. It is clear that this causes no problems, however. (An equivalent definition obeying the usual conventions can be given, but it is more cumbersome.)

Let us consider what would be required for the semantics of $\underline{rec}$ ... declarations. In the case of the non-functional style, we would require

$$\mathcal{D}[\![\underline{rec}\, I = E]\!]\rho \quad = \quad \rho[\epsilon/I]\ .$$

where $\epsilon = \mathcal{E}[\![E]\!]\rho[\epsilon/I]$. Thus the value needed to substitute for the identifier I is not defined directly but is the solution to an equation. At the moment we have no reason to believe that this equation has any solution, and if it had several we would not know which one to pick. Exactly the same problem appears in the case of a functional $\underline{rec}$ ... declaration. In later chapters we will discover how to find the required solutions to equations of this type, and will thus be able to deal with semantic definitions like this one.

One surprising fact is that the introduction of recursive declarations, which as we observed above adds significantly to our difficulty in defining semantics, adds nothing to the expressive power of the language. Consider the recursive definition of the factorial function $fact$ above. It may be replaced by the following.

$$\begin{array}{l} \underline{define}\ fact\ =\ (\underline{define}\ (G\,f\,n)\ =\ \underline{if}\ (=\ n\,0)\ \underline{then}\ 1 \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \underline{else}\ (\times\, n\,(f\,f\,(-\,n\,1))) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{in}\ (G\,G) \end{array}$$

To understand this definition, consider the expression $fact\,3$ which reduces to $G\,G\,3$. Since 3 is not equal to 0 this reduces to $\times\,3\,(G\,G\,2)$ – note that G copies itself as it is applied to itself. Repeating this process twice reduces the expression further to $\times\,3\,(\times\,2\,(\times\,1\,(G\,G\,0)))$. But $G\,G\,0$ reduces to 1, so the expression evaluates to 6 as hoped.

The idea of *self-application* used here is so powerful that it can be used to replace all recursive definitions non-recursively. But note how closely it depends on the structure of our domain $M \cong B + (M \to M)$, for if the domain M did not contain a component of the form $M \to N$ for some N then it could not make any sense to apply an object to itself. In fact what we have found is that a solution to our domain equation is powerful enough to construct directly solutions to the sorts of equations we came up against when attempting to give the semantics of $\underline{rec}$. More than anything else this perhaps illustrates the power and paradoxical qualities of the domains we are trying to create.

We *could* have defined the semantics of $\underline{rec}$ ... by this sort of trick, but this would neither have been elegant nor standard. Also, it would simply have cancelled the need for one thing we do not yet know how to do (solve equations such as $\epsilon = \mathcal{E}[\![E]\!]\rho[\epsilon/I]$) by reference to another (solve the domain equation $M \cong B + (M \to M)$). In fact the second of these will turn out to be the more difficult.

Exercises

1.2.1 What is the meaning (semantic value) of the following program fragments in the above language. If replacing $\underline{define}$ by $\underline{rec}$ would have made any difference, give both meanings.

(i) $\underline{define}\ (f\ x) = (\underline{if}\ (=\ x\ 0)\ \underline{then}\ 0\ \underline{else}\ (+\ (f(-\ x\ 1))\ x)$
$\quad\quad\mathbf{in}\ (f\ 3)$

(ii) $\underline{define}\ x = 0\ \mathbf{in}\ (+\ (\underline{define}\ x = 1\mathbf{in}\ x)\ x)$

(iii) $\underline{define}\ (f\ x) = (x\ x)\ \mathbf{in}\ (f\ 3)$

(iv) $\underline{define}\ (f\ x) = (x\ x)\ \mathbf{in}\ (f\ f)$

(v) $\underline{define}\ (f\ x) = (x\ x)\ \mathbf{in}\ (\underline{define}\ (g\ x) = 3\ \mathbf{in}\ (f\ g))$

1.2.2 Formulate non-recursive declarations that are equivalent to the following recursive ones:

(i) $\underline{rec}\ (iter\ f\ x) = \underline{if}\ (<\ abs(-(f\ x)\ x)\ tolerance)\ \underline{then}\ x$
$\quad\quad\underline{else}\ (iter\ f\ (f\ x))$

where *abs* computes the absolute value (modulus) of a number. (This is a function which finds approximate solutions of the equation $f\ x = x$ for suitable functions f and starting values x_0 for the iteration $iter\ f\ x_0$. Notice how this function varies slightly from the function *iterate* defined in the text earlier: it iterates until a condition holds (which might be never) rather than a fixed number of times.)

(ii) The declaration of *fib* on page 16.

^{M}1.2.3 Show that the set of all sequences $\langle x_i \mid i \in \mathbb{N}\rangle$ of real numbers has the same cardinal as $\mathbb{R}$ (i.e., $2^{\aleph_0} = c$, the continuum). Deduce that the set of continuous functions from $\mathbb{R}$ to $\mathbb{R}$ also has the same cardinal as $\mathbb{R}$. (Two sets X and Y are said to have the same cardinal when they can be put in 1-1 correspondence; but it is provably sufficient to produce 1-1 maps $f : X \to Y$ and $g : Y \to X$.)

Chapter 2

Partial orders

The rest of this book will depend heavily on partial orders. For not only will the 'domains' we create be partial orders (see Chapter 3) but also we will make use of an order between domains themselves (see Chapter 4). This chapter covers all of the material on partial orders needed later on, and more generally gives an introduction to essentials of partial order theory as used commonly in computer science. It is divided into four sections: the first three are elementary and cover all that is required later, while the final section contains some slightly more advanced material that will only be accessible to readers with a mathematical background. We would expect the final section to be omitted from most introductory courses.

2.1 Basics

A partial order is just a relation between objects of some sort indicating when one is smaller than another in some way. Thus, formally, it is a set X together with a binary relation such as $\leq$ or $<$ which satisfies certain axioms. The first symbol (or variants of it) usually means a less-than-or-equal-to type order which satisfies the following:

1. $x \leq y \wedge y \leq z \Rightarrow x \leq z$
2. $x \leq y \wedge y \leq x \Leftrightarrow x = y$.

In other words it is transitive (Axiom 1), reflexive (namely, $x \leq x$ for all x) and otherwise asymmetric (Axiom 2). The second symbol is used for *strict* partial orders where no element is less than itself. They are characterised by being transitive (Axiom 1) and by a different asymmetry axiom:

$2'$. $x < y \Rightarrow \neg(y < x)$

In mathematical literature the term 'partial order' refers to either of these. In this book we will usually use the $\leq$ type – but it should always be clear from the notation used whether we are using one or the other. Indeed we will often use them for the same order depending on whether we mean 'less than or equal to' or 'strictly less than'. See Exercise 2.1.2 for a demonstration that there is a natural correspondence between the two sorts of order. We will also use reversed order symbols in the natural way: for example $x \geq y$ means the same as $y \leq x$.

We should remark at this point that any partial order can be turned upside-down: if we define $x \preceq y \Leftrightarrow y \leq x$ for some partial order $\leq$, then $\preceq$ is a partial order too.

There are a large number of partial orders which should be known to anyone familiar with basic mathematics, for example the following.

EXAMPLES

(a) The real numbers, rationals, integers or natural numbers with their usual less-than-or-equal-to order.

(b) The powerset (set of all subsets) of a given set ordered by inclusion ($\subseteq$).

(c) The finite sequences of letters ordered lexicographically (i.e., as in a dictionary), where the first letter is most significant. Thus, for example,

$$\langle\rangle \leq \langle a\rangle \leq \langle aa\rangle \leq \langle b\rangle \leq \langle bazzzzzz\rangle \leq \langle bb\rangle\,.$$

(d) The finite sequences of letters ordered by *prefix*: $s \leq t$ if and only if there is a sequence r such that $s\hat{}r = t$, where $s\hat{}r$ is the concatenation of s and r (so $s \leq t$ if and only if s is an initial subsequence of t). ∎

We can notice several things from these examples. From (a) it should be apparent that any subset of a partial order inherits an order, just as the order on the natural numbers can be thought of as restriction to the natural numbers of the usual order on the reals. From (c) and (d) it is clear that the same set can be given a number of different orders. Note that in (b) and (d) there are pairs of elements x, y such that neither $x \leq y$ nor $y \leq x$. However this situation nevers occurs in (a) or (c), which both turn out to be *total* (or equivalently *linear*) orders, satisfying an additional axiom:

3. $x \leq y \vee y \leq x$.

Often we will wish to draw pictures of partial orders to illustrate various points. Figure 2.1 is an example of the style of illustration: it shows Example (b) above in the case where the underlying set is $\{a, b, c\}$. The partial order relation is shown by the lines: objects which have a line between them are related with

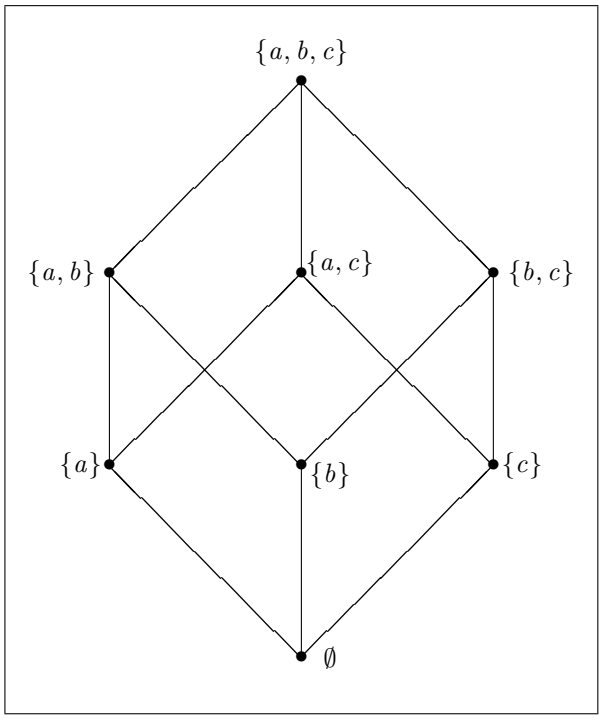

Figure 2.1 A simple partial order.

the higher one in the picture being greater. Not all such relations are shown – the partial order which such a picture determines is the smallest one which orders all the pairs shown ordered, and which is reflexive and transitive in the senses discussed above: almost invariably we will only draw the lines between different objects x and y with no other between them. Such illustrations are often called *Hasse diagrams* in the literature.

In many parts of pure mathematics one defines two objects to be *isomorphic* if they are essentially the same (at least in the terms which they are being studied).

Definitions
If $(X, \leq)$ and $(Y, \preceq)$ are two partial orders, then we say that the function $f : X \to Y$ is an *order isomorphism* if and only if

(a) f is 1-1 (or injective), namely, $f(x) = f(x') \Rightarrow x = x'$.

(b) f is onto (or surjective), namely, for every $y \in Y$ there is $x \in X$ such that $f(x) = y$.

(c) f and f^{-1} are order-preserving, namely, $x \leq x' \Leftrightarrow f(x) \preceq f(x')$.

$(X, \leq)$ and $(Y, \preceq)$ are said to be *order isomorphic* if there is an order isomorphism between them. (Note that any order isomorphism has an inverse which is also an order isomorphism.) ∎

From the point of view of partial order theory, two isomorphic partial orders are essentially the same and have all the same properties. The *order type* of a given partial order is the class of all partial orders to which it is isomorphic. Thus, though there are clearly infinitely many different partial orders with any given number of elements, there are only finitely many order types for orders with any fixed finite number (see Exercise 2.1.1).

We will come back to the study of functions between partial orders in the next section.

Given a subset X of a partial order P, we say that $x \in P$ is an *upper bound* for X if $y \leq x$ for all $y \in X$. Similarly x is a *lower bound* if $y \geq x$ for all $y \in X$. In Figure 2.1 note that $\{a, b\}$ and $\{a, b, c\}$ are the only upper bounds for $\{\{a\}, \{b\}\}$ and that the only lower bound of $\{\{a, b\}, \{b, c\}, \{a, c\}\}$ is $\emptyset$. Notice that if $X = \emptyset$ then *every* element of the partial order is both an upper and a lower bound for X. Sometimes a set may have no upper (or lower) bound. In Example (d) above (finite sequences of letters ordered by prefix) it is clear that the set $\{\langle a\rangle, \langle c\rangle\}$ has no upper bound since no sequence begins both with an a and a c. In Example (a), notice that none of the sets of all reals, rationals or integers has either an upper or lower bound.

We will find at frequent points in the following chapters that we need some idea of 'convergence' within partial orders – for we will often want to construct objects as limits. This will be done by giving better and better approximations to the desired object – where 'better' means greater in the partial order. It is thus helpful if increasing sequences of points determine some unique limit. The appropriate 'limit' of such a sequence turns out to be its *least* upper bound, if it has one.

$X \subseteq P$ has *least upper bound* x if x is both an upper bound for X and whenever y is another upper bound for X then $x \leq y$. Similarly x is the *greatest lower bound* of X if it is both a lower bound and greater than all others. Notice that least upper bounds and greatest lower bounds are, when they exist, necessarily unique. (For example, if X had least upper bounds x and y, then necessarily $x \leq y$ and $y \leq x$ proving that $x = y$ by one of the axioms of partial orders.)

All subsets X of the order in Figure 2.1 have a least upper bound given by $\bigcup X$ and a greatest lower bound given by $\bigcap X$ (where $\bigcap \emptyset$ is defined to be the 'universal set' $\{a, b, c\}$). In a general order it is usual to modify these symbols a little and denote the greatest lower bound and least upper bound of a set by $\bigsqcap X$ and $\bigsqcup X$ respectively. Just because a set has an upper bound (or lower bound) does not mean that it has a least (or greatest) one. One example of this is shown in

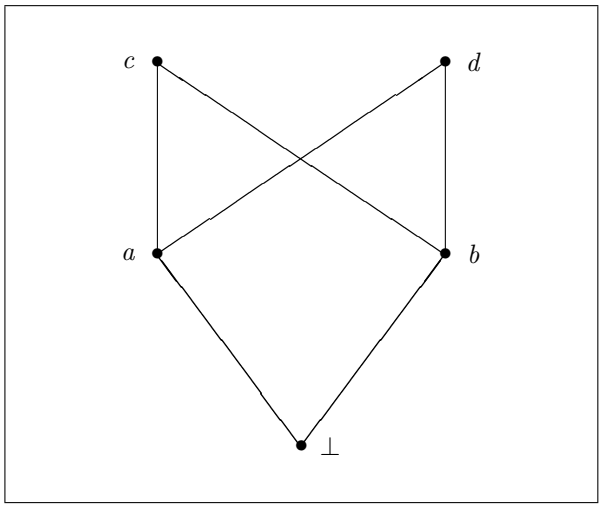

Figure 2.2 Least upper bounds need not exist.

Figure 2.2, where the elements a and b have two upper bounds c and d but no least one. We have given this domain a least element – given its conventional name of $\bot$ (pronounced 'bottom') – though it is not necessary here, because this will be useful later. A second example is given by the rational numbers (Example (a) above), where, if we set $X = \{q \mid q < \sqrt{2}\}$ then X has no least upper bound since there are rationals arbitrarily close on either side to $\sqrt{2}$, even though $\sqrt{2}$ is not itself rational. A partial order where, for every nonempty finite X, both $\bigsqcap X$ and $\bigsqcup X$ exist, is called a *lattice* (note that the rationals and every other linear order form a lattice). If $\bigsqcap X$ and $\bigsqcup X$ exist for *every* X, then the order is called a *complete lattice*. But every complete lattice contains a greatest ('top') element (the least upper bound of the whole order) and many of the partial orders we need to study later will have no top, so we need to study sorts of partial order where we guarantee the existence of least upper bounds (i.e., limits) for more restricted classes of subsets.

Least upper bounds will play the rôle of the limits of converging sequences described above. For technical reasons which need not concern us here[1] it is better to generalise the obvious notion of increasing sequence discussed earlier when we look at convergence in partial orders.

Definition
Suppose that $(P, \leq)$ is a partial order. We say that $\Delta \subseteq P$ is *directed* if

(a) Δ is nonempty, and

[1] But see Section 2.4.

(b) if $x, y \in \Delta$, then there is some $z \in \Delta$ such that $x \leq z$ and $y \leq z$. ∎

Any linearly ordered nonempty subset of a partial order P is directed, as is any set which contains a greatest element. But not all directed sets are linearly ordered – consider the set of finite subsets of any set, ordered by $\subseteq$.

The vital property of directed sets is that, given any $F \subseteq^{\text{fin}} \Delta$, with Δ directed, there is some upper bound for F in Δ; in other words some $x \in \Delta$ such that $y \leq x$ for every $y \in F$. (This is easily proved by induction on the size of F, (a) and (b) above providing respectively the base case and inductive step.) We will use this fact repeatedly in proofs.

Definition
The partial order $(P, \leq)$ is said to be *complete* if it has a least element (also called a bottom or minimum element) and every directed $\Delta \subseteq P$ has a least upper bound. ∎

We will usually abbreviate 'complete partial order' by 'cpo'.

If Δ is a directed set and $F \subseteq^{\text{fin}} \Delta$, the set

$$\Delta_F = \{a \in \Delta \mid \forall\, b \in F.\ b \leq a\}$$

is directed, for we have already observed it is nonempty and it clearly inherits property (b) from Δ. We might think of Δ_F as consisting of the 'sufficiently large' elements of Δ or as those elements which are sufficiently converged. If $a \in \Delta$, there exists $a' \in \Delta_F$ with $a \leq a'$, for we know Δ contains an upper bound for $F \cup \{a\}$. Thus for every finite collection of 'information' that appears among the elements of a directed set Δ, every sufficiently large element of Δ contains all of that information. Indeed it is easy to see that if one of the sets Δ and Δ_F has a least upper bound z, then z is also the least upper bound of the other. Hence it is plausible to think of a directed set as being an analogy of a Cauchy sequence in real analysis (informally, a Cauchy sequence[2] is one that looks as though it ought to be convergent). Thus a complete partial order is one in which each 'Cauchy sequence' Δ converges to a limit $\bigsqcup \Delta$.

Notice that the least element of a cpo is equal to $\bigsqcup \emptyset$.

Of the examples given earlier the only cpo is the powerset (Example (b)). For the set $\{0, 1, 2, 3, \ldots\}$ is directed in each case of Example (a), but with no upper bound. And in Examples (c) and (d) the set $\{\langle z\rangle, \langle zz\rangle, \langle zzz\rangle, \ldots\}$ is directed with no

[2]Formally a Cauchy sequence $\langle x_i \mid i \in \mathbb{N}\rangle$ is a sequence of real numbers such that for all $\epsilon > 0$ we can find $N \in \mathbb{N}$ so that $n, m \geq N$ implies $\mid x_n - x_m \mid < \epsilon$; this definition can be adapted easily to any metric space. They have the property that any infinite subsequence is also a Cauchy sequence, with the same limit as the original, if any. A complete metric space is one where every Cauchy sequence converges, giving a very similar definition to that of a cpo.

upper bound. It is worth noting, however, that any finite partial order with a least element is complete, since any finite directed set contains its least upper bound (see Exercise 2.1.3).

Complete partial orders are widely used in computer science, for as we shall see in the next section of this chapter, a large class of functions over them have a property that allows us to give meaning to recursive programs. For the rest of this chapter we will concentrate mainly on cpos.

Definition
Suppose $(P, \leq)$ is a partial order and $S \subseteq P$. We say that S is *po-consistent* if each of its finite subsets has an upper bound in P. (Thus, a po-consistent finite set S must have an upper bound in P, but this need not hold in general for infinite S. For example, in the partial order of finite subsets of $\mathbb{N}$, any set is po-consistent, but *only* finite sets have upper bounds.)

We have called it 'po-consistency' (with 'po' standing for 'partial order'), rather than 'consistency', which is its more usual name, to avoid confusion with other definitions which will appear later and which are more important in the context of this book. ∎

Definition[3]
A complete partial order $(P, \leq)$ is said to be *consistently complete* if every po-consistent $S \subseteq P$ has a least upper bound. ∎

We have already seen an example of a cpo that is not consistently complete – namely the one in Figure 2.2. However many cpos one meets do have this property and it is a useful one to have, as is illustrated by the following result.

2.1.1 LEMMA
Any nonempty subset X of a consistently complete cpo P has a greatest lower bound.

PROOF

[3]The definitions of po-consistency and consistent completeness vary a little in the literature. For example, some authorities only allow that a set is po-consistent if it has an upper bound. In noncomplete partial orders this different definition would change the meaning of consistent completeness. However all the definitions of consistent completeness mean the same thing when applied to cpos. Since we only need to discuss consistent completeness in the context of cpos, we have therefore restricted our definition to this case. Indeed, the reader might notice that our definition of consistently complete implies completeness (in nonempty partial orders). This is because any directed set is po-consistent, and so also is the empty set, whose least upper bound is the least element.

The set Y of all lower bounds for X is po-consistent (for each $x \in X$ is greater than all $y \in Y$). Hence $\bigsqcup Y$ exists. But each element x of X is an upper bound for Y and $\bigsqcup Y$ is the *least* upper bound – it follows that $\bigsqcup Y \leq x$ and hence that $\bigsqcup Y \in Y$. Clearly $\bigsqcup Y$ is the *greatest* lower bound for X. ∎

For the dual of this result see Exercise 2.1.6. Perhaps because of this property, which is nearly as strong being a complete lattice, consistently complete cpos are sometimes termed *complete semilattices*. But we shall not generally do so.

Definition
An element x of the complete partial order $(P, \leq)$ is said to be *po-finite* if, whenever $\Delta \subseteq P$ is directed with $\bigsqcup \Delta \geq x$, there is some $y \in \Delta$ such that $y \geq x$. ∎

Once again we have added the prefix 'po' in order to avoid confusion later on. The po-finite elements of a cpo are in some sense those elements x which can be characterised by a finite amount of information. Because no directed set Δ can have limit above x without containing each piece of information, and every finite subset of Δ has an upper bound within Δ, there is some element of Δ that contains all the information required to characterise x. A po-finite element can be thought of as being 'isolated' from the others, because it is impossible to converge to it in any interesting way.

It is easy to see that the po-finite elements of the cpo in Example (b) above are precisely the finite sets – so here, at least, po-finite really does mean finite!

2.1.2 LEMMA

(a) If P is a cpo and E is a finite set of po-finite elements of P, then if $\bigsqcup E$ exists, it is po-finite.

(b) Suppose that P is a consistently complete cpo and that $x \in P$, then the set $\{e \in P \mid e \leq x \text{ and } e \text{ is po-finite}\}$ is directed.

PROOF
(a) Suppose Δ is a directed set such that $\bigsqcup \Delta \geq \bigsqcup E$, and that $E = \{e_1, e_2, \ldots, e_n\}$. Then for each i we have $\bigsqcup \Delta \geq e_i$, so there is some $x_i \in \Delta$ such that $x_i \geq e_i$. By directedness of Δ there is then some $x^* \in \Delta$ which is an upper bound for $\{x_1, x_2, \ldots, x_n\}$. Clearly $x^* \geq \bigsqcup E$ as required.
(b) Observe that the least element of every cpo is po-finite. Thus the set $\{e \in P \mid e \leq x \text{ and } e \text{ is po-finite}\}$ is nonempty. Every subset of it must have a least upper bound because it is po-consistent by construction (x is an upper bound) and P is consistently complete. Thus each two-element subset $\{e, f\}$ has a least upper bound which is po-finite by (a), and obviously $\bigsqcup\{e, f\} \leq x$. ∎

The cpos used in computer science tend to have the following property.

Definition
The complete partial order P is said to be *algebraic* if, for every $x \in P$, the set $\{e \in P \mid e \leq x \text{ and } e \text{ is po-finite}\}$ is directed with limit x. ∎

This says that each element is determined by the po-finite ones less than it. Thus even though the po-finite elements can never themselves be the limits of directed sets not containing them, combinations of them converge to every point in the space. An important strengthening of this condition is where the set of all po-finite elements is countable – such domains are said to be $\omega-$algebraic. It is often natural to assume one's domains to be ω-algebraic because computers themselves (even idealised) are usually countable in extent. However it is not an assumption that would gain us much in the rest of the book so we will not usually make it. Because ω-algebraic cpos give identifications of po-finite elements with natural numbers and of general elements with sets of natural numbers, there has been a substantial body of work on links between these, and subclasses (for example, po-consistent ones where the least upper bound function on po-finite elements is computable) and the formal theory of computability (recursive function theory.)

The partial order of Example (b) above (the powerset) is easily seen to be algebraic (and ω-algebraic if the underlying set is countable). However there are consistently complete cpos that are not algebraic – for example, the closed interval $[0,1] = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$, where the only po-finite element is 0 – and algebraic cpos that are not consistently complete (Figure 2.2 again).

The partial orders used by computer science often have the property that the po-finite elements are in some sense objects which are observable in a finite time, while infinite (i.e., non-po-finite ones) are ones which either have infinitely much to observe or are even the results of computations which take infinitely long. For a cpo to be algebraic in this context is rather like saying that any computation is determined by all of the finite observations which can be made of it.

Exercises

2.1.1 How many different partial orders are there, up to isomorphism, with n elements when $n = 1, 2, 3, 4$?

2.1.2 Show that if $(P, <)$ is a strict partial order then the relation $x \leq y \Leftrightarrow x < y \vee x = y$ is a non-strict partial order, that different $<$ yield different $\leq$ and that every $\leq$ is obtained from some $<$. This shows that there is a natural correspondence between strict and non-strict partial orders.

2.1.3 Prove that every finite directed set contains its least upper bound and hence

that every finite partial order with a bottom element is complete.

2.1.4 Suppose that in the partial order P all of the sets $\{\psi(x,y) \mid y \in B\}$ (as x varies over A) and $\{\psi(x,y) \mid x \in A \wedge y \in B\}$ have least upper bounds. Show that $\bigsqcup\{\psi(x,y) \mid x \in A \wedge y \in B\}$ is the least upper bound of $\{\bigsqcup\{\psi(x,y) \mid y \in B\} \mid x \in A\}$. (This is a fact that we will frequently make use of in proofs.)

2.1.5 (a) Show how Example (d) above becomes a cpo if we extend the domains to include infinite sequences $\langle a_i \mid i \in \mathbb{N}\rangle$ of letters with natural extensions to the partial order.

*(b) Show that the same is true of Example (c).
[Hint: Since this is a linear order you must show that any subset has a least upper bound. One approach to this is to compute the upper bound of a set S one element at a time. Another is to identify the extended order with a certain subset of the real numbers – think what you could do if there were only eight letters in the alphabet.]

Determine whether the extended orders are algebraic.

2.1.6 Prove the reverse of Lemma 2.1.1, namely that, in a cpo where every nonempty set has a greatest lower bound, every po-consistent set has a least upper bound.
[Hint: Prove that finite po-consistent sets have least upper bounds, and generalise.]

2.1.7 Suppose $(P, \leq)$ is a consistently complete cpo. Show that it becomes a complete lattice when we add a new element $\top$ (pronounced 'top') and define $x \leq \top$ for all $x \in P \cup \{\top\}$. Show that $\top$ is po-finite in this order.

2.1.8 Suppose that a and b are two po-finite elements of the complete partial order P, and that $\bigsqcap\{a,b\}$, the greatest lower bound of a and b, exists. Need $\bigsqcap\{a,b\}$ be po-finite? (Give a proof or a counter-example.)

2.1.9 Classify the following partial orders. (In other words, which of the properties discussed in this section does each of them have?)

(a) Given any set X and a special element $\perp \notin X$, define the *flat* order on $X \cup \{\perp\}$ by setting $\perp \leq x$ for all x, and if $y \in X$ then $y \leq x \Leftrightarrow y = x$.

(b) Define an order $\preceq$ on pairs of natural numbers by

$$(n,m) \preceq (n',m') \Leftrightarrow m + n < m' + n' \vee (n = n' \wedge m = m')\,.$$

Now extend this order by a single top element $\top$.

(c) The natural numbers $\mathbb{N}$ with the order $n \preceq m$ if and only if n divides m. (Note that all numbers divide 0.)

2.2 Functions and fixed points

We have already met one form of function between partial orders: the order isomorphism. The following is a generalisation of that class – retaining only part of the third property.

Definition
If P and Q are partial orders and $f : P \to Q$ then f is said to be *monotonic* or *order preserving* if, whenever $x, y \in P$ are such that $x \leq y$, then $f(x) \leq f(y)$. ∎

Informally, a monotonic function can be thought of one where, the more one puts in, the more one gets out.

Recall the example partial orders on page 24. In Example (a) the function $x \mapsto x+1$ is monotone, whereas $x \mapsto -x$ (not defined over $\mathbb{N}$) is not. Over Examples (c) and (d) the function $s \mapsto \langle c \rangle^\frown s$ is monotone, but the function $s \mapsto s^\frown \langle c \rangle$ is not. (The reader might like to consider why not, and also to show that the function $s \mapsto s^\frown \langle a \rangle$ is monotone in (c) but not (d).)

We shall find that almost all of the functions that arise naturally from our work later on will be monotonic – a most useful property as we shall see later. One obvious property that they have is to map directed sets to directed sets, since if $x, y \in \Delta$ then there is $z \in \Delta$ such that $x \leq z$ and $y \leq z$ – it follows that $f(x) \leq f(z)$ and $f(y) \leq f(z)$.

The following simple result is one that we will use frequently.

2.2.1 LEMMA
If P and Q are partial orders, $f : P \to Q$ is monotone, $X \subseteq P$, and $\bigsqcup X$, $\bigsqcup\{f(x) \mid x \in X\}$ both exist then

$$\bigsqcup\{f(x) \mid x \in X\} \leq f(\bigsqcup X)\,.$$

PROOF
Note that $x \leq \bigsqcup X$ whenever $x \in X$, and so $f(x) \leq f(\bigsqcup X)$. This means that $f(\bigsqcup X)$ is an upper bound for the set $\{f(x) \mid x \in X\}$ and so is greater than the least upper bound. ∎

The concept of a function whose value at a limit point can be determined from its values on a sequence converging to that point is familiar to students of real analysis and topology: it is said to be continuous. Recalling that we are thinking of a directed set in a cpo as being a generalised convergent sequence, we can extend the idea of continuity to partial orders.

Definition
If P and Q are two cpos and $f : P \to Q$, then f is said to be *continuous* if, whenever $\Delta \subseteq P$ is directed, $\bigsqcup\{f(x) \mid x \in \Delta\}$ exists and equals $f(\bigsqcup \Delta)$. ∎

The following lemma shows that continuity implies monotonicity. This is no accident: indeed, if the following lemma did not hold, we would have incorporated monotonicity into the *definition* of continuity.

2.2.2 LEMMA
Suppose P, Q are cpos and $f : P \to Q$ is continuous. Then f is monotonic.

PROOF
Suppose $x, y \in P$ and $x \leq y$. Then $\{x, y\}$ is clearly a directed set and $\bigsqcup\{x, y\} = y$. Hence, by continuity of f we have $f(y) = \bigsqcup\{f(x), f(y)\}$. This trivially means that $f(x) \leq f(y)$. ∎

Because we already know that monotone functions preserve directedness, this lemma shows that continuous functions map 'convergent sequences' to 'convergent sequences', and by definition map the limit of one to the limit of the other[4]. Not all monotonic functions need be continuous: consider the partial order $P = (\mathcal{P}(\mathbb{N}), \subseteq)$ (subsets of the natural numbers under set inclusion), and the function $f : P \to P$ defined

$$f(x) = \begin{cases} \emptyset & \text{if } x \text{ is finite} \\ \mathbb{N} & \text{if } x \text{ is infinite.} \end{cases}$$

Continuity fails for this function whenever Δ is an infinite directed set of finite sets, for example $\Delta = \wp(\mathbb{N})$.

Figure 2.3 illustrates the definitions of monotonicity and continuity by three more simple examples. The partial orders with 'and so on' dots (ellipsis) all consist of an infinite increasing sequence below a single limit point.

The following lemma is very easy to prove.

2.2.3 LEMMA
Suppose P, Q and R are partial orders and $f : P \to Q$, $g : Q \to R$. Then the composition $f \circ g$ of these functions is monotone (respectively continuous) if both f and g are monotone (continuous). ∎

In algebraic cpos where the po-finite elements represent the finitely computable objects, it is reasonable to argue that all functions resulting from real processes are continuous. For what is observable in a finite time about the result of applying the function can only have arisen from what the function has itself

[4]These properties define what it means to be continuous in real analysis or over metric spaces.

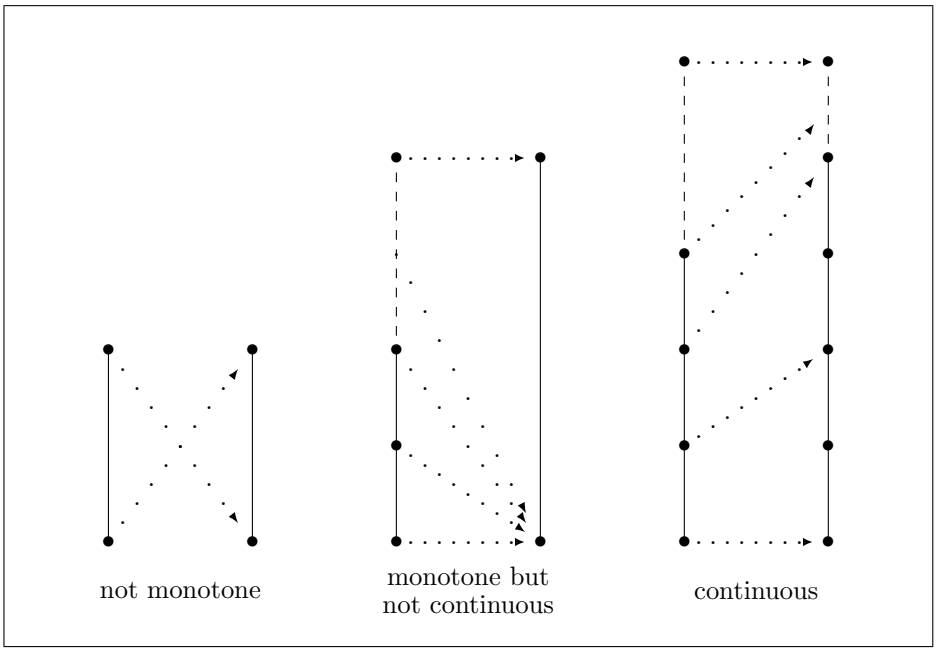

Figure 2.3 Montonicity and continuity.

'observed' of the argument in a finite time. We will return to this idea in later chapters when dealing with rather more concrete partial orders where it will be more tangible.

The next result shows the extent to which continuous functions on such domains are determined by their restriction to po-finite elements. It shows that every monotone function on the po-finite elements of an algebraic cpo has a unique continuous extension.

2.2.4 LEMMA
Suppose P and Q are cpos and P is algebraic with set E of po-finite elements. Then the continuous functions from P to Q are in 1-1 correspondence with the monotone functions from E to Q; the correspondence being given by $f \mapsto f\restriction E$ (i.e., the restriction of f to the smaller domain).

PROOF
If f is continuous then Lemma 2.2.2 tells us it is monotone and hence $f\restriction E$ is monotone. But $x \in P$ implies $x = \bigsqcup\{e \mid e \leq x \wedge e \in E\}$ – a directed set (Lemma 2.1.2) – and hence $f(x) = \bigsqcup\{f(e) \mid e \leq x \wedge e \in E\}$. This proves that f is determined

by $f|^{\setminus}E$.

Now, if $g : E \to Q$ is monotone, define $f_g : P \to Q$ by

$$f_g(x) = \bigsqcup\{g(e) \mid e \leq x \wedge e \in E\} .$$

(Note that this is well defined since the set on the right hand side is directed.) $f_g|^{\setminus}E = g$, for plainly if $e \in E$ then $g(e)$ is the greatest element (and hence the least upper bound) of $\{g(e') \mid e' \leq e \wedge e' \in E\}$.

f_g is trivially monotone, which means that $\bigsqcup\{f_g(x) \mid x \in \Delta\}$ exists and by Lemma 2.2.1 is less-than-or-equal-to $f_g(\bigsqcup \Delta)$. But if $e \in E$ is such that $e \leq \bigsqcup \Delta$ then (by definition of po-finite) there is $x \in \Delta$ with $e \leq x$. Hence $f_g(e) \leq f_g(x)$, and so

$$\bigsqcup\{f_g(e) \mid e \leq \bigsqcup\Delta \wedge e \in E\} \leq \bigsqcup\{f_g(x) \mid x \in \Delta\} ,$$

and by what we already know ($f_g|^{\setminus}E = g$) the left hand side here is just $f_g(\bigsqcup \Delta)$. This proves that f_g is continuous.

This proves the result since we have shown that the correspondence between the two function spaces is 1-1 and onto. ∎

This has much in common with the result alluded to in Exercise 1.2.3 that a continuous function from the rational $\mathbb{Q}$ to the reals $\mathbb{R}$ has a unique continuous extension to a function from $\mathbb{R}$ to $\mathbb{R}$. It is, perhaps, a more interesting result since we did not have to assume continuity of the function on E, only monotonicity.

The main pragmatic reason why cpos with monotonic/continuous functions are used so frequently in theoretical computer science is that they yield fixed point theorems. A fixed point of a function $f : X \to X$ is $x \in X$ such that $f(x) = x$. It is very easy to construct sets X and functions $f : X \to X$ that have no fixed point, for example $x \mapsto x + 1$ over the natural numbers. Fixed points are needed to define the semantics of recursive programs – see, for example the case of $\underline{rec}$ at the end of Chapter 1.

There is essentially one theorem here in different guises depending on the context. It is usually known as Tarski's theorem though Tarski was by no means the first to discover a version of the result, one having been used by Dedekind in the 1890's to give a proof of the Schröder-Bernstein theorem[5]. And the most general version: every monotone function $f : P \to P$ on a cpo P has a least fixed point is due, we believe, to David Park. The proof of this last version requires advanced mathematics and so is delayed to Section 2.4.

Here we give two slightly weaker versions which have elementary (and quite different) proofs.

[5]The Schröder-Bernstein theorem states that if X and Y are sets such that there exist 1-1 functions $f : X \to Y$ and $g : Y \to X$ then there is a bijection (1-1 and onto) from X to Y.

2.2.5 THEOREM
If P is a cpo and $f : P \to P$ is continuous then f has a least fixed point (i.e., $x \in P$ such that $f(x) = x$ and, if $f(y) = y$, then $x \leq y$).

PROOF
P has a least element $\bot$. Since $\bot$ is the least element we have $\bot \leq f(\bot)$. Now suppose $f^n(\bot) \leq f^{n+1}(\bot)$. It follows immediately from the fact that f is monotone that $f^{n+1}(\bot) \leq f^{n+2}(\bot)$. We have thus proved by induction that $\langle f^n(\bot) \mid n \in \mathbb{N}\rangle$ is an increasing sequence in P.

But f is continuous, so

$$f(\bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}) = \bigsqcup\{f^{n+1}(\bot) \mid n \in \mathbb{N}\}\ .$$

It is obvious that if Δ is any directed set then $\bigsqcup \Delta = \bigsqcup(\Delta \cup \{\bot\})$. (Clearly any upper bound of one set is an upper bound of the other!) Therefore $\bigsqcup\{f^{n+1}(\bot) \mid n \in \mathbb{N}\} = \bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}$, and it follows that $x = \bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}$ is a fixed point of f.

Now, if y is any other fixed point then clearly $\bot \leq y$. Suppose $f^n(\bot) \leq y$. Then

$$f^{n+1}(\bot) \leq f(y) = y$$

as f is monotone and y is a fixed point. It follows that $f^n(\bot) \leq y$ for all n and hence that y is an upper bound for $\{f^n(\bot) \mid n \in \mathbb{N}\}$. But x is the least upper bound, so $x \leq y$. ∎

2.2.6 THEOREM
If P is a complete lattice and $f : P \to P$ is monotonic then f has a least fixed point.

PROOF
Let $x = \bigsqcap S$, where $S = \{y \in P \mid f(y) \leq y\}$. This exists as all subsets of a complete lattice have greatest lower bounds. Now, whenever $y \in S$ we have $x \leq y$ and thus $f(x) \leq f(y) \leq y$, proving that $f(x)$ is a lower bound for S and hence $f(x) \leq x$. Of course this just says $x \in S$; but also note that $f(x) \leq x \Rightarrow f(f(x)) \leq f(x)$ as f is monotone, which implies $f(x) \in S$. But $x \leq y$ for all $y \in S$ which proves $f(x) \geq x$. So we may conclude $f(x) = x$. And since $y \in S$ for any fixed point y, it is clear that x is the least fixed point. ∎

The least fixed point of a function f is often denoted μf.

EXAMPLE: LOOPS
We illustrate the use of this type of result by developing a semantics for a language akin to that of Section 1.1, but including loops and conditionals. The syntax is as

follows.

SYNTACTIC DOMAINS

x	$\in$	Var	variables
N	$\in$	Num	numerals
E	$\in$	Exp	expressions
Γ	$\in$	Cmd	commands

$$E ::= x \mid N \mid E_1 + E_2 \mid E_1 - E_2 \mid \ldots \mid \underline{true} \mid E_1 = E_2 \mid E_1 < E_2 \mid \ldots$$

$$\Gamma ::= \underline{skip} \mid x := E \mid \Gamma_1;\ \Gamma_2 \mid \underline{if}\ \ E\ \ \underline{then}\ \ \Gamma_1\ \ \underline{else}\ \ \Gamma_2 \mid \underline{while}\ E\ \underline{do}\ \Gamma \quad .$$

$\underline{skip}$ is a command whose execution has no effect.

Expressions now take values in the set $V = \{error\} \cup \mathbb{T} \cup Z$, where $\mathbb{T} = \{true, false\}$ and $\int$ is the set of all integers. The most obvious extension of the semantic domain used in Section 1.1, namely $S \to S$, where $S = Var \to V$ does not work, because we will require a partial order to take fixed points and this domain has no obvious order at all. The solution is to use the extended function space $S \to (S \cup \{\bot\})$, where $\bot \notin S$. The order on this space is simple:

$$\phi \leq \psi \equiv \forall \sigma \in S. \phi(\sigma) \neq \bot \Rightarrow \phi(\sigma) = \psi(\sigma)\ .$$

In other words, ψ extends ϕ if it agrees with ϕ whenever ϕ produces a 'proper' state, but may turn some of ϕ's $\bot$s into proper states. The least element of this space is the one ($\bot^*$, say) that sends every state to $\bot$. And if Δ is a directed set then its least upper bound is ϕ, where

$$\phi(\sigma) = \begin{cases} \bot & \text{if } \psi(\sigma) = \bot \text{ for all } \psi \in \Delta \\ \sigma' & \text{if there is some } \psi \in \Delta \text{ with } \psi(\sigma) = \sigma' \neq \bot. \end{cases}$$

This definition is never ambiguous, since if $\psi, \psi' \in \Delta$ have $\psi(\sigma) \neq \bot$ and $\psi'(\sigma) \neq \bot$ then we know (as Δ is directed) that there is $\psi'' \in \Delta$ with $\psi, \psi' \leq \psi''$. But the definition of $\leq$ then means that $\psi(\sigma) = \psi''(\sigma) = \psi'(\sigma)$.

In fact the new value $\bot$ has a very real interpretation: $\phi(\sigma) = \bot$ means that the program does not terminate normally when started in state σ. Notice that this sort of behaviour can and does arise in programs with $\underline{while}$ loops, and that we have no way of modelling it in the old model $S \to S$. (All programs in the language of Section 1.1 terminate.) Thus the needs of our fixed point theorems actually match well with reality!

If we were to extend our language so that it became possible that an expression evaluation did not terminate then we would have to include a $\perp$ value in V. In general the shape of a domain needs to be designed very carefully so that it is sufficiently powerful to capture the essence of a language without introducing any unwanted values that get in the way. These are issues which we will be returning to study in detail later on, particularly in Chapters 9 and 10.

The semantic function $\mathcal{E} : Exp \to S \to V$ is very similar to those seen in the last chapter and its definition is omitted. The semantic function for commands is now

$$\mathcal{C} : Cmd \to S \to (S \cup \{\perp\}) .$$

With the exception of the clause for loops, the semantic clauses are straightforward extensions of those already seen in Chapter 1. Perhaps the main points to note are that the evaluation of an expression to an error value (possible both because of division by 0 and because one might find a number where a boolean was expected, or *vice versa*) now produces the undefined 'state' $\perp$[6], and that sequential composition now has to take account of the fact that the first command may go wrong.

$$\begin{array}{rcll}
\mathcal{C}[\![\underline{skip}]\!]\sigma & = & \sigma & \\
\mathcal{C}[\![x := E]\!]\sigma & = & \sigma[\mathcal{E}[\![E]\!]\sigma/x] & \text{if } \mathcal{E}[\![E]\!]\sigma \neq error \\
 & = & \perp & \text{if } \mathcal{E}[\![E]\!]\sigma = error \\
\mathcal{C}[\![\Gamma_1;\ \Gamma_2]\!]\sigma & = & \mathcal{C}[\![\Gamma_2]\!](\mathcal{C}[\![\Gamma_1]\!]\sigma) & \text{if } \mathcal{C}[\![\Gamma_1]\!]\sigma \in S \\
 & = & \perp & \text{if } \mathcal{C}[\![\Gamma_1]\!]\sigma = \perp \\
\mathcal{C}[\![\underline{if}\ E\ \underline{then}\ \Gamma_1\ \underline{else}\ \Gamma_2]\!]\sigma & = & \mathcal{C}[\![\Gamma_1]\!]\sigma & \text{if } \mathcal{E}[\![E]\!]\sigma = true \\
 & = & \mathcal{C}[\![\Gamma_2]\!]\sigma & \text{if } \mathcal{E}[\![E]\!]\sigma = false \\
 & = & \perp & \text{otherwise}
\end{array}$$

Of course the interesting case is that of loops. Intuitively one would expect that if $\Gamma = \underline{while}\ E\ \underline{do}\ \Gamma_1$ then the two programs

$$\Gamma \quad \text{and} \quad \underline{if}\ E\ \underline{then}\ \Gamma_1;\ \Gamma\ \underline{else}\ \underline{skip} \qquad (*)$$

would be equivalent (i.e., have the same semantic value). Assuming that this holds, we can deduce from the clauses above that $\phi = \mathcal{C}[\![\Gamma]\!]$ is a fixed point of the functional

[6]This is a fairly harsh, though elegant, treatment of error handling. For less severe ones see Chapters 9 and 10.

$F : (S \to (S \cup \{\bot\})) \to (S \to (S \cup \{\bot\}))$ defined

$$F(\phi)\sigma = \begin{cases} \bot & \text{if } \mathcal{E}[\![E]\!]\sigma \notin \{true, false\} \text{ or} \\ & \mathcal{E}[\![E]\!]\sigma = true \text{ and } \mathcal{C}[\![\Gamma_1]\!]\sigma = \bot \\ \phi(\mathcal{C}[\![\Gamma_1]\!]\sigma) & \text{if } \mathcal{E}[\![E]\!]\sigma = true \text{ and } \mathcal{C}[\![\Gamma_1]\!]\sigma \neq \bot \\ \sigma & \text{if } \mathcal{E}[\![E]\!]\sigma = false \end{cases}$$

(This is simply the function which calculates the semantics of the program $\underline{if}\ E\ \underline{then}\ \Gamma_1;\ \Gamma\ \underline{else}\ skip$ in terms of those of Γ.)

The crucial fact to observe about F is that, for a given σ, either $F(\phi)\sigma$ is independent of ϕ or there is a fixed σ' $(= \mathcal{C}[\![\Gamma_1]\!]\sigma)$ such that $F(\phi)\sigma = \phi(\sigma')$ for all ϕ. Having observed this it is easy to see that F is monotone; for if $\phi \leq \psi$ then if $F(\phi)\sigma \neq \bot$ then in either case we get $F(\phi)\sigma = F(\psi)\sigma$.

We need to show that F is continuous; in other words that if $\Delta \subseteq (S \to (S \cup \{\bot\}))$ is directed then $F(\bigsqcup \Delta) = \bigsqcup\{F(\phi) \mid \phi \in \Delta\}$. By the principle of extensionality (discussed in the last chapter) it is sufficient to prove

$$F(\bigsqcup \Delta)\sigma = (\bigsqcup\{F(\phi) \mid \phi \in \Delta\})\sigma \quad \text{for all } \sigma \in S\,.$$

As F is monotone we know that the right hand side exists and that (by Lemma 2.2.1) if the right hand side is not $\bot$ then the two are equal. So suppose the left hand side is not $\bot$. Then if the first case above holds and $F(\phi)\sigma$ is independent of ϕ the result is trivial. And if the second case holds (and σ' is as above) then we know that the left hand side is $(\bigsqcup \Delta)\sigma'$. But the definition of $\bigsqcup \Delta$ then means that there is some $\phi \in \Delta$ with $\phi(\sigma')$ equal to $(\bigsqcup \Delta)(\sigma')$ – which gives us exactly what we want.

We can conclude that F has a least fixed point given by $\bigsqcup\{F^n(\bot^*) \mid n \in \mathbb{N}\}$. And of course we define $\mathcal{C}[\![\underline{while}\ E\ \underline{do}\ \Gamma_1]\!]$ to be this fixed point.

This definition seems very abstract and unconnected with the reality of the execution of $\underline{while}$ loops. But in fact it is not. If we think again about how we constructed F out of the equivalence $(*)$ it should be clear that $F^n(\bot^*)$ is the semantic value of a program which behaves correctly if the loop terminates after up to $n-1$ executions of Γ_1 before giving up and returning $\bot$. And the limit allows any number of executions, but produces $\bot$ if the loop does not terminate – precisely the answer we would expect.

If we consider the loop $\underline{while}\ x \neq 0\ \underline{do}\ x := x - 1$ then

$$F^n(\bot^*)\sigma = \begin{cases} \sigma[0/x] & \text{if } 0 \leq \sigma[\![x]\!] < n \\ \bot & \text{otherwise} \end{cases}$$

and the least fixed point $\phi = \bigsqcup\{F^n(\bot^*) \mid n \in \mathbb{N}\}$ is the function one might expect

$$\phi(\sigma) = \begin{cases} \sigma[0/x] & \text{if } 0 \leq \sigma[\![x]\!] \\ \bot & \text{otherwise.} \end{cases}$$

In other words, this loop sets x to 0 if it was initially non-negative, and does not terminate otherwise.

Note that any loop that never terminates on any initial state, such as

while *true* *do* *skip*

is mapped by the semantic function to the bottom element $\bot^*$ of its target. ∎

The way partial orders were used in the above example is fairly typical of their use in programming language semantics. The bottom element $\bot^*$ represents an undefined or nonterminating program, and increasing in the partial order means that the program becomes more defined or more likely to terminate. Notice also that the order we used has no top element; philosophically this perhaps means that there is no perfect program.

Since fixed points play such an important rôle in programming language semantics, it is often necessary to prove things about them. Frequently we can use the cpo structure of the underlying mathematical model to help us. One obvious but useful trick applies when we have a function F with a *unique* fixed point (i.e., only one): if x is any value for which it can be shown $x = F(x)$, then $x = \mu\, F$. And any function whose least fixed point can be shown to be maximal in its cpo – which in the example above is equivalent to mapping every state to a proper (non-$\bot$) result – obviously has only one.

There are a number of related proof techniques grouped under the term *fixed point induction*. We give one in this section, which requires the following definition.

Definition
A property R of a cpo P is said to be *inclusive* if, whenever, Δ is a directed set with $R(x)$ true for all $x \in \Delta$, then R is true of $\bigsqcup \Delta$. ∎

Thus, for example, $R(x) \equiv x \leq y$ is always an inclusive property, while $R'(x) \equiv x \not\geq y$ is inclusive precisely when y is po-finite.

If we think of the truth values as being ordered, with $true < false$, then a property R is inclusive if, and only if, $R(\bigsqcup \Delta) \leq \bigsqcup\{R(x) \mid x \in \Delta\}$ for all directed $\Delta \subseteq P$. (Over this two point order there is no problem about the existence of the least upper bounds!) For a monotonic R (which, in this context, means $R(x) \wedge y \leq x \Rightarrow R(y)$) it is *equivalent* to continuity.

FIXED POINT INDUCTION
If P is a cpo, R an inclusive property of P and $f : P \to P$ is continuous, then if $R(\bot)$ holds and

$$\forall\, x.R(x) \Rightarrow R(f(x))$$

we may infer $R(\mu\, f)$. ∎

The validity of this principle is obvious, given the proof of Theorem 2.2.5, since the chain constructed there is clearly a directed set all of whose elements have R true. It can sometimes be inconvenient that the property we are proving must be true at $\bot$ – for $\bot$ is often (as in the example above) rather a badly-behaved element of a semantic model. Some methods for getting round this are described in Exercises 2.2.6 and 2.4.8.

2.2.1 Other conditions on functions

Aside from monotonicity and continuity, there are several other conditions on functions that are frequently encountered in computer science. We conclude this section by describing some.

Definition
If the partial orders P and Q have least elements ($\bot_P$ and $\bot_Q$ respectively), a function $f : P \to Q$ is said to be *strict* if $f(\bot_P) = \bot_Q$. ∎

Essentially, a strict function is one that cannot 'recover' from an argument which is 'broken'. We will see later, in chapters on denotational semantics, how useful this idea can be. Strictness, and analysis of strictness, are extremely important subjects in functional programming.

Notice that the least fixed point of a strict function $f : P \to P$ is always $\bot$ – so that strict functions are not suitable for defining the semantics of recursive constructs like *while* above.

Definition
If the partial orders P and Q both have pairwise greatest lower bounds $x \sqcap y$ for all x and y, a function $f : P \to Q$ is said to be *distributive* if, for all x and y,

$$f(x \sqcap y) = f(x) \sqcap f(y)\,.$$ ∎

Any distributive function is monotone for, if $x \leq y$, then

$$f(x) = f(x \sqcap y) = f(x) \sqcap f(y)\,.$$

And if P is a linear order then any monotone $f : P \to Q$ is distributive. There are, however, monotone functions that are not distributive.

Where arbitrary nonempty sets have greatest lower bounds (for example consistently complete cpos) this can be extended to *infinite distributivity* in the obvious way.

Distributive functions are useful for a number of technical reasons and also when proving programs meet certain sorts of specifications. If, in an order P with arbitrary nonempty greatest lower bounds, we can find a set X such that every $x \in P$ equals $\bigsqcap \{y \in X \mid y \geq x\}$, then an infinitely distributive function of P is determined by its restriction to X. You should notice the analogy with the way continuous functions on algebraic cpos are determined by their values on the po-finite elements.

Exercises

2.2.1 Classify the following functions from the complete lattice $[0, 1]$ (the closed interval with its usual order) to itself. (In other words, which are monotone and which are continuous, both in the partial order sense?)

(a) The identity function $x \mapsto x$.
(b) $x < \frac{1}{2} \Rightarrow x \mapsto 0$, $x \geq \frac{1}{2} \Rightarrow x \mapsto 1$.
(c) $x \leq \frac{1}{2} \Rightarrow x \mapsto 0$, $x > \frac{1}{2} \Rightarrow x \mapsto 1$.
(d) $x \mapsto 1 - x$

2.2.2 Prove that every monotonic function from Q to P (P and Q being cpos) is continuous if Q is finite. Find a non-finite Q of which this is true (for all possible P).

*2.2.3 Let X be any set and $\mathcal{R}$ be the set of relations on X (i.e., subsets of $X \times X$) ordered by reverse inclusion ($R \leq S \Leftrightarrow R \supseteq S$). Show this is a complete lattice. Let S be any element of $\mathcal{R}$. Define a function F from $\mathcal{R}$ to $\mathcal{R}$ by defining $F(R)$ to be R'; where

$$x\, R'\, x' \Leftrightarrow (\forall\, y.x\, S\, y \Rightarrow \exists\, y'.x'\, S\, y' \wedge y\, R\, y') \wedge (\forall\, y'.x'\, S\, y' \Rightarrow \exists\, y.x\, S\, y \wedge y\, R\, y')$$

Show that F is monotone, and is continuous if $\{y \mid x\, S\, y\}$ is finite for all x. We may deduce that F has a least fixed point $\mu\, F$.
Use fixed point induction to show that $\mu\, F$ is symmetric (i.e., $x\ \mu\, F\, y \Leftrightarrow y\ \mu\, F\, x$). (The existence of fixed points of this type – known as *bisimulations* – underpins much of the theory of concurrency, in particular that of CCS [].)

2.2.4 Suppose f, g are both continuous functions from the complete partial order P to itself, and that $f \circ g = g \circ f$. Prove that $\mu(f \circ g)$ is a fixed point of both

f and g, and is the least point which is the fixed point of both. Show by means of example that in general two continuous functions with a common fixed point *need not* have a least common fixed point.

2.2.5 Prove that the following programs are all equivalent in the example language given above (for any E and Γ).

(a) $\underline{while}\ E\ \underline{do}\ \Gamma$

(b) $\underline{while}\ E\ \underline{do}\ (\underline{if}\ \ E\ \ \underline{then}\ \ \Gamma\ \ \underline{else}\ \ \underline{skip})$

(c) $\underline{while}\ E\ \underline{do}\ (\Gamma;\ (\underline{if}\ \ E\ \ \underline{then}\ \ \Gamma\ \ \underline{else}\ \ \underline{skip}))$.

) *[Hint: Prove (a) and (b) equal by demonstrating that the functions of which you are taking least fixed points are the same. Prove (a) and (c) equal by showing that, if F and G are respectively the functions whose fixed points we want in (a) and (c) respectively, then F^2 and G have the same set of fixed points. (If you compare F^2 and G carefully you will find that they are not identical.)]*

2.2.6 Suppose that, for all n, R_n is an inclusive predicate on the cpo P, that $\forall\, x.R_{n+1}(x) \Rightarrow R_n(x)$ and that $R_0(\bot)$ holds. Suppose that

$$\forall\, x\, \forall\, n.R_n(x) \Rightarrow R_{n+1}(f(x))\ ,$$

where $f : P \to P$ is continuous. Show that $\forall\, n.R_n(\mu\, f)$.
Construct appropriate R_n to prove in the example semantics above that the program

$$\underline{while}\ x < 0\ \underline{do}\ x\ :=\ x + 1$$

always terminates successfully when x is initialised with an integer.

2.3 Product spaces

A product space is one where objects have a number of components, each from some smaller space. The simplest product space is the cartesian product $P \times Q$ of the sets P and Q – consisting of all pairs $\langle x, y\rangle$ where $x \in P$ and $y \in Q$. If P and Q are both partially ordered then $P \times Q$ can be given an order in several different ways. The standard order (which we shall always use except in a few exercises) is defined

$$\langle x, y\rangle \leq \langle x', y'\rangle \Leftrightarrow x \leq x' \wedge y \leq y'\ .$$

It is sometimes called the componentwise order because one pair is less than another if and only if each of its components is less than the corresponding component of the other pair.

We will frequently want to use product spaces where there are many or even infinitely many components. The elements of such a space can be thought of as vectors or tuples with the appropriate number of components, or alternatively as functions from an indexing set to the union of the sets from which the components are taken. Thus for ordered pairs in $P \times Q$ we would choose an indexing set with two elements (say $\{1, 2\}$) and think of pairs as being functions from $\{1, 2\}$ to $P \cup Q$, where the image of 1 is in P, and the image of 2 is in Q.

In fact we have already used a product space in the example semantics seen in the last section. $S \to (S \cup \{\bot\})$ is the product of S copies of the flat domain on S (see Exercise 2.1.9 (a)) – notice that the order we defined earlier is exactly this one.

A typical example of an infinite product space is P^ω, the infinite sequences of elements of P; this is identified with the functions from $\mathbb{N}$ to P. Another is the $\{0, 1\}^X$ for any set X – this is naturally isomorphic to the powerset $\mathcal{P}(X)$ (Example (b) of Section 2.1) since we can identify a tuple with the set of all components with value 1, and the product order becomes the usual subset order $\subseteq$.

Note that we have used the 'to-the-power-of' notation X^Y to denote the function space $Y \to X$ where this means the product of Y copies of X. This is justified by the observation that if X has n elements and Y has m elements, then the product space X^Y has n^m elements. A more general notation is

$$\prod_{\lambda \in \Lambda} P_\lambda$$

which denotes the product space with indexing set Λ whose λth component is picked from P_λ.

It is often convenient to use vector notation for product spaces (especially large ones): a typical element of P^Λ will be underlined, $\underline{x}$ and its λ component is x_λ. In 'tuple' notation we then have $\underline{x} = \langle x_\lambda \mid \lambda \in \Lambda \rangle$.

The partial order on an arbitrary product space is a straightforward extension of the one above:

$$\langle x_\lambda \mid \lambda \in \Lambda \rangle \leq \langle y_\lambda \mid \lambda \in \Lambda \rangle \Leftrightarrow \forall \lambda \,.\, x_\lambda \leq y_\lambda \,.$$

With the exception of linear ordering, which is *not* preserved, almost all of the properties identified in the first section are preserved by product. This is shown by the following result.

2.3.1 THEOREM
If each of P_λ has (the same) one of the following properties, then $\prod_{\lambda \in \Lambda} P_\lambda$ also has that property.

(a) complete

(b) lattice

(c) complete lattice

(d) consistently complete cpo

(e) algebraic cpo

Additionally, if each P_λ is an ω-algebraic and Λ is countable (i.e., finite or countably infinite) then $\prod_{\lambda\in\Lambda} P_\lambda$ is ω-algebraic.

PROOF
Suppose that $X \subseteq \prod_{\lambda\in\Lambda} P_\lambda$ and that, for each $\lambda \in \Lambda$, the set $X_\lambda = \{x_\lambda \mid \underline{x} \in X\}$ has a least upper bound y_λ. Then $\underline{y} = \langle y_\lambda \mid \lambda \in \Lambda\rangle$ is an upper bound for X – since if $\underline{x} \in X$ we know $x_\lambda \leq y_\lambda$ for all λ – and if $\underline{z}$ is any other upper bound then z_λ is an upper bound for X_λ for all λ, and so $\underline{y} \leq \underline{z}$. In other words $\underline{y}$ is the least upper bound of X.

The same componentwise construction clearly works for greatest lower bounds as well.

These observations prove (a), (b), (c) and (d) since if $X \subseteq \prod_{\lambda\in\Lambda} P_\lambda$ is (a) directed, (b) finite or (d) po-consistent then it is clear that all X_λ also have the corresponding property.

To establish (e) we claim that if E_λ is the set of po-finite elements of P_λ and $\bot_\lambda$ is the least element of P_λ then

$$E = \{\underline{x} \in \textstyle\prod_{\lambda\in\Lambda} E_\lambda \mid \{\lambda \mid x_\lambda \neq \bot_\lambda\} \text{ is finite}\}$$

is the set of po-finite elements of the product space. For if $\underline{x}$ is po-finite then clearly each component must be po-finite (or we could approximate it keeping all other components fixed), and the number of non-$\bot$ components is finite since the directed set

$$\{\underline{x}\restriction F \mid F \subseteq^{\text{fin}} \Lambda\}$$

has limit $\underline{x}$, where

$$(\underline{x}\restriction F)_\lambda = \begin{cases} x_\lambda & \text{if } \lambda \in F \\ \bot_\lambda & \text{if } \lambda \notin F \end{cases}$$

And if $\underline{x} \in E$ with set F of non-bottom components, and Δ is directed with $\bigsqcup \Delta \geq \underline{x}$, then for all $\lambda \in F$ we can find $\underline{y} \in \Delta$ with $y_\lambda \geq x_\lambda$. Since F is finite and Δ directed there is $\underline{y}^* \in \Delta$ which is greater than all these y. Clearly $\underline{y}^* \geq \underline{x}$. This establishes our claim.

Now if $x \in \prod_{\lambda\in\Lambda} P_\lambda$ and each P_λ is algebraic then, from the formula for upper bounds computed above, it is clear that

$$\bigsqcup\{\underline{e} \in E \mid \underline{e} \leq \underline{x}\} = \underline{x}\,,$$

establishing (e). The final part comes from the fact that E is countable if all E_λ are and Λ is countable – there are then countably many finite subsets of Λ and countably many finite elements for each. ∎

Clearly a function that has more than one argument drawn from partial orders can be thought of as a function with a single argument drawn from some product space. This identification defines what it means for such a function to be monotone or continuous. The following result shows that where there are two (and hence, inductively, any finite number of) arguments, there is a straightforward test for these properties.

LEMMA 2.3.2
Suppose P, Q, R are partial orders and $f : P \times Q \to R$. Then we may define two functions $f'_x : Q \to R$ and $f''_y : P \to R$ for any $x \in P$ and $y \in Q$ by

$$f'_x(y) = f''_y(x) = f(x, y)\,.$$

The continuity and monotonicity of f is related to that of f'_x and f''_y as follows.

(a) f is monotone if and only if all f'_x and f''_y are (i.e., if f is monotone in each argument separately).

(b) f is continuous if and only if all f'_x and f''_y are (i.e., if f is continuous in each argument separately).

PROOF
The fact that f being monotone implies all f'_x and f''_y are is trivial, so suppose that all f'_x and f''_y are monotone and that $(x, y) \leq (x', y')$. Then $x \leq x'$ and $y \leq y'$, so

$$\begin{array}{rcll} f(x,y) & = & f'_x(y) & \\ & \leq & f'_x(y') & \text{as } f'_x \text{ monotone} \\ & = & f''_{y'}(x) & \\ & \leq & f''_{y'}(x') & \text{as } f''_{y'} \text{ monotone} \\ & = & f(x',y') & \end{array}$$

This proves part (a).

If Δ_1, Δ_2 are directed subsets of P and Q respectively then $\{(x, y) \mid x \in \Delta_1\}$ and $\{(x, y) \mid y \in \Delta_2\}$ are directed in $P \times Q$ with respective limits $(\bigsqcup \Delta_1, y)$ and

$(x, \bigsqcup \Delta_2)$. It follows immediately that if f is continuous then all f'_x and f'_y are. So suppose all f'_x and f''_y are continuous and that $\Delta \subseteq P \times Q$ is directed. Since f is monotone (by part (a)) we know by Lemma 2.2.1 that

$$\bigsqcup\{f(x,y) \mid (x,y) \in \Delta\} \leq f(\bigsqcup \Delta)$$

so to prove continuity it will be sufficient to prove the reverse. Define

$$\Delta_1 = \{x \mid (x,y) \in \Delta\} \quad \text{and} \quad \Delta_2 = \{y \mid (x,y) \in \Delta\}\,.$$

We know already from Theorem 2.3.1 that $\bigsqcup \Delta = (\bigsqcup \Delta_1, \bigsqcup \Delta_2)$. It is clear that $\Delta^* = \Delta_1 \times \Delta_2$ is directed with the same limit as Δ, and that $\Delta \subseteq \Delta^*$. If $(x,y) \in \Delta^*$ then there are x', y' such that (x,y') and (x',y) are in Δ and hence by directedness, there is $(x'',y'') \in \Delta$ with $(x,y) \leq (x'',y'')$ and so by monotonicity of f:

$$\bigsqcup\{f(x,y) \mid (x,y) \in \Delta^*\} \leq \bigsqcup\{f(x,y) \mid (x,y) \in \Delta\}$$

and it is easy to see that reverse inequality holds, so that in fact the terms are equal. But we then have

$$\begin{array}{rcll}
\bigsqcup\{f(x,y) \mid (x,y) \in \Delta\} & = & \bigsqcup\{f(x,y) \mid (x,y) \in \Delta^*\} & \\
& = & \bigsqcup\{f(x,y) \mid y \in \Delta_2 \wedge x \in \Delta_1\} & \\
& = & \bigsqcup\{\bigsqcup\{f(x,y) \mid y \in \Delta_2\} \mid x \in \Delta_1\} & (1) \\
& = & \bigsqcup\{f(x, \bigsqcup \Delta_2)\} \mid x \in \Delta_1\} & (2) \\
& = & f(\bigsqcup \Delta_1, \bigsqcup \Delta_2) & (3) \\
& = & f(\bigsqcup \Delta^*) &
\end{array}$$

Line (1) is an application of the result established in Exercise 2.1.4. Lines (2) and (3) follow respectively from the continuity of f'_x and $f''_{\bigsqcup \Delta_2}$. This completes the proof. ∎

This result does not generalise to the case of infinite product spaces (see Exercise 2.3.5). However the corresponding result for functions *to* product spaces is not restricted in this way. All parts of the following result are easy to prove.

LEMMA 2.3.3
The function $f : P \to \prod_{\lambda \in \Lambda} Q_\lambda$ is (a) monotone or (b) continuous if and only if all of the component functions $f_\lambda : P \to Q_\lambda$ are (a) monotone or (b) continuous, where $f_\lambda(x) = f(x)_\lambda$. ∎

2.3.1 Function Spaces

We have already seen how product spaces can be thought of as the sets of functions from some indexing set to a partial order (or several partial orders). These spaces

pay no attention to any partial order structure that may be present in the indexing set, but we have already met two classes of functions that do, namely the monotonic ones and the continuous ones. In this subsection we will examine the partial orders these restricted function spaces inherit from the full product space. Notice that the partial order can be written

$$f \leq g \Leftrightarrow \forall\, x.f(x) \leq g(x)\,.$$

Definitions
If P and Q are two cpos, define $P \xrightarrow{m} Q$ and $P \xrightarrow{c} Q$ respectively to be the sets of all monotone and continuous functions from P to Q. ∎

These constructs are not quite as good at preserving the various partial order properties as the full product described earlier, but one does have the following result (among others).

LEMMA 2.3.4
Both of the spaces $P \xrightarrow{m} Q$ and $P \xrightarrow{c} Q$ are cpos if P and Q are.

PROOF
The least upper bound of a directed set Φ is, in both cases, the same as that in the full product space, or in other words

$$(\bigsqcup \Phi)(x) = \bigsqcup\{f(x) \mid f \in \Phi\}\,.$$

To prove the result we must simply prove that if all elements of Φ are respectively monotone or continuous then so is $\bigsqcup \Phi$.

The monotone case is very straightforward since if $x \leq y$ then $f(x) \leq f(y)$ for all $f \in \Phi$ so certainly

$$\bigsqcup\{f(x) \mid f \in \Phi\} \leq \bigsqcup\{f(y) \mid f \in \Phi\}\,.$$

For the continuous case, suppose $\Delta \subseteq P$ is directed. Then we have

$$\begin{array}{rcll} (\bigsqcup \Phi)(\bigsqcup \Delta) & = & \bigsqcup\{f(\bigsqcup \Delta) \mid f \in \Phi\} & \\ & = & \bigsqcup\{\bigsqcup\{f(x) \mid x \in \Delta\} \mid f \in \Phi\} & (1) \\ & = & \bigsqcup\{\bigsqcup\{f(x) \mid f \in \Phi\} \mid x \in \Delta\} & (2) \\ & = & \bigsqcup\{(\bigsqcup \Phi)(x) \mid x \in \Delta\} & (3) \end{array}$$

where line (1) is by continuity of the fs, line (2) is a general property of partial orders (see Exercise 2.1.4) and line (3) is by definition of $\bigsqcup \Phi$. ∎

For more similar results and counterexamples to other properties see the exercises at the end of this section.

The final result of this chapter is important in many cases of denotational semantics. It shows that the least fixed point operator is continuous on the space of continuous functions.

LEMMA 2.3.5
If P is a cpo then the least fixed point operator $\mu : (P \xrightarrow{c} P) \to P$ (see Theorem 2.2.5) is continuous.

PROOF
Suppose $\Phi \subseteq P \xrightarrow{c} P$ is directed. We know from the proof of Theorem 2.2.5 that, for any continuous f,

$$\mu f = \bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\} .$$

Claim that for all n we have

$$(\bigsqcup \Phi)^n(\bot) = \bigsqcup\{f^n(\bot) \mid f \in \Phi\} .$$

This is certainly true when $n = 0$ (both sides are $\bot$), so suppose it holds for n. Then by induction, the definition of $\bigsqcup \Phi$ and the continuity of all $g \in \Phi$ we have

$$(\bigsqcup \Phi)^{n+1}(\bot) = \bigsqcup\{g(f^n(\bot)) \mid g, f \in \Phi\} .$$

But since all $f, g \in \Phi$ have $k \in \Phi$ with $k \geq f, g$ this is in turn equal to

$$\bigsqcup\{k^{n+1}(\bot) \mid k \in \Phi\}$$

as claimed.

Given this, we have

$$\begin{array}{rcl} \mu(\bigsqcup \Phi) & = & \bigsqcup\{(\bigsqcup \Phi)^n(\bot) \mid n \in \mathbb{N}\} \\ & = & \bigsqcup\{\bigsqcup\{f^n(\bot) \mid f \in \Phi\} \mid n \in \mathbb{N}\} \\ & = & \bigsqcup\{\bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\} \mid f \in \Phi\} \\ & = & \bigsqcup\{\mu f \mid f \in \Phi\} \end{array}$$

which proves the Lemma. ∎

Exercises

2.3.1 Suppose P, Q and R are cpos. Prove that the following pair of partial orders are order isomorphic when $\to$ is interpreted as (i) the full function space (equivalently product space) operator (ii) $\stackrel{m}{\to}$ and (iii) $\stackrel{c}{\to}$.

$$(P \times Q) \to R \qquad \text{and} \qquad P \to (Q \to R)$$

This is the identity underlying Currying (see Chapter 1); to prove these isomorphisms you should show that the maps

$$\begin{array}{rl} curry: & ((P \times Q) \to R) \to (P \to (Q \to R)) \quad \text{and} \\ uncurry: & (P \to (Q \to R)) \to ((P \times Q) \to R) \end{array}$$

are (well-defined) order isomorphisms in each of the three cases.

2.3.2 If X is any partial order, show that X^ω and $(X^\omega)^\omega$ are order isomorphic.

2.3.3 Suppose P and Q are cpos. Show that if Q is consistently complete then so are $P \stackrel{m}{\to} Q$ and $P \stackrel{c}{\to} Q$.

2.3.4 Let P be the unit interval $[0, 1]$ and Q be the two point cpo $\{\bot, \top\}$. Show that $P \stackrel{m}{\to} Q$ is algebraic but that $P \stackrel{c}{\to} Q$ is not.

2.3.5 Let X be the two point linear order. Find functions $f, g : X^\omega \to X$ which are *constant* in each argument separately (i.e, changing one component of the infinite tuple of arguments has no effect) but such that (i) f is not monotone and (ii) g is monotone but not continuous. This shows that Lemma 2.3.2 does not extend to infinite product spaces.

2.4 Advanced topics

Relying on only basic set theory we have already developed all of the partial order theory necessary for the rest of the book. There are however a number of other insights which can be gained if we allow ourselves to assume more advanced mathematical knowledge (in particular ordinal numbers and topology). This final section is included so that readers with a suitable background can appreciate just a few of these.

Recall that we introduced directed sets as our generalised convergent sequences rather than using increasing ω-sequences. The canonical example of an order which is ω-complete (i.e., all such sequences have limits) but not complete is ω_1, the set of all countable ordinals. This set is not complete since it has no greatest element even though it is linearly ordered, but is ω-complete since the union of any countable set of countable ordinals is a countable ordinal. A more elementary example is given by the set of all finite and countably infinite subsets of $\mathbb{R}$ (under

the subset order). It is ω-complete since the union of a countable set of countable sets is countable.

Very often ω-completeness is all we actually need (witness the proof of Theorem 2.2.5), but full completeness combines much better with the other properties we have introduced. And one can plausibly argue that the reason ω-completeness is the definition used in real analysis and metric spaces is simply that it is all that is needed there: beefing it up with something like directed sets would add complexity without changing the meaning of the definition at all.

A related question that is interesting is whether completeness is the same as *chain* completeness (i.e., every linearly ordered subset has a least upper bound). Completeness clearly implies chain completeness, but the reverse is by no means obvious. In fact they are the same, as is demonstrated by a sequence of exercises at the end of this section.

We stated in Section 2.2 that every monotone function on a cpo has a least fixed point. Recall that we proved two subcases there. The proof of this result is a straightforward use of ordinals extending the proof of 2.2.5.

THEOREM 2.4.1
Every monotone function f from a complete partial order P to itself has a least fixed point.

PROOF
We will define $f^\alpha(\bot)$ for all ordinals α. $f^0(\bot) = \bot$, and if $f^\alpha(\bot)$ is defined set $f^{\alpha+1}(\bot) = f(f^\alpha(\bot))$. Assuming $f^\alpha(\bot)$ exist for all $\alpha < \lambda$ (λ a limit ordinal), satisfying $f^\alpha(\bot) \leq f^\beta(\bot)$ when $\alpha \leq \beta$, we know $\{f^\alpha(\bot) \mid \alpha < \lambda\}$ is directed and so may set

$$f^\lambda(\bot) = \bigsqcup\{f^\alpha(\bot) \mid \alpha < \lambda\}\,.$$

This definition can only go wrong if some $f^\alpha(\bot)$ is not greater than all preceding ones. We show this by induction. We know the following

(a) $f^0(\bot) = \bot \leq f^1(\bot)$ as $\bot$ is less than all $x \in P$.

(b) If $f^\alpha(\bot) \leq f^{\alpha+1}(\bot)$ then $f^{\alpha+1}(\bot) \leq f^{\alpha+2}(\bot)$ by monotonicity.

(c) If $f^\lambda(\bot)$ is defined (λ a limit ordinal) then it is greater than all preceding $f^\alpha(\bot)$ by definition.

These deal easily with all cases other than $f^\lambda(\bot) \leq f^{\lambda+1}(\bot)$, which is proved as follows. We know $f^\alpha(\bot) \leq f^\lambda(\bot)$ for all $\alpha < \lambda$, and so by induction and monotonicity

$$f^\alpha(\bot) \leq f^{\alpha+1}(\bot) \leq f^{\lambda+1}(\bot)\,.$$

It follows that $f^{\lambda+1}(\bot)$ is an upper bound for $\{f^\alpha(\bot) \mid \alpha < \lambda\}$ and so is greater than the least upper bound $f^{\lambda+1}(\bot)$.

Thus the $f^\alpha(\bot)$ all exist and form an increasing chain through P. If f has no fixed point then $f^\alpha(\bot) < f^{\alpha+1}(\bot)$ for all α and so they form a strictly increasing chain. But then all the $f^\alpha(\bot)$ would be different, contradicting the well known result (Hartog's Theorem) that says that for every set P there is an ordinal so large that it cannot be mapped injectively into P.

This contradiction shows that for some ordinal γ we have $f^{\gamma+1}(\bot) = f^\gamma(\bot)$, and it easy to see that then $f^\beta(\bot) = f^\gamma(\bot)$ for all $\beta \geq \gamma$.

If x is any other fixed point of f then it is an easy transfinite induction to prove $f^\beta(\bot) \leq x$ for all β, and it follows that $f^\gamma(\bot)$ (with γ as above) is the least fixed point. ∎

The essence of this proof is that, even if a fixed point is not reached after the ω steps of Theorem 2.2.5, the value reached then is mapped up by f and so the whole process can begin again; and this process could be carried on for so long that we would run out of room in P unless there is a fixed point.

The fixed point produced by this theorem can be reached at any ordinal. To see this let $P = \alpha + 1$ (the set of all ordinals less than or equal to α, which is easily shown to be a cpo) and set

$$f(\beta) = \begin{cases} \beta + 1 & \text{if } \beta < \alpha \\ \alpha & \text{if } \alpha = \beta. \end{cases}$$

It should be clear that the iteration produces its fixed point (α) after exactly α steps.

If x is any element of a cpo such that $x \leq f(x)$ for monotone f it is an easy corollary to the above result that there is a least $y \geq x$ such that $f(y) = y$. (Consider the restriction of f to the cpo $\{y \in P \mid y \geq x\}$.) This observation has a number of useful consequences, one of which is the following.

THEOREM 2.4.2
If P is a cpo and $f : P \to P$ is monotone then the set F of fixed points of F is itself a cpo. If, additionally, P is consistently complete or is a complete lattice then F also has that property.

PROOF
Since we know P has a *least* fixed point (the bottom element of F) it is sufficient for the first part to show that each directed subset Δ of F has a least upper bound in F. Since each element of Δ is a fixed point and by Lemma 2.2.1 we have

$$\bigsqcup \Delta = \bigsqcup \{f(x) \mid x \in \Delta\} \leq f(\bigsqcup \Delta)$$

(where $\bigsqcup$ is defined over P). It follows by the argument above that f has a least fixed point greater than $\bigsqcup \Delta$, which is exactly what we wanted.

In fact this argument shows that if $X \subseteq F$ is such that $\bigsqcup X$ exists in P then $\bigsqcup X$ exists in F. The final parts of the result follow easily from this. ∎

Notice that if f is continuous and Δ is a directed set of fixed points of f then the least upper bound of Δ in P is also a fixed point (and so is the least upper bound in the space of fixed points). This does not in general hold if f is only monotone.

It is possible to generalise the usual ideas of convergence and continuity in mathematics by looking at them topologically. It turns out that if we put an appropriate topology on cpos then continuity defined in terms of directed sets just becomes topological continuity. Formally speaking a *topology* $\mathcal{T}$ on a set X is a set of subsets of X (known as open sets) which contains $\emptyset$ and X and which is closed under arbitrary unions and finite intersections.

Definition
Given a cpo $(P, \leq)$, we can define a topology $(P, \mathcal{T})$ by saying that $U \subseteq P$ is open if and only if, for all directed sets Δ, if there is $y \in U$ with $y \leq \bigsqcup \Delta$ then there is $x \in \Delta$ such that $x \in U$. This is known as the Scott topology (after Dana Scott, who invented it). ∎

This definition is very close to one definition of the standard topology on a metric space which says that a set U is open if and only if any convergent sequence with limit in U must contain some element of U. The open sets we have defined here are all upwards closed, in the sense that $y \in U$ and $x \geq y$ implies $x \in U$. To see this observe that the singleton set $\{x\}$ is directed with limit x and look at this case in the definition above. Given this it is easy to show that it is a topology, the only slightly tricky part being finite intersections. For if U and V are open and Δ is directed with $\bigsqcup \Delta \geq z$ with $z \in U \cap V$, then there exist $a, b \in \Delta$ with $a \in U$ and $b \in V$. But because Δ is directed there is $c \geq a, b$ in Δ; and $c \in U \cap V$ as both U and V are upwards closed.

This topology is quite unlike those usually encountered in other parts of mathematics. For notice that if $x < y$ then there is no open set containing x that does not contain y. This contrasts with the common Hausdorff property which states that any two distinct points should be contained in disjoint open sets. The only standard separation property that the Scott topology has is known as T_0: given two distinct points x and y then *either* there is an open set containing x but not y, *or* there is one containing y but not x. To see that the Scott topology has this property we observe that the set $\{y \mid y \not\leq x\}$ is always open (if a directed set is contained in $\{y \mid y \leq x\}$ then its limit is, too). And given two distinct points x and y we can assume without loss of generality that $y \not\leq x$ (i.e., either $x < y$ or the two points are incomparable). But then y is an element of $\{z \mid z \not\leq x\}$ while x is

not.

At the time of writing T_0 spaces and their relations with partial orders are becoming an important topic of research in computer science. But we will content ourselves with the following basic result, which justifies our introduction of the Scott topology.

THEOREM 2.4.3
If P and Q are cpos then the function $f : P \to Q$ is continuous in the partial order sense if and only if it is continuous with respect to the Scott topology (i.e., if $U \subseteq Q$ is open then $f^{-1}(U)$ is open in P).

PROOF
First suppose f is partial order continuous and $U \subseteq Q$ is open. $f^{-1}(U)$ is upwards closed because, if $f(x) \in U$ and $y \geq x$, then $f(y) \geq f(x)$ which implies $f(y) \in U$. To complete the proof that f^{-1} is open, suppose $\Delta \subseteq P$ is directed with limit $\bigsqcup \Delta$ in $f^{-1}(U)$. Then $f(\bigsqcup \Delta) \in U$ and, since f is continuous, $\{f(x) \mid x \in \Delta\}$ is directed with limit $f(\bigsqcup \Delta)$. So there must be some $x \in \Delta$ with $f(x) \in U$. It follows that $x \in f^{-1}(U)$ which is what was required.

Next suppose f is topologically continuous. It must be monotone since if $x \leq y$ but $f(x) \not\leq f(y)$ then $f(x) \in \{z \in Q \mid z \not\leq f(y)\} = U$, say (an open set as shown above). This means $x \in f^{-1}(U)$ while $y \notin f^{-1}(U)$, contradicting the fact that open subsets of P are upwards closed.

So suppose $\Delta \subseteq P$ is directed. We know by Lemma 2.2.1 that $f(\bigsqcup \Delta) \geq \bigsqcup\{f(x) \mid x \in \Delta\}$. If this inequality were strict then $f(\bigsqcup \Delta)$ would be an element of the open set $U = \{z \in Q \mid z \not\leq \bigsqcup\{f(x) \mid x \in \Delta\}\}$. It would follow that $\bigsqcup \Delta \in f^{-1}(U)$ so that there is some $x \in \Delta$ with $f(x) \in U$. This would be in clear contradiction to the definition of U. We may thus conclude that f is partial order continuous, completing the proof. ∎

Notice that the set $\{y \mid y \geq x\}$ is open if and only if x is po-finite. In a topology where open sets are upwards closed this is the smallest conceivable neighbourhood of the point x (in the sense that $\{y \mid y \geq x\}$ is, for any x, the intersection $V(x)$ of all open sets containing x), just as the singleton $\{x\}$ is in a more conventional (T_1 or Hausdorff) topology. For $V(x)$ to be open is thus a reasonable extension to T_0 spaces of the usual topological definition of *isolated point* (i.e., one such that $\{x\}$ is open). Thus the po-finite elements are precisely the isolated points.

Given this observation, it is interesting to see what can be said about algebraic cpos where the po-finite elements play a special rôle, since every point is a limit of them. In topological terms this is just the spirit of saying that the po-finite points are *dense*. But once again the usual topological definition of a dense set

(every nonempty open set contains an element of it) is less satisfactory because we are in a T_0 space. After all, if our order P has a greatest element $\top$ then the set $\{\top\}$ would then be dense! The po-finite elements are indeed dense and in fact they satisfy the following stronger definition of denseness of a set D (which coincides with the usual one for T_1 spaces).

$$x \in U,\ U \text{ open } \Rightarrow \exists\, y \in D \cap U . y \notin (V(x) \setminus \{x\})$$

In other words U contains some element of D which is not in U simply because it is strictly greater than x. To see this all we have to note is that the directed set of po-finite elements less than or equal to x contains an element of U (as U is open) and no point strictly greater than x. (Notice that what we have actually proved is that there is a finite e in U with $x \in V(e)$.) Thus an algebraic cpo has the property that isolated points are dense.

A *basis* for a topology is a set $\mathcal{B}$ of open sets such that, given any $x \in U$ with U open, there is an element V of $\mathcal{B}$ with $x \in V \subseteq U$. Thus a set is open if, and only if, it is the union of some subset of $\mathcal{B}$. For example, in the usual topology on $\mathbb{R}$, a typical basis is the set of all open intervals with endpoints in $\mathbb{Q}$. There is an interesting basis for the Scott topology on an algebraic cpo. We have already noticed that all elements of

$$\mathcal{B} = \{\, V(e) \mid e \text{ is po-finite}\}$$

are open. They form a basis since, if x belongs to open U, the directed set $\{e \mid e \leq x \wedge e \text{ is po-finite}\}$ must contain an element e of U. Clearly $x \in V(e) \subseteq U$.

This is far from being the only topology of interest when studying partial orders. There are others which have far more conventional properties (such as compact Hausdorff). We will meet another of these when we come to study powerdomains in Chapter 6.

Having introduced the idea of studying partial orders via topology, we should perhaps finally mention that some computer scientists prefer to go the whole way and get their fixed point theorems from topology rather than cpos. Perhaps the best known fixed point theorem in the whole of mathematics is the *contraction mapping theorem* (also known as the Banach fixed point theorem). This states that, in a complete metric space X (one where every Cauchy sequence converges), every contraction mapping has a unique fixed point. A contraction mapping is a function $f : X \to X$ such that there is a positive constant $\kappa < 1$ with $d(f(x), f(y)) \leq \kappa d(x, y)$ for all $x, y \in X$. A sizable minority of the mathematical structures now being created to model computation now rely on this fixed point theorem rather than the ones established earlier in this chapter. Even when one does have a partial order fixed point theory it can be useful to look at a space as a metric space when recursions are contractions: see Exercise 2.4.8 below.

Readers who are interested in pursuing the theory of partial orders in more depth than we have had space to provide might wish to look at [].

2.4.1 König's Lemma

König's Lemma is one of the most frequently used results in theoretical computer science – and we shall need it in Chapter 6. Although it is frequently used in connection with partial orders, this is in fact a result about trees. A tree is a connected graph with no cycles which, in this case, has a designated root node. Such a tree can be broken up into 'levels' according to distance (length of shortest path) from the root, where all non-root nodes have a unique 'predecessor' (the first node passed through on the path to the route) and every node can have any number of 'successors' (nodes of which it is the predecessor). We prove König's Lemma here both because we need it later on, but mainly because it is such an important result generally.

2.4.4 THEOREM (KÖNIG'S LEMMA)
Suppose T is an infinite tree (i.e., has infinitely many nodes) which is finitely branching (each node has only finitely many successors). Then T contains an infinite path (a sequence of nodes $\langle n_0, n_1, \ldots \rangle$ where n_0 is the root and n_{i+1} is always a successor of n_i).

PROOF
Let S be the set of nodes n in T such that the (direct) paths from the root n_0 to infinitely many nodes pass through n. Clearly $n_0 \in S$, because T is infinite. And if, in general, $n \in S$ then at least one of n's successors must be in S. For otherwise there are only finitely many nodes reachable through each of n's finitely many successors, which would contradict the fact that infinitely many are reachable through n. Thus to construct our infinite path all we have to do is, starting with the root n_0, pick n_{i+1} to be one of the successors of n_i that is in S. ∎

Typically, König's Lemma is used to show that if, in some suitable branching structure, some process (involving walking through the structure) can continue to any finite depth, then it can in fact continue forever.

Exercises

2.4.1 Show that any countable directed set Δ has a linearly ordered subset C such that $x \in \Delta \Rightarrow \exists\, y \in C.x \leq y$. Deduce that any countable directed set in a chain-complete partial order has a least upper bound.

2.4.2 By considering the directed subset $\wp(X)$ of the partial order $\mathcal{P}(X)$ where X

is an uncountable set, show that first conclusion of Exercise 2.4.1 does not hold for uncountable directed sets.

2.4.3 Let Δ be directed. Construct an operator $F : \mathcal{P}(\Delta) \to \mathcal{P}(\Delta)$ with the following properties

(a) $X \subseteq F(X)$;
(b) $F(X)$ is directed;
(c) $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$;
(d) $F(X)$ has the same cardinal as X if X is infinite.

2.4.4 Assuming the result of Exercise 2.4.1 and the existence of an operator as in Exercise 2.4.3, prove by induction on the cardinality of a directed subset Δ of a chain-complete partial order that Δ has a least upper bound.

2.4.5 Recall Lemma 2.3.5, where we showed that the least fixed point operator on $P \xrightarrow{c} P$ is continuous. Show that it is monotone on $P \xrightarrow{m} P$ and find an example where it is not continuous.

2.2.6 Suppose P is a set on which there are two complete partial orders $\leq$ and $\preceq$ with the following properties:

(a) The two orders have the same least element $\perp$.
(b) $x \preceq y \Rightarrow x \leq y$.
(b) If Δ is $\preceq$-directed then its least upper bounds with respect to the two orders are the same.

Prove (i) that if f is monotone with respect to both orders and continuous with respect to $\leq$, then it is continuous with respect to $\preceq$, and (ii) that if f is monotone with respect to the two orders then the two least fixed points are the same. (Notice that this situation applies to the finite and infinite sequences of letters ordered by prefix and lexicographically. See Exercise 2.1.5.)

2.4.7 Another (usually different) T_0 topology can be defined on an arbitrary partial order by defining U to be open if and only if it is upwards closed. Show that a function is monotone if and only if it is continuous with respect to this topology.

2.4.8 Suppose X is a metric space (not necessarily complete) and that the contraction map f is known to have a fixed point μf in X. Further suppose C is a nonempty closed subset of X such that $f(C) \subseteq C$ (i.e., $x \in C \Rightarrow f(x) \in C$). Prove that $\mu f \in C$. (This is a potentially stronger fixed point rule than the ones established at the end of Section 2.2.)

2.4.9 One can use the Scott topology to define a topology on the product

$$\prod_{\lambda \in \Lambda} P_\lambda$$

of cpos in two different ways: either the Scott topology of the product order or the topological product of the Scott topologies on the components. (The product topology is defined by the basis consisting of open sets of the form

$$\{\underline{x} \mid \forall \lambda \in F . x_\lambda \in U_\lambda\}$$

as F varies over finite subsets of Λ and each U_λ is open in P_λ.) Show that, if all the P_λ are algebraic, then these two topologies are the same (i) when Λ is finite and (ii) in general.
[Hint: It is enough to show that all elements of a basis for each topology are open in the other.]
How would your answer change if instead of using the Scott topology you used the one defined in Exercise 2.4.7?

Chapter 3

Information systems

3.1 Basics

In Chapter 1 we saw the need for a theory which allows us to build mathematical models, or domains, that solve domain equations such as

$$M \cong B \ + \ (M \to M).$$

The first theory of this type was developed by D.S. Scott circa 1969, and a number of similar approaches have been developed since by various authors. The theory presented in this work is essentially due to Scott, though it is not his original one.

Our basic approach is to develop, for each language, a suitable set of propositions that can be true of programs written in it. The propositions will always be 'finite', in the sense that they can be deduced about the program by observing it for a finite time. We will then identify each program with the set of all propositions that are true of it. Consider, for example, the above equation in the context of the language we met at the end of Chapter 1. A typical proposition about such an expression E might be '$E = 42$', or '*E is a function*', or '*E is a function which, when its argument equals* 12*, gives the value* 13'. It will turn out that, by developing suitable spaces of such propositions, we can naturally solve domain equations. These propositions will usually be called *tokens*.

For example, if the set of objects we are modelling are the functions from $\mathbb{N}$ to $\mathbb{N}$, one appropriate set of tokens is $\mathbb{N} \times \mathbb{N}$ ($\langle n, m \rangle$ meaning '$f(n) = m$'). Notice that, in regarding a function f as the set of all tokens that are true of it, we are identifying it with $\{\langle n, m \rangle \mid f(n) = m\}$: this is exactly the way in which set-theorists model functions.

The central feature of each information system we build will be a set of tokens. However, the structure of the domain generated by a given set A of tokens will

depend not so much on the identities of the individual tokens as on the relationships which exist between them. For example, it is clear that exactly the same set of tokens we used above for functions will suffice for the *relations* between $\mathbb{N}$ and $\mathbb{N}$ and the monotonic functions from $\mathbb{N}$ to $\mathbb{N}$ (where $n \leq m$ implies $f(n) \leq f(m)$). Thus in describing one of these domains, or *information systems*, we will need to record the relationships between its tokens.

Given a finite set X of tokens, we should be able to tell whether or not they are *consistent*, or in other words, whether or not they can all be true at once of something. In our function space example, if $n \neq n'$, then the tokens $\{\langle n, m\rangle, \langle n', m'\rangle\}$ are consistent, because there is always a function that maps n to m and n' to m'. However, if $m \neq m'$ the tokens $\{\langle n, m\rangle, \langle n, m'\rangle\}$ are not consistent, because no function can map n both to m and to m'. Notice that in the space of relations, *any* finite set of tokens is consistent, but that in the space of monotonic functions, if $n < n'$, then $\{\langle n, m\rangle, \langle n', m'\rangle\}$ is only consistent if $m \leq m'$, and $\{\langle n, m\rangle, \langle n, m'\rangle\}$ is consistent only when $m = m'$. A component of each information system will be a set *Con* which consists precisely of those finite sets of tokens that are consistent.

It is possible that even consistent groups of tokens might not be independent of one another. When we know all the information represented by some $X \in Con$ about an object, then it might be possible to deduce that some other token a must also be true about it. In this case we will say that X *entails* a, and write $X \vdash a$. To know the structure of the domain generated by a given set of propositions one must know the form of their entailment relation. Therefore each information system will include $\vdash$, which will be a relation between *Con* and A, the set of tokens.

The entailment relation in our function space example is not very interesting, because any pair of tokens are either inconsistent or completely independent. If X is a consistent finite subset of $\mathbb{N} \times \mathbb{N}$ and $a \in \mathbb{N} \times \mathbb{N}$, then $X \vdash a$ if and only if $a \in X$: any set of tokens can only tell us about a function's value at $n \in \mathbb{N}$ if it contains a token with n as its first component. It becomes less trivial, however, in the context of monotonic functions. Here we will have such entailments as

$$\begin{aligned} \{\langle n, 0\rangle\} &\vdash \langle m, 0\rangle \quad \text{whenever } m \leq n \\ \{\langle n, k\rangle, \langle n', k\rangle\} &\vdash \langle m, k\rangle \quad \text{whenever } n \leq m \leq n'. \end{aligned}$$

If the propositions were drawn from a logic with connectives such as $\wedge$ (and) and $\vee$ (or), we would certainly expect entailments like

$$\begin{aligned} \{a \wedge b\} &\vdash a \\ \{a, b\} &\vdash a \wedge b \\ \{a\} &\vdash a \vee b \end{aligned}$$

In cases like this where the propositions are drawn from some structured

logic, it is likely that we will be able to deduce the consistency and entailment relations from it. What is important, though, is that these relations are exactly the information we require about *any* set of propositions. And by making the relations explicit we are able to abstract completely from the structure of the set of propositions. Indeed the name 'token' we have given them carries the implication that we do not have to look inside them to see their meaning. Formally then, an information system is a triple $\langle A, Con, \vdash \rangle$, where A is any set, Con is a nonempty subset of $\wp(A)$ (the set of all finite subsets of A), and $\vdash$ is a relation between Con and A. Of course, not all such triples will constitute reasonable information systems. Specifically, every information system $\langle A, Con, \vdash \rangle$ must satisfy the following five axioms, each of which specifies a different 'reasonableness' condition.

1. If $X \in Con$, and $Y \subseteq X$, then $Y \in Con$.
2. If $a \in A$, then $\{a\} \in Con$.
3. If $X \in Con$ and $X \vdash a$, then $X \cup \{a\} \in Con$.
4. If $X \in Con$ and $a \in X$, then $X \vdash a$.
5. If $X, Y \in Con$ and $a \in A$ are such that $X \vdash b$ for each $b \in Y$ and $Y \vdash a$, then $X \vdash a$.

Axiom 1 observes that if a set X of tokens can be true of some object, then so can each subset of X (because the same object works). Axiom 2 simply says that no token is completely redundant: there must be *some* object of which it is true. Axiom 3 says that if the set X of tokens can all be true of some object, and if the truth of all of X entails the truth of a, then $X \cup \{a\}$ is consistent, because they are all true of the object that X is. Axiom 4 asserts the trivial fact that if we know that every token in a set that includes a holds, then we know a. Axiom 5 is the transitivity law. It says that if, from X, we can deduce enough information (Y) to deduce a, then we can deduce a from X.

Of these axioms only Axiom 2 is in any sense less than 'clearly true'. One could well imagine a language of propositions in which some were formally unsatisfiable by any program – i.e., were equivalent to *false*. Fortunately it is the only one that is not really used in an important way: if it were omitted hardly any of what follows would be invalidated though occasional modifications would be needed. Essentially, the inclusion of this axiom is a matter of personal taste.

EXAMPLES

The set $A = \mathbb{N} \times \mathbb{N}$ of tokens can easily be used to set up three information systems $\mathbf{F} = \langle A, Con_F, \vdash_F \rangle$ (functions) $\mathbf{R} = \langle A, Con_R, \vdash_R \rangle$ (relations) and $\mathbf{M} =$

$\langle A, Con_M, \vdash_M \rangle$ (monotonic functions) along the lines described above.

$$\begin{array}{rcl}
X \in Con_F & \Leftrightarrow & X \subseteq^{\text{fin}} A \ \wedge \ (\{\langle n,m\rangle, \langle n,m'\rangle\} \subseteq X \Rightarrow m = m') \\
X \vdash_F \langle n,m\rangle & \Leftrightarrow & X \in Con_F \ \wedge \ \langle n,m\rangle \in X \\
X \in Con_R & \Leftrightarrow & X \subseteq^{\text{fin}} A \\
X \vdash_R \langle n,m\rangle & \Leftrightarrow & X \in Con_R \ \wedge \ \langle n,m\rangle \in X \\
X \in Con_M & \Leftrightarrow & X \subseteq^{\text{fin}} A \wedge \\
 & & (n \leq n' \ \wedge \ \{\langle n,m\rangle, \langle n',m'\rangle\} \subseteq X \ \Rightarrow \ m \leq m') \\
X \vdash_M \langle n,m\rangle & \Leftrightarrow & X \in Con_M \wedge \\
 & & ((m = 0 \ \wedge \ \exists\, n' \geq n . \langle n',0\rangle \in X) \vee \\
 & & (\exists\, n', n''.\ n' \leq n \leq n'' \wedge \{\langle n',m\rangle, \langle n'',m\rangle\} \subseteq X))
\end{array}$$

Notice that the definition of Con_M implies the condition directly stated in the definition of Con_F that no number has more than one image, for if $\langle n,m\rangle$, $\langle n,m'\rangle \in X \in Con_M$ then $m \leq m'$ and $m' \leq m$. The reader is invited to check that the triples **R** and **M** are indeed information systems. We will illustrate the sort of calculations required by verifying that **F** is one.

- Axiom 1 holds since if $X \in Con_F$ and $Y \subseteq X$ then $\langle n,m\rangle, \langle n,m'\rangle \in Y \Rightarrow \langle n,m\rangle, \langle n,m'\rangle \in X$.
- Axiom 2 holds since each singleton set $\{\langle n,m\rangle\}$ is trivially in Con_F.
- Axioms 3, 4 and 5 are all easy consequences of the definition that $X \vdash_F \langle n,m\rangle$ if and only if $\langle n,m\rangle \in X$. (These three axioms are trivial in any information system where $X \vdash a \Leftrightarrow a \in X$.)

When, subsequently, we define the Con and $\vdash$ components of information systems, we will not always state explicitly conditions such as '$X \subseteq^{\text{fin}} A$' which can be inferred from the basic structure of information systems. ∎

Remember that the reason for setting up information systems is so that we have a framework for identifying an object with the set of propositions that are true of it. An important feature of each information system will therefore be the set of 'objects' described by it. Such an object, an *element* of the information system, will be any set of tokens which can reasonably be thought to be the set of all propositions true of something.

Definition
Suppose $\mathbf{A} = \langle A, Con, \vdash \rangle$ is an information system. $x \subseteq A$ is said to be an *element* of **A** if it is

(a) *consistent*, in that every $X \in Con$ whenever $X \subseteq^{\text{fin}} x$, and

(b) *deductively closed*, in that, whenever $X \subseteq^{\text{fin}} x$ and $X \vdash a$, then $a \in x$.

The set of all elements of $\mathbf{A}$ is written $|\ \mathbf{A}\ |$. ∎

Clearly each of these conditions must be true of the set of all tokens describing any object. What we are doing is, in some sense, postulating the existence of an object corresponding to every set of tokens satisfying them. The sets of elements of information systems will be very important to us: they are the 'domains' we will construct and use to model programming languages.

Consider the function space example $\mathbf{F}$. Here, because the entailment relation is trivial, *every* subset of $\mathbb{N}\times\mathbb{N}$ is deductively closed. Thus the elements are precisely those subsets x of $\mathbb{N} \times \mathbb{N}$ that are consistent, or equivalently those satisfying the condition:

$$\{\langle n, m\rangle, \langle n, m'\rangle\} \subseteq x \ \Rightarrow\ m = m' \ .$$

In other words, the elements are not the *functions* from $\mathbb{N}$ to $\mathbb{N}$ but the *partial functions*. (Observe that $\emptyset$, $\{\langle 3, 4\rangle\}$ and $\{\langle 2n, n\rangle \mid n \in \mathbb{N}\}$ are all elements.) In fact, the total functions are precisely the *maximal* elements of $\mathbf{F}$, those $x \in|\ \mathbf{F}\ |$ which satisfy

$$y \in|\ \mathbf{F}\ |\ \wedge x\ \subseteq y\ \Rightarrow\ x = y \ .$$

Given a maximal element x, there is no element y of which strictly more tokens are true (either because there are no more tokens to add or because the addition of any further token would mean inconsistency). This turns out to be a common feature of information systems: in defining a system which is intended to describe some known set of elements, you find that the language of propositions you have devised adds classes of partial elements. As we shall see shortly every information system must have a structure where all objects are approximated by a set of partial ones, so that, if the objects we are trying to describe have no such structure themselves, then we are bound to add partial elements as above. See the exercises for more examples of this.

In the information system $\mathbf{R}$ of relations, it is clear that every set is deductively closed and consistent. Thus $|\ \mathbf{R}\ | = \mathcal{P}(\mathbb{N} \times \mathbb{N})$, which is, of course, the set of relations between $\mathbb{N}$ and $\mathbb{N}$. Thus this time the underlying domain of our information system is exactly what we had expected. It is easy, however, to devise an information system of relations in which the relations correspond to the maximal elements (i.e., complete objects): one simply invents, for each pair $\langle n, m\rangle$ an extra token $\neg\langle n, m\rangle$ with the meaning 'n does not relate to m'. ($\neg\langle n, m\rangle$ will of course be inconsistent with $\langle n, m\rangle$.) Which of these representations one uses will depend on whether one wants all fully defined relations to be independent, or whether one wants one relation to be 'stronger', or 'more defined than' another. When using

information systems to model computer programs, this decision would probably rest on whether the n *not* being related to m was actually observable, or whether we only actually see when things *are* related.

The structure of the information system $\langle M, Con_M, \vdash_M \rangle$ of monotonic functions is left as an exercise to the reader.

We will now prove a few basic results about information systems. The first establishes two rather obvious facts: the empty set of tokens is consistent, and, if a token a is entailed by some set X, then it is entailed by every larger element of Con.

3.1.1 LEMMA
Suppose $\mathbf{A} = \langle A, Con, \vdash \rangle$ is an information system, then

(a) $\emptyset \in Con$;

(b) if $X \in Con$, $Y \subseteq X$ and $Y \vdash a$, then $X \vdash a$.

PROOF
(a) follows because Con is by assumption nonempty, and by Axiom 1. (b) holds by Axiom 5 since, by Axiom 4, $X \vdash b$ for each $b \in Y$. ∎

Suppose $\langle A, Con, \vdash \rangle$ is an information system and $B \subseteq A$. We define $\overline{B}$, the *deductive closure* of B, by

$$\overline{B} = \{a \in A \mid \exists X \subseteq^{\text{fin}} B.\ X \vdash a\} .$$

$\overline{B}$ is the set of all tokens deducible from B. Notice that $B \subseteq C \Rightarrow \overline{B} \subseteq \overline{C}$. Note also that Lemma 3.1.1(b) shows that, if $X \in Con$, then $\overline{X} = \{a \in A \mid X \vdash a\}$.

3.1.2 LEMMA
Suppose $\mathbf{A} = \langle A, Con, \vdash \rangle$ is an information system, $X \in Con$ and $Y \subseteq^{\text{fin}} A$. Then

(a) if $X \vdash a$ for every $a \in Y$ then $Y \in Con$;

(b) $\overline{X} \in |\ \mathbf{A}\ |$;

(c) $\overline{\emptyset} \subseteq x$ for every $x \in |\ \mathbf{A}\ |$.

PROOF
(a) We will prove that $X \cup Y \in Con$ by induction on the size of Y; the result then follows by Axiom 1. If $Y = \emptyset$, this result is trivial, so suppose $Y = Y' \cup \{a\}$ and

$X \cup Y' \in Con$. Then since, by assumption, $X \vdash a$, we have $X \cup Y' \vdash a$ by Lemma 3.1.1(b), so that $X \cup Y \in Con$ by Axiom 3.

(b) If $Y \subseteq^{\text{fin}} \overline{X}$ then $Y \in Con$ by (a); hence $\overline{X}$ is consistent. If $Y \subseteq^{\text{fin}} \overline{X}$ and $Y \vdash a$, then $X \vdash a$ by definition of $\overline{X}$ and Axiom 5; hence $\overline{X}$ is deductively closed. (c) If $a \in \overline{\emptyset}$ and $x \in \mid \mathbf{A} \mid$, then $a \in x$ because $\emptyset \subseteq^{\text{fin}} x$, $\emptyset \vdash a$ and x is deductively closed. ∎

In the light of (a) above it is convenient informally to allow '$\vdash$' to be used as a relation between Con and Con: henceforward '$X \vdash Y$' will mean the same as '$X \vdash a$ for each $a \in Y$'.

Notice that (c) above tells us that the partial order of $\mid \mathbf{A} \mid$ under $\subseteq$ has a *minimum* element. $\overline{\emptyset}$ is the set of tokens that tell us nothing at all, because they are true of every object. (In most of the examples we meet $\overline{\emptyset}$ will equal $\emptyset$, because there is generally no reason to include tokens that carry no information.)

EXAMPLE

The above result shows that the set of elements of each information system must have a least element under $\subseteq$. It is therefore not possible to have an information system representing a totally 'flat' set (of more than one element) where there is no order between the elements. However we can do the next best thing and define a domain which has all the elements from a given set, unordered, plus a bottom element below them all (recall Exercise 2.1.9). Given such a set A, we can form an information system $\mathbf{flat}(A) = \langle A, Con, \vdash \rangle$, the *flat domain on* A, by setting

- $Con = \{\emptyset\} \cup \{\{a\} \mid a \in A\}$, and
- $X \vdash a \Leftrightarrow a \in X$.

It is easy to see that $\mid \mathbf{flat}(A) \mid = Con$ has exactly the required order, with each member a of A represented by the singleton $\{a\}$. The order structure of the flat domain on $\mathbb{N}$, the natural numbers, is shown in Figure 3.1. ∎

Recall that any set of tokens is said to be consistent if each of its finite subsets is in Con. The next Lemma justifies this term, for it shows that every consistent set of tokens is true of some object (i.e., element) in the underlying domain.

3.1.3 LEMMA

If $\mathbf{A} = \langle A, Con, \vdash \rangle$ is an information system and $U \subseteq A$ is consistent, then $\overline{U} \in \mid \mathbf{A} \mid$.

PROOF

First we show that $\overline{U}$ is consistent. If $Y = \{a_1, a_2, \ldots, a_n\}$ is any finite subset of $\overline{U}$, then for each $i \in \{1, 2, \ldots, n\}$, there exists $Z_i \subseteq^{\text{fin}} U$ such that $Z_i \vdash a_i$. But $Z = Z_1 \cup \ldots \cup Z_n$ is a finite subset of U, and so is in Con. Now $Z \vdash a_i$ for each i (by Lemma 3.1.1(b)); Lemma 3.1.2(a) then tells us that $Y \in Con$.

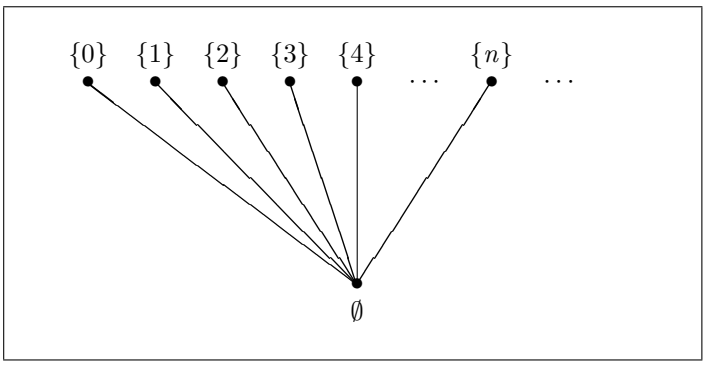

Figure 3.1 The flat domain on $\mathbb{N}$

To show deductive closure, suppose that $Y \subseteq^{\text{fin}} \overline{U}$ and that $Y \vdash b$. Exactly as above we can find $Z \subseteq^{\text{fin}} U$ such that $Z \vdash Y$. But then $Z \vdash b$ by Axiom 5, so $b \in \overline{U}$ as required. ∎

Definition
If $\mathbf{A} = \langle A, Con_A, \vdash_A \rangle$ and $\mathbf{B} = \langle B, Con_B, \vdash_B \rangle$ are any two information systems, the map $\phi : A \to B$ is said to be an *isomorphism* between $\mathbf{A}$ and $\mathbf{B}$ if (a) it is a bijection and (b) it preserves the consistency and entailment relations in the sense that

- $X \in Con_A \Leftrightarrow \{\phi(a) \mid a \in X\} \in Con_B$
- $X \vdash_A a \Leftrightarrow \{\phi(a') \mid a' \in X\} \vdash_B \phi(a)$.

In this case we write $\mathbf{A} \equiv \mathbf{B}$. Two isomorphic information systems are in a strong sense essentially the same. In particular it is easy to see that, if ϕ, $\mathbf{A}$ and $\mathbf{B}$ are as above then

$$| \mathbf{B} |= \{\{\phi(a) \mid a \in x\} \mid x \in | \mathbf{A} |\} .$$

It is easy to see that $\mathbf{A} \equiv \mathbf{B}$ and $\mathbf{B} \equiv \mathbf{C}$ implies that $\mathbf{A} \equiv \mathbf{C}$. ∎

Except for the names of their tokens, two isomorphic information systems are essentially the same, and will have all of the same properties.

Exercises

3.1.1 Let S be any set. Take $\mathbf{S}$ to be the information system $\langle S, Con_S, \vdash_S \rangle$, where $Con_S = \{X \mid X \subseteq^{\text{fin}} S\}$ and $X \vdash_S a$ iff $a \in X$. Show that $| \mathbf{S} |= \mathcal{P}(S)$.

Now, let $F = \{X \mid X \subseteq^{\text{fin}} S\}$. Define Con_F and $\vdash_F$ in such a way that $\mathbf{F} = \langle F, Con_F, \vdash_F \rangle$ is an information system and $\mid \mathbf{F} \mid$ is naturally isomorphic to $\mid \mathbf{S} \mid$.
[Hint: Say that $G \in F$ is 'information towards' $T \subseteq S$ if $G \subseteq T$. You should think of a token G in F as containing the same information as the conjunction of the finitely many tokens of **S** *it contains.]*

3.1.2 Let A be a set with at least two elements. Construct an information system whose token-set is A and whose elements are those of the flat domain on A together with a single extra *top* element which is greater than all others.

3.1.3 Let B be the set of finite nonempty sequences of $\underline{0}$s and $\underline{1}$s whose last element is a $\underline{1}$ – for example $\underline{0}\,\underline{1}$, $\underline{1}\,\underline{1}$, $\underline{1}\,\underline{0}\,\underline{0}\,\underline{1}$.
We write $s \leq t$ iff s is an initial subsequence of t; in other words if $t = s\hat{}u$ for some sequence u.
Let $Con_B = \{X \subseteq^{\text{fin}} B \mid a, b \in X \Rightarrow a \leq b \text{ or } b \leq a\}$, and say $X \vdash_B a$ iff $a \leq b$ for some $b \in X$. Show that $\mathbf{B} = \langle B, Con_B, \vdash_B \rangle$ is an information system, and that the maximal elements of $\mid \mathbf{B} \mid$ are in natural 1-1 correspondence with the interval $(0, 1] = \{x \in \mathbb{R} \mid 0 < x \leq 1\}$.
[Hint: notice that every such real number has a unique binary expansion $0.b_1 b_2 b_3 \ldots$ where infinitely many of the b_i are 1.]

3.1.4 Construct an information system whose maximal elements represent the *strictly* monotonic functions from $\mathbb{N}$ to $\mathbb{N}$ (namely, functions f such that $n < m \Rightarrow f(n) < f(m)$).
[Hint: Notice that the set $\mathbb{N} \times \mathbb{N}$ of tokens used for the examples in the text is no longer correct, since for example no strictly monotone function can map 3 to 1.]

3.1.5 Given sets A and B, construct an information system with token set $A \times B$ which represents the 1-1 functions from A to B (i.e., $f(x) = f(y) \Rightarrow x = y$. What are the maximal elements of your system in the following cases:

(a) A is larger than B;
(b) B is larger than A;
(c) A and B are the same finite size
(d) A and B have the same infinite size (i.e., there is a bijection between them).

*3.1.6 Let I be the set of all nonempty open intervals in $\mathbb{R}$. Let $Con_I = \{X \subseteq^{\text{fin}} I \mid \bigcap X \neq \emptyset\}$, and define $X \vdash_I a$ iff $a \supseteq \bigcap X$. Show that $\mathbf{I} = \langle I, Con_I, \vdash_I \rangle$ is an information system. Describe the maximal elements of $\mid \mathbf{I} \mid$.

3.1.7 Prove that, if $\mathbf{A} = \langle A, Con, \vdash \rangle$ is an information system then $\mid \mathbf{A} \mid$ contains

a maximum element (i.e., an x such that $x \supseteq y$ for all $y \in |\ \mathbf{A}\ |$) if and only if $Con = \wp(A)$. (In other words, if and only if the consistency relation is trivial.)

3.1.8 Show that it is possible to determine the triple $\langle A, Con_A, \vdash_A \rangle$ from the corresponding set $|\ \langle A, Con_A, \vdash_A \rangle\ |$ of elements. (Note that this implies that, in general, if $\mathbf{A} \neq \mathbf{B}$ then $|\ \mathbf{A}\ | \neq |\ \mathbf{B}\ |$.)

3.2 The partial order structure of information systems

There is a natural partial order on the set of elements $|\ \mathbf{A}\ |$ of information systems determined by set inclusion. $x \subseteq y$ if every token that is true of x is also true of y; if y is in some sense *more defined* than x. This partial order will be very important to us: indeed from one point of view, information systems are little more than concrete representations of their partial orders of elements. In this section we will investigate the properties it has.

Recall that two partial orders $\langle P, \leq_1 \rangle$ and $\langle Q, \leq_2 \rangle$ are said to be *order isomorphic* if there is a bijection $\psi : P \to Q$ such that $x \leq_1 y \Leftrightarrow \psi(x) \leq_2 \psi(y)$ for all x, y in P. If $|\ \mathbf{A}\ |$ and $|\ \mathbf{B}\ |$ are order isomorphic then we write $\mathbf{A} \cong \mathbf{B}$. Note that $\mathbf{A} \equiv \mathbf{B} \Rightarrow \mathbf{A} \cong \mathbf{B}$ since, if $\phi : A \to B$ is an isomorphism between $\mathbf{A}$ and $\mathbf{B}$, the map defined $\psi(x) = \{\phi(a) \mid a \in x\}$ is plainly an order isomorphism from $|\ \mathbf{A}\ |$ to $|\ \mathbf{B}\ |$.

The reverse is not true in general, since there are many cases of non-isomorphic information systems giving rise to order-isomorphic domains. A simple example was given in Exercise 3.1.1. As a further example, given any system $\mathbf{A} = \langle A, Con, \vdash \rangle$ and a nonempty set B, the system $\mathbf{A}^B$, with token set $A \times B$ and consistency and entailment relations defined

- $X \in Con' \Leftrightarrow \{a \mid \exists\, b. \langle a, b \rangle \in X\} \in Con$
- $X \vdash' \langle a, b \rangle \Leftrightarrow \{a' \mid \exists\, b'. \langle a', b' \rangle \in X\} \vdash a$,

has a set of elements order isomorphic to $|\ \mathbf{A}\ |$ but only rarely will $\mathbf{A}^B \equiv \mathbf{A}$. (In $\mathbf{A}^B$ we have simply made a copy of each of $\mathbf{A}$'s tokens for each element of $\mathbf{B}$, but treated all these copies as equivalent pieces of information. One can quickly show that $|\ \mathbf{A}^B\ | = \{x \times B \mid x \in |\ \mathbf{A}\ |\}$.)

Recall the definition of a directed set from the last chapter. A directed set Δ of elements of an information system has the property that, given any finite set of tokens taken from elements of Δ, there is a single element of Δ containing all the tokens. This follows easily from the fact that every finite subset of Δ has an upper bound in Δ.

3.2.1 THEOREM
Suppose $\mathbf{A} = \langle A, Con, \vdash \rangle$ is an information system. Then $|\ \mathbf{A}\ |$ is a cpo. Furthermore

(a) if Δ is a directed subset of $|\ \mathbf{A}\ |$, then the least upper bound of Δ is given by $\bigcup \Delta$;

(b) if $\mathcal{S}$ is any nonempty subset of $|\ \mathbf{A}\ |$, then the greatest lower bound of $\mathcal{S}$ exists and is equal to $\bigcap \mathcal{S}$.

PROOF
To show that $|\ \mathbf{A}\ |$ is a cpo it will be enough, by Lemma 3.1.2(c), to prove (a). Clearly $x \subseteq \bigcup \Delta$ whenever $x \in \Delta$, and if $y \in |\ \mathbf{A}\ |$ is any upper bound for Δ then clearly $\bigcup \Delta \subseteq y$. Hence to prove (a) it will be enough to prove that $\bigcup \Delta \in |\ \mathbf{A}\ |$.
Consistency. If $Z = \{a_1, a_2, \ldots, a_n\} \subseteq \bigcup \Delta$, then there are $x_1, x_2, \ldots, x_n \in \Delta$ such that $a_i \in x_i$. By directedness of Δ, there is some $x^* \in \Delta$ such that $x_i \subseteq x^*$ for each i. But then $Z \subseteq x^*$, so $Z \in Con$ because x^* is consistent.
Deductive closure. If $Z = \{a_1, a_2, \ldots, a_n\} \subseteq \bigcup \Delta$ and $Z \vdash a$ then as above we can find $x^* \in \Delta$ with $Z \subseteq x^*$. But then $a \in x^*$ by deductive closure of x^*, so $a \in \bigcup \Delta$ as required.

The proof of (b) is even easier. Again, it will obviously suffice to show that $\bigcap \mathcal{S} \in |\ \mathbf{A}\ |$. *Consistency* follows from the fact that $\mathcal{S}$ is nonempty: each finite subset of $\bigcap \mathcal{S}$ is a subset of every (and so at least one) element of $\mathcal{S}$. *Deductive closure* follows because if $X \subseteq^{\text{fin}} \bigcap \mathcal{S}$ and $X \vdash a$ then, for each $x \in \mathcal{S}$, $X \subseteq^{\text{fin}} x$ so that $a \in x$ by deductive closure of x. ∎

Thus not only is the domain of elements of an information system a cpo, but the limit of a directed set is simply its union: the set of all tokens contained in any element of the set. We used the directedness of Δ crucially in the proof of (a): that this was needed can be seen from our function space examples **F** and **M**. In both of these domains it is easy to construct subsets that have no upper bound (for example the set containing two different total functions). In **M**, because it has a non-trivial entailment relation, it is possible to find nondirected sets $\mathcal{S}$ of elements such that, even though $\bigsqcup \mathcal{S}$ exists, it is not equal to $\bigcup \mathcal{S}$. (For example the set $\{\{\langle 3,3\rangle\}, \{\langle 23,3\rangle\}\}$ has least upper bound $\{\langle n,3\rangle \mid n \in \{3,4,\ldots,23\}\}$.)

A direct application of Zorn's Lemma[1] shows that every element of an information system **A** can be extended to a maximal element. As we have already seen,

[1]Zorn's Lemma states that, in a partial order P where each linearly ordered subset has an upper bound, each element x is weaker than some maximal y. Zorn's Lemma is an equivalent of the axiom of choice, and in fact the full power of the axiom of choice is required to prove that information systems have maximal elements: see Exercise 3.2.3.

in flat domains and the function space examples, an information system can have many different maximal elements.

Recall that a subset S of a partial order is said to be po-consistent if all its finite subsets have an upper bound in P, and that a cpo is said to be consistently complete if every po-consistent subset has a least upper bound. This notion of consistency is an obvious analogue of our information system definition, for consistency there is determined from finite subsets as well. The following Lemma shows that it is easy to determine the po-consistent subsets of an information system.

3.2.2 LEMMA
Suppose $\mathbf{A}$ is an information system, and $\mathcal{S}$ is a subset of $|\ \mathbf{A}\ |$. Then $\mathcal{S}$ is po-consistent if and only if $\bigcup\mathcal{S}$ is consistent as a set of tokens. Furthermore, in this case, $\bigsqcup\mathcal{S}$ exists and equals $\overline{\bigcup\mathcal{S}}$.

PROOF
If $\bigcup\mathcal{S}$ is consistent then, by Lemma 3.1.3, $x^* = \overline{\bigcup\mathcal{S}}$ is an element of $|\ \mathbf{A}\ |$ and is obviously *an* upper bound for $\mathcal{S}$. That it is the *least* upper bound is an easy consequence of the deductive closure of elements of $|\ \mathbf{A}\ |$.

If $\mathcal{S}$ is po-consistent and $X = \{a_1, a_2, \ldots, a_n\}$ is any finite subset of $\bigcup\mathcal{S}$, then we can find $x_1, x_2, \ldots, x_n \in \mathcal{S}$ such that $a_i \in x_i$. But by po-consistency of $\mathcal{S}$ there must be some upper bound $x^* \in|\ \mathbf{A}\ |$ for $\{x_1, x_2, \ldots, x_n\}$. But then $X \subseteq^{\text{fin}} x^*$, so $X \in \mathit{Con}$ as x^* is consistent. Hence $\bigcup\mathcal{S}$ is consistent. That $\bigsqcup\mathcal{S} = \overline{\bigcup\mathcal{S}}$ then follows by the above. ∎

The following Lemma is a trivial corollary to Lemma 3.2.2 (and is also a corollary to the result in Exercise 2.1.6, given Theorem 3.2.1).

3.2.3 LEMMA
The domain of elements of each information system is consistently complete. ∎

The next theorem completes the sequence of results proving properties about the partial orders associated with information systems. It establishes that, in addition to being consistently complete cpos they are also *algebraic* (every element is the least upper bound of the po-finite elements below it). It also characterises the po-finite elements of these partial orders.

3.2.4 THEOREM
If $\mathbf{A} = \langle A, \mathit{Con}, \vdash\rangle$ is an information system, then the po-finite elements of $|\ \mathbf{A}\ |$ are precisely $\{\overline{X} \mid X \in \mathit{Con}\}$. Furthermore the partial order $|\ \mathbf{A}\ |$ is algebraic.

PROOF

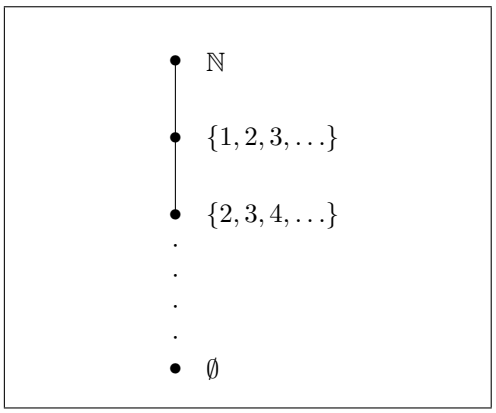

Figure 3.2 N – po-finite elements that are not finite sets.

Suppose first that $X = \{a_1, a_2, \ldots, a_n\} \in Con$ and Δ is a directed set with $\bigcup \Delta \supseteq \overline{X}$. Then for each i we can find $x_i \in \Delta$ with $a_i \in x_i$. But then there is some $x^* \in \Delta$ which is an upper bound for the x_i, so that $X \subseteq x^*$. $\overline{X} \subseteq x^*$ now follows from the deductive closure of x^*. Hence $\overline{X}$ is po-finite.

Now if x is any element of $|\ \mathbf{A}\ |$ it is clear that $\{\overline{X} \mid X \subseteq^{\text{fin}} x\}$ is a directed set with limit x. Thus, if x is po-finite, there must be some $X \subseteq^{\text{fin}} x$ with $\overline{X} \supseteq x$; but clearly $\overline{X} \subseteq x$ so that $\overline{X} = x$ as desired. Whether x is po-finite or not we know by the above that each element of $\{\overline{X} \mid X \subseteq^{\text{fin}} x\}$ is, so we also have that $|\ \mathbf{A}\ |$ is algebraic. ∎

Notice that, while the above result implies that any finite (as a set) $x \in |\ \mathbf{A}\ |$ is necessarily po-finite, it does not imply the reverse. (This is why we use the term po-finite to avoid confusion.) Consider the following example.

EXAMPLE

We define an information system $\mathbf{N} = \langle \mathbb{N}, Con, \vdash \rangle$ whose tokens are the natural numbers:

- $Con = \wp(\mathbb{N})$
- $X \vdash n \Leftrightarrow \exists\, m \in X . m \leq n.$

The elements of this system are $\{\emptyset\} \cup \{\{n \mid n \geq m\} \mid m \in \mathbb{N}\}$; their partial order structure is illustrated in Figure 3.2. As is easily verified, *all* the elements of $|\ \mathbf{N}\ |$ are po-finite; but only $\emptyset$ is a finite set. ∎

We have now shown that each information system represents an algebraic, consistently complete cpo. That this is a complete characterisation is shown by

the final theorem in this section, where we see that every such partial order can be represented by an information system.

3.2.5 THEOREM
If $\langle P, \leq \rangle$ is an algebraic, consistently complete cpo then P is order isomorphic to the domain of elements of some information system.

PROOF
We construct an information system $\mathbf{E} = \langle E, Con, \vdash \rangle$ as follows:

- E is the set of po-finite elements of P;
- $X \in Con$ if and only if $X \subseteq^{\text{fin}} E$ and X is po-consistent;
- $X \vdash e$ if and only if $X \in Con$ and $e \leq \bigsqcup X$.

It is quite easy to verify that $\mathbf{E}$ is an information system. We set up a map from P to $|\,\mathbf{E}\,|$ in the natural way:

$$\phi(x) \;=\; \{e \in E \mid e \leq x\}\,.$$

It is easy to verify that ϕ is well-defined (i.e., that $\phi(x) \in |\,\mathbf{E}\,|$ for all $x \in P$). It is injective because, since P is algebraic, $x = \bigsqcup \phi(x)$ for every $x \in P$ (i.e., x is determined by $\phi(x)$). It is clearly order-preserving, because

$$x \leq y \;\Rightarrow\; \{e \in E \mid e \leq x\} \subseteq \{e \in E \mid e \leq y\}\,.$$

Thus it only remains to show that ϕ is surjective. Suppose $z \in |\,\mathbf{E}\,|$; define $x_z = \bigsqcup z$ (this is a well-defined element of P because z is consistent and hence po-consistent, and P is consistently complete). Certainly $\phi(x_z) \supseteq z$. Claim z is a directed set (note that it contains the least element $\bot$ of P and so is nonempty). If $X \subseteq^{\text{fin}} z$ then $\bigsqcup X$ is po-finite by Lemma 2.1.2 (a), and thus $X \vdash \bigsqcup X$ by definition of $\vdash$. Deductive closure of z then means $\bigsqcup X \in z$ as required to prove z directed as claimed.

Now whenever $e \in \phi(x_z)$ we have $e \leq \bigsqcup z$, where z is known to be directed. Thus (for we have assumed P is algebraic) $e \leq e'$ for some $e' \in z$. Trivially then $e \in z$ as z is deductively closed. Therefore we must have $\phi(x_z) = z$, which demonstrates that ϕ is surjective. Thus ϕ is an order isomorphism. ∎

Exercises

3.2.1 For each of the following partial orders, either construct an information system whose set of elements is order isomorphic to it, or explain why no such information system exists.

(i) The numbers $\{0, 1, 2, \ldots, n\}$ with their usual order.
(ii) The natural numbers $\mathbb{N}$ with their usual order $\leq$.
(iii) $\mathbb{N}$ with the *reverse* order: $n \preceq m \Leftrightarrow m \leq n$.
(iv) The closed interval $[0, 1] (= \{x \mid 0 \leq x \leq 1\})$ with its usual order.
(v) $\wp(\mathbb{N}) \cup \{\mathbb{N}\}$ with the subset order $\subseteq$.
(vi) $\wp(\mathbb{N}) \cup \{\mathbb{N}\}$ with the superset order $\supseteq$ (i.e., the reverse inclusion order).

3.2.2 The information system $\mathbf{R} = \langle \mathbb{Q}, Con, \vdash \rangle$ (where $\mathbb{Q}$ is the set of rational numbers) is formed by setting

– $X \in Con \Leftrightarrow X \subseteq^{\text{fin}} \mathbb{Q}$
– $X \vdash a \Leftrightarrow \exists\, b \in X . b \geq a$.

Describe the po-finite elements of $\mid \mathbf{R} \mid$ and show that the non-po-finite elements are in natural, order-preserving correspondence with $\mathbb{R} \cup \{\infty\}$, the set of all real numbers plus a point at (positive) infinity. *(This construction of the real numbers from the rationals is known as Dedekind cuts.)*

^{M}3.2.3 Let $\mathcal{X}$ be any set of nonempty sets and set $A = \{\langle x, a \rangle \mid x \in \mathcal{X} \wedge a \in x\}$. By forming a suitable information system $\langle A, Con, \vdash \rangle$ prove, from the principle that information systems have maximal elements, that there is a total *choice function* $f : \mathcal{X} \to \bigcup \mathcal{X}$ such that $f(x) \in x$ for all x. *(The statement that there always exists such a function is known as the axiom of choice. This exercise proves that the principle that information systems have maximal elements is an equivalent of the axiom of choice.)*

^{M}3.2.4 Let $n > 2$. Which of the following sets of subsets of Euclidean space $\mathbb{R}^n$ is the set of elements of an information system with token set $\mathbb{R}^n$?

(i) The open sets.
(ii) The closed sets.
(iii) The convex sets. (A subset of $\mathbb{R}^n$ is said to be *convex* if, whenever $\mathbf{x}, \mathbf{y} \in C$ and $0 \leq \lambda \leq 1,$, $\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in C$.)
(iv) The set of proper vector subspaces. (A subset of $\mathbb{R}^n$ is a *vector subspace* if it is closed under taking linear combinations of its elements. A subspace is *proper* if it is not equal to the whole space $\mathbb{R}^n$.)
(v) The set of all connected sets. (A set C is said to be connected if, whenever U and V are disjoint open sets such that $C \subseteq U \cup V$, then either $C \cap U = \emptyset$ or $C \cap V = \emptyset$.)

3.2.5 Is it always the case that if $X \subset \mathcal{P}(A)$ is such that $\bigcup X = A$ and X is an algebraic, consistently complete cpo under $\subseteq$, then there is an information system with token set A whose elements are X?
[Hint: Remember that, in an information system, the greatest lower bound of any nonempty set is its intersection.]

3.3 Mappings between information systems

Many objects in computing can be thought of as mechanisms that take input and produce output. In this section we meet a class of relations that can reasonably be claimed to contain all computable mechanisms of this sort: they give information about the output as a consequence of information about their input.

Definition
Suppose that $\mathbf{A} = \langle A, Con_A, \vdash_A \rangle$ and $\mathbf{B} = \langle B, Con_B, \vdash_B \rangle$ are information systems. We will say that a relation $r \subseteq Con_A \times B$ is an *approximable mapping* from $\mathbf{A}$ to $\mathbf{B}$ if it satisfies

(a) $X\ r\ b_1 \wedge X\ r\ b_2 \wedge \ldots \wedge X\ r\ b_n \ \Rightarrow\ \{b_1, b_2, \ldots b_n\} \in Con_B$

(b) $(X\ r\ b_1 \wedge X\ r\ b_2 \wedge \ldots \wedge X\ r\ b_n) \ \wedge \{b_1, b_2, \ldots b_n\} \vdash_B b \ \Rightarrow\ X\ r\ b$

(c) $X \vdash_A X' \ \wedge\ X'\ r\ b \ \Rightarrow\ X\ r\ b$.

If r is an approximable mapping from $\mathbf{A}$ to $\mathbf{B}$ we will write $r : \mathbf{A} \to \mathbf{B}$. ∎

An approximable mapping can be regarded as an 'information respecting' relation between two information systems. We think of r as being a machine for giving us information about an element of $|\ \mathbf{B}\ |$ if we give it information about an element of $|\ \mathbf{A}\ |$: $X\ r\ b$ means 'if r is told X about an element of $|\ \mathbf{A}\ |$, then b is one of the things it tells us about the corresponding element of $|\ \mathbf{B}\ |$'. Under this interpretation the three conditions above are quite natural.

(a) says that if we put consistent information into r, then we get consistent information out.

(b) says that if when we give r the information X, we can get enough information to deduce b, then we can get b itself.

(c) says that anything which can be got from r by giving it information X' can also be got by giving it more information.

Remembering that we are thinking of the tokens of an information system as the facts that can be observed in a finite time, it is natural to expect any computable process for computing $|\ \mathbf{B}\ |$s from $|\ \mathbf{A}\ |$s to correspond to an approximable mapping. This is discussed further below.

EXAMPLE
We can define an approximable mapping $sum : \mathbf{F} \to \mathbf{M}$ (respectively the information systems of functions and monotonic functions from $\mathbb{N}$ to itself seen earlier in this chapter) to represent a mechanism which, given information about an element f of $|\ \mathbf{F}\ |$, returns information about the function

$$sum_f(n) \ = \ \textstyle\sum_{i=0}^{n-1} f(i)\ .$$

The approximable mapping is formed by setting $X\ sum\ \langle n, m\rangle$ if and only if

$$\exists\, k_0, \ldots, k_{n-1}.\ (\forall\, i \in \{0, 1, \ldots, n-1\}.\ \langle i, k_i\rangle \in X)\ \wedge\ \textstyle\sum_{i=0}^{n-1} k_i = m\ .$$

It is easy to check that *sum* is indeed an approximable mapping. This example shows why we have taken an approximable mapping to be a relation between *sets* of tokens and tokens: sometimes it is necessary to know many facts about the input to a machine to deduce some single fact about its output. For example, to be able to deduce the value of $sum_f(4)$, we need to know all of the values $\{f(0), f(1), f(2), f(3)\}$. Notice also that it is sometimes possible to deduce facts about the result without knowing *any* facts about the input: we know that $sum_f(0) = 0$ without needing any information about f. ∎

Each approximable mapping gives rise naturally to a function from the elements of its left hand system to those of its right hand one. One simply collects together all the information that is deducible about the result from finite subsets of its argument.

Definition
If $r : \mathbf{A} \to \mathbf{B}$ is an approximable mapping, then we define a function $|\ r\ |\!:\!|\ \mathbf{A}\ |\!\to\!|\ \mathbf{B}\ |$ by

$$|\ r\ |\ (x) = \{b \in B \mid \exists\, X \subseteq^{\text{fin}} x.\ X\ r\ b\}\ .$$ ∎

3.3.1 LEMMA
If $r : \mathbf{A} \to \mathbf{B}$ is an approximable mapping then $|\ r\ |$ is a well-defined monotonic function, in that if $x, y \in |\ \mathbf{A}\ |$ then

$$x \subseteq y\ \Rightarrow |\ r\ |\ (x) \subseteq |\ r\ |\ (y)\ .$$

PROOF
To show that $\mid r \mid$ is well-defined we must prove that, for each $x \in\mid \mathbf{A} \mid$, $\mid r \mid (x)$ is an element of $\mid \mathbf{B} \mid$.
Consistency. This follows because if $\{b_1, b_2, \ldots, b_n\} \subseteq\mid r \mid (x)$ then for each i we can find $X_i \subseteq^{\text{fin}} x$ such that $X_i \; r \; b_i$. But then $X^* = \bigcup_{i=1}^{n} X_i \subseteq^{\text{fin}} x$, and $X^* \; r \; b_i$ for each i by Lemma 3.1.1(b) and condition (c) above. Condition (a) above then tells us that $\{b_1, b_2, \ldots, b_n\} \in Con_B$.
Deductive closure. This follows because if $Y = \{b_1, b_2, \ldots, b_n\} \subseteq\mid r \mid (x)$ and $Y \vdash_B b$, then as above we can find $X^* \subseteq^{\text{fin}} x$ such that $X^* \; r \; b_i$ for each i. Condition (b) above then immediately tells us that $X^* \; r \; b$, which implies that $b \in\mid r \mid (x)$.

That $\mid r \mid (x) \subseteq\mid r \mid (y)$ when $x \subseteq y$ follows immediately from the definition of $\mid r \mid$. ∎

The last part of the above Lemma says that the more defined the argument to $\mid r \mid$, the more defined the result. This is because a more defined argument simply has more tokens true of it, so r is able to deduce more tokens about the result.

Notice that an approximable mapping r allows us to deduce tokens about its image only from *finite* sets of tokens from its argument. Thus $\mid r \mid (x)$ consists of only those tokens which it is possible to deduce from finite subsets of x. In particular it is clear that any token $b \in\mid r \mid (x)$ is in $\mid r \mid (\overline{X})$ for some $X \subseteq^{\text{fin}} x$, so that $\mid r \mid (x)$ is simply the union of the results of applying $\mid r \mid$ to the finite approximations $\{\overline{X} \mid X \subseteq^{\text{fin}} x\}$. (See below for a proof of this fact.) This is why approximable mappings are so called: it is always possible to deduce the result of applying the associated function to an infinite object by looking at its finite approximations.

We constructed approximable mappings as mathematical models of 'realistic' mechanisms for computing functions: in order to deliver a finite amount of information they can only consume a finite amount. In modelling the behaviour of computer programs it will generally be sufficient to restrict ourselves to the type of function which arises from these mechanisms.

Recall that a function $f : P \to Q$ (P, Q cpos) is said to be continuous if, whenever $D \subseteq P$ is directed, $\bigsqcup\{f(x) \mid x \in D\}$ exists and equals $f(\bigsqcup D)$. There is clearly a close relationship between continuous functions on partial orders and approximable mappings on information systems. This is confirmed by the following Theorem, which shows that the continuous functions between the sets of elements of two information systems are precisely those induced by approximable mappings.

3.3.2 THEOREM
If $\mathbf{A} = \langle A, Con_A, \vdash_A \rangle$ and $\mathbf{B} = \langle B, Con_B, \vdash_B \rangle$ are two information systems, then a function $f :\mid \mathbf{A} \mid \to\mid \mathbf{B} \mid$ is continuous if and only if there is an approximable mapping $r_f : \mathbf{A} \to \mathbf{B}$ such that $\mid r_f \mid = f$. Furthermore, this r_f is the *only* approximable

mapping such that $\mid r_f \mid = f$ (which means there is a natural 1-1 correspondence between the spaces of continuous functions and approximable mappings).

PROOF
Suppose first that r is an approximable mapping and that $\Delta \subseteq \mid \mathbf{A} \mid$ is directed. We must show that $\mid r \mid (\bigcup \Delta) = \bigcup \{\mid r \mid (x) \mid x \in \Delta\}$. (Since $\mid r \mid$ is monotonic, we already know that the set $\{\mid r \mid (x) \mid x \in \Delta\}$ is directed and so has the right hand side as its least upper bound.) We know, by Lemma 2.2.1, that

$$\mid r \mid (\bigcup \Delta) \supseteq \bigcup \{\mid r \mid (x) \mid x \in \Delta\} .$$

So suppose that $b \in \mid r \mid (\bigcup \Delta)$. Then there is some finite subset $X = \{a_1, a_2, \ldots, a_n\}$ of $\bigcup \Delta$ such that $X \, r \, b$. We can find $x_1, x_2, \ldots, x_n \in \Delta$ such that $a_i \in x_i$ and, since Δ is directed, $x^* \in \Delta$ such that $x_i \subseteq x^*$ for each i. But then $X \subseteq x^* \in \Delta$ and $X \, r \, b$, so that $b \in \bigcup \{\mid r \mid (x) \mid x \in \Delta\}$ as required. This completes the proof that $\mid r \mid$ is continuous.

Secondly suppose that $f : \mid \mathbf{A} \mid \rightarrow \mid \mathbf{B} \mid$ is a continuous function. We define an approximable mapping $r_f : \mid \mathbf{A} \mid \rightarrow \mid \mathbf{B} \mid$ as follows:

$$X \, r_f \, b \;\Leftrightarrow\; b \in f(\overline{X}) .$$

That r_f is an approximable mapping follows easily from the fact that f is a (well-defined) monotonic function. Observe that, for each $X \in Con_A$, we have $\mid r_f \mid (\overline{X}) = f(\overline{X})$, because $Y \subseteq^{\text{fin}} \overline{X}$ implies that $\overline{Y} \subseteq \overline{X}$ so that $f(\overline{Y}) \subseteq f(\overline{X})$. Hence, for each $x \in \mid \mathbf{A} \mid$ we have

$$\begin{array}{rcll} f(x) & = & f(\bigcup \{\overline{X} \mid X \subseteq^{\text{fin}} x\}) & \text{by Theorem 3.2.4} \\ & = & \bigcup \{f(\overline{X}) \mid X \subseteq^{\text{fin}} x\} & \text{as } f \text{ is continuous and} \\ & & & \{\overline{X} \mid X \subseteq^{\text{fin}} x\} \text{ directed} \\ & = & \bigcup \{\mid r_f \mid (\overline{X}) \mid X \subseteq^{\text{fin}} x\} & \text{by the above remarks} \\ & = & \mid r_f \mid (\bigcup \{\overline{X} \mid X \subseteq^{\text{fin}} x\}) & \text{as } \mid r_f \mid \text{ is continuous} \\ & = & \mid r_f \mid (x) . & \end{array}$$

r_f is unique in having this property, since if r and r' are different approximable mappings there must be $X \in Con_A$ such that

$$\{b \in B \mid X \, r \, b\} \neq \{b \in B \mid X \, r' \, b\} .$$

But it is easy to verify that the left hand side above is $\mid r \mid (\overline{X})$ and the right hand side is $\mid r' \mid (\overline{X})$, showing that $\mid r \mid$ and $\mid r' \mid$ are different functions. This completes the proof of the Theorem. ∎

This result, together with the results of the last section, shows that whenever P and Q are algebraic, consistently complete cpos and $f : P \to Q$ is continuous then there is an approximable mapping between any pair of information systems with domains of elements isomorphic to P and Q which corresponds to f. Just as we can think of information systems as giving us concrete representations of this class of partial orders, we can think of the approximable mappings as being representations of the continuous functions between them. As we will see in the next chapter, this is most useful.

Exercises

3.3.1 Suppose $\mathbf{A}$ is an information system. Do there always exist approximable mappings from $\mathbf{A}$ to itself whose effect, as a function from $\mid \mathbf{A} \mid$ to $\mid \mathbf{A} \mid$, is:

(i) the identity map $f(x) = x$;
(ii) the constant function $f(x) = y$;
(iii) the function

$$f(x) = \begin{cases} \overline{\emptyset} & \text{if} \quad y \not\subseteq x \\ y & \text{if} \quad y \subseteq x \end{cases}$$

(y being some element of $\mid \mathbf{A} \mid$; note that the possibility of (ii) and (iii) might depend on which y is chosen)? In the cases where this is possible, construct the approximable mapping.

3.3.2 Suppose $\mathbf{A} \cong \mathbf{B}$. Construct an approximable mapping $r : \mathbf{A} \to \mathbf{B}$ such that $\mid r \mid$ is an order isomorphism. Need r be unique?

3.3.3 Consider the set $\mathbf{A} \to \mathbf{B}$ of approximable mappings from $\mathbf{A}$ to $\mathbf{B}$ as a partial order under $\subseteq$. Show that, if r and s are any two approximable mappings then $r \subseteq s$ if and only if $\mid r \mid \leq \mid s \mid$ (the order on functions being as defined in Chapter 2).
Show that if Δ is a directed set of approximable mappings then $\bigcup \Delta$ is one and that $\mid \bigcup \Delta \mid = \bigsqcup \{\mid r \mid \mid r \in \Delta\}$.

Chapter 4

Constructing domains

We now know what an information system is and how it represents the 'domain' of its elements. In this chapter we will learn how to construct the information systems we need for defining the semantics of programming languages. Specifically we examine what is meant by domain equations such as

$$\mathbf{D} \;=\; \mathbf{B} + (\mathbf{D} \to \mathbf{D})$$

and learn how to solve them.

4.1 Simple domain constructors

Before we can think about solving an equation like the one above we need to know what symbols like '+' and '$\to$' mean. They will be operators, *domain constructors*, on information systems whose effects on the underlying domains of elements are what we expect of '+', '$\to$' etc. An n to m domain constructor will be an operator which, given an n-tuple of information systems, returns an m-tuple. Most of the basic constructors we meet will be 1 to 1 (unary) or 2 to 1 (binary), but combinations of these sometimes give constructors with more complicated 'types'. The basic constructors which it is useful to have in constructing semantic domains include the following.

$\uparrow$ Lifting: the elements of $|\ \mathbf{A}_{\uparrow}\ |$ should be in natural correspondence with those of $|\ \mathbf{A}\ |$ plus a single extra bottom element. We will see many ways in which this is useful later on.

$\times$ Cartesian product: the elements of $|\ \mathbf{A} \times \mathbf{B}\ |$ should be in natural correspondence with the pairs of elements from $|\ \mathbf{A}\ |$ and $|\ \mathbf{B}\ |$.

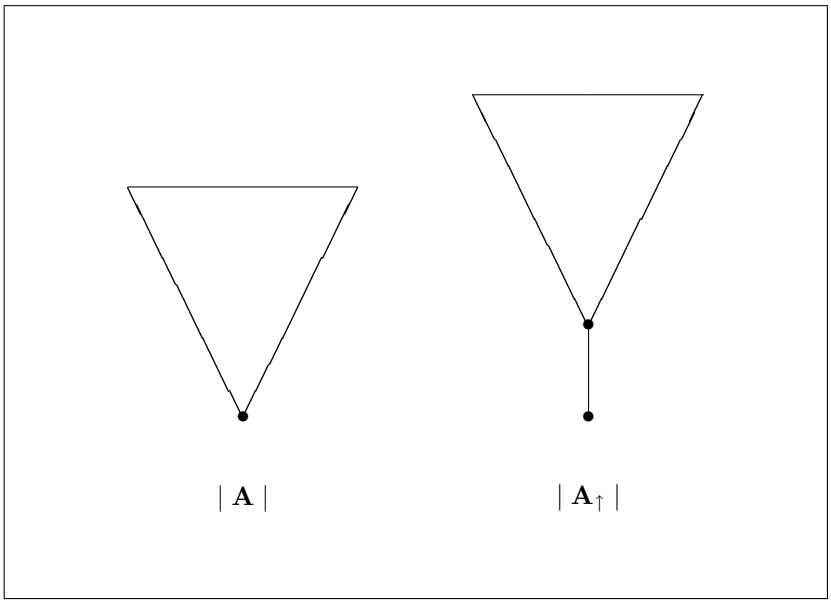

Figure 4.1 Lifting

$+$ Sum: $|\ \mathbf{A} + \mathbf{B}\ |$ should consist of disjoint copies of $|\ \mathbf{A}\ |$ and $|\ \mathbf{B}\ |$.

$\rightarrow$ Function space: $|\ \mathbf{A} \rightarrow \mathbf{B}\ |$ should consist of the continuous functions from $|\ \mathbf{A}\ |$ to $|\ \mathbf{B}\ |$.

$\mathcal{P}$ Powerset (usually called *powerdomain*): $|\ \mathcal{P}(\mathbf{A})\ |$ should essentially consist of the subsets of $|\ \mathbf{A}\ |$. What 'essentially' means here will become clearer when we come to discuss powerdomains in the next chapter.

In this section we will see how a few of the simpler constructors (including $\times$ and $+$) are defined. We will then develop the theory needed to solve domain equations before returning to define the function space constructor at the end of the chapter. Further constructors (including powerdomain) will be covered in later chapters.

Lifting

A simple but useful example is provided by *lifting*. If $\mathbf{A}$ is any information system, $\mathbf{A}_{\uparrow}$ should have essentially the same structure as $\mathbf{A}$ except that $\mathbf{A}$ has been 'lifted' on top of a new bottom element (see Figure 4.1).

Remember that each element of an information system is the set of all tokens, or propositions, that are true of it. To construct $\mathbf{A}_{\uparrow}$ from $\mathbf{A}$ we need to have a rich

enough language of propositions to decide whether an element is in the lifted part of the domain or not (noting that only the bottom element is not in the lifted part). The new bottom element must be a strict subset of all elements of the lifted version of **A**. The obvious way to construct $\mathbf{A}_\uparrow$ is to retain all the structure of **A**, but to add an extra token, say a, to the token set with the meaning 'this element is in the lifted part'. (The new bottom cannot have any special tokens, since it is a subset of everything.)

Because every element of **A** is in the lifted part of $\mathbf{A}_\uparrow$, any of **A**'s tokens should entail a, and a should be consistent with any consistent set of them. However, a should not be entailed by the empty set of tokens: this will be the new bottom of the domain.

We need a way of finding the new token a so that it cannot be confused with the old tokens. We cannot simply select a fixed object a and, whatever the structure of **A**, take the token set of $\mathbf{A}_\uparrow$ to be $A \cup \{a\}$. For example, in any system of the form $(\mathbf{A}_\uparrow)_\uparrow$, there would be confusion between the tokens introduced at the first and second liftings. It would be possible to avoid confusion by always selecting a in such a way that it is guaranteed not to be in A: for example, we could set $a = A$ as no set is an element of itself. However this approach is unsatisfactory for several reasons, not the least of which is that (contrary to the idea that tokens are propositions) it is then impossible to tell simply by looking at a token whether it is the 'new' token or an 'old' one – we would have to know the context (i.e., the complete set of tokens) in which it arose. A much better solution is obtained by labelling all the old tokens as such, so that we can introduce a fixed new token without fear of confusion.

Definitions
If A is any set (of tokens), we define

$$A_\uparrow \;=\; \{lifted\} \cup \{\langle old, a\rangle \;\mid\; a \in A\}\,.$$

The obvious injection function $\iota_\uparrow : A \to A_\uparrow$ is defined

$$\iota_\uparrow(a) \;=\; \langle old, a\rangle$$

and the (partial) projection function $\pi_\uparrow : A_\uparrow \to A$ is defined

$$\pi_\uparrow(\langle old, a\rangle) \;=\; a\,.$$

When we look at an element a of $A_\uparrow$ we know immediately if it is the new token $lifted$ or if it is 'in A'. If it is 'in A' we write $a \sqsubseteq A$ (pronounced 'a is squarely in A'). ∎

Definition
If $\mathbf{A} = \langle A, Con, \vdash \rangle$ is an information system then $\mathbf{A}_\uparrow$ is defined to be $\langle A_\uparrow, Con', \vdash' \rangle$, where $A_\uparrow$ is as defined above and

(a) $X \in Con' \;\Leftrightarrow\; \pi_\uparrow(X) \in Con$

(b) $X \vdash' a \;\Leftrightarrow\; X \neq \emptyset \;\wedge\; (a = lifted \;\vee (a \sqsubseteq A \;\wedge\; \pi_\uparrow(X) \vdash \pi_\uparrow(a)\,))$

Here, $\pi_\uparrow$ has been extended to sets of tokens in the usual way: $\pi_\uparrow(X) = \{\pi_\uparrow(a) \mid a \in X\}$. ∎This is just what was described informally above: the new token $lifted$ is a weaker proposition than any of $\mathbf{A}$'s tokens except that it is exactly as strong as any elements of $\mathbf{A}$'s $\overline{\emptyset}$. Like these, it says nothing more than that any element of which it is part is not the bottom element of $\mathbf{A}_\uparrow$.

4.1.1 PROPOSITION
If $\mathbf{A}$ is an information system then so is $\mathbf{A}_\uparrow$, and furthermore:

$$|\,\mathbf{A}_\uparrow\,| = \{\emptyset\} \cup \{x_\uparrow \mid x \in |\,\mathbf{A}\,|\}\,.$$

PROOF
It is easy but tedious to verify that $\mathbf{A}_\uparrow$ satisfies the axioms for an information system. Observe that $\emptyset \not\vdash' a$ for any $a \in A_\uparrow$ so that $\overline{\emptyset} = \emptyset \in |\,\mathbf{A}_\uparrow\,|$. If $x \in |\,\mathbf{A}_\uparrow\,|$ and $x \neq \emptyset$ then, by (b) above, $lifted \in x$. By (a) and (b), plainly $\pi_\uparrow(x) \in |\,\mathbf{A}\,|$. Equally, it is easy to see that if $x \in |\,\mathbf{A}\,|$ then $x_\uparrow \in |\,\mathbf{A}_\uparrow\,|$. ∎

Clearly the map $lift : |\,\mathbf{A}\,| \to |\,\mathbf{A}_\uparrow\,|$ defined

$$lift(x) \;=\; x_\uparrow$$

is an order isomorphism from $|\,\mathbf{A}\,|$ to $|\,\mathbf{A}_\uparrow\,| - \{\emptyset\}$.

EXAMPLE
When we lift the one point domain $\perp = \langle \emptyset, \{\emptyset\}, \emptyset \rangle$ we get an information system with one token and two elements:

$$\begin{aligned} \perp_\uparrow &= \langle \{lifted\}, \{\emptyset, \{lifted\}\}, \{\langle \{lifted\}, lifted \rangle\} \rangle \\ |\,\perp_\uparrow\,| &= \{\emptyset, \{lifted\}\}\,. \end{aligned}$$

If we repeat this lifting n times to get $\perp_n = (\ldots(\perp_\uparrow)_\uparrow \ldots)_\uparrow$, the resulting information system has the n tokens $\{a_0, \ldots, a_{n-1}\}$, where $a_0 = lifted$ and $a_{m+1} = \langle old, a_m \rangle$. All sets of these elements are consistent, and

$$X \vdash a_i \;\Leftrightarrow\; X \text{ contains an } a_j \text{ with } j \geq i.$$

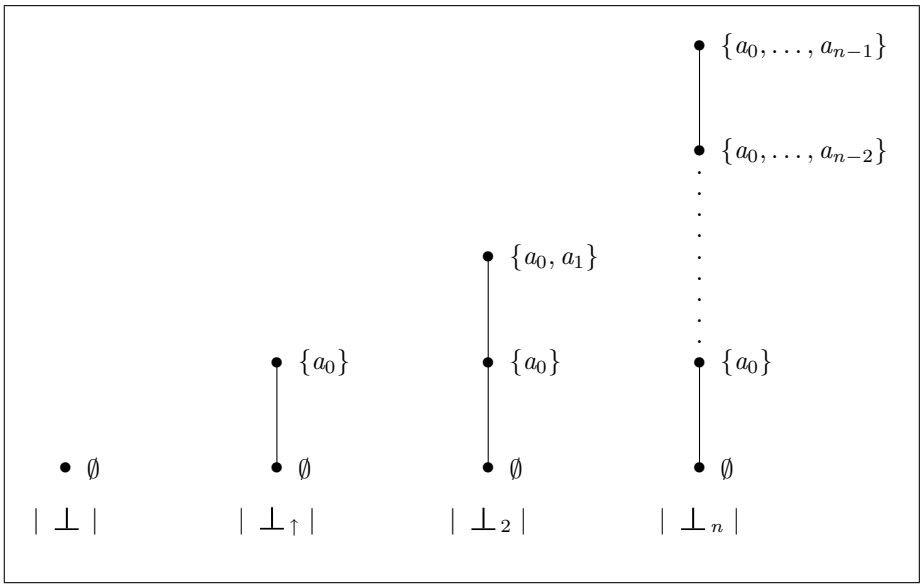

Figure 4.2 The results of repeated lifting.

Thus the tokens are strictly ordered as propositions, with those that have been 'lifted' most often being strongest. $\perp_n$ contains $n+1$ linearly ordered elements (see Figure 4.2).

It is interesting how the domain $\perp_n$ sits inside $\perp_m$ when $m \geq n$. We will soon see that this is important. ∎

Cartesian product

The Cartesian product $S \times T$ of two sets S and T consists of all pairs $\langle s, t \rangle$ with $s \in S$ and $t \in T$. Recall from Chapter 2 that, if S and T are partially ordered, we can give an order to $S \times T$ by defining

$$\langle s, t \rangle \leq \langle s', t' \rangle \iff s \leq_S s' \wedge t \leq_T t' .$$

We need a way of combining two information systems so that the elements of the result are in natural, order preserving correspondence with the Cartesian product of the domains of elements of the original systems.

The first thing to decide about $\mathbf{A} \times \mathbf{B}$ is the shape of its token set. It is tempting to use tokens of the form $\langle a, b \rangle$, for $a \in A$ and $b \in B$. But this does not work since if A (say) were empty (i.e., $\mathbf{A} = \perp$), so would be $A \times B$ – leaving us no tokens to describe the second component of the pair. The correct answer is that

a collection of pieces of information about x and y separately, *provided we know which of x and y each item refers to*, will be enough to describe $\langle x, y\rangle$ fully. This is because complete descriptions of x and y fully determine $\langle x, y\rangle$. Thus an adequate set of tokens for $\mathbf{A} \times \mathbf{B}$ will be the union of separate copies of A and B.

Definitions
If A and B are sets (of tokens), we define

$$A + B \;=\; \{\langle left, a\rangle \;\mid\; a \in A\} \cup \{\langle right, b\rangle \;\mid\; b \in B\}\,.$$

Injection functions $\iota_{left} : A \to A + B$ and $\iota_{right} : B \to A + B$ are defined

$$\begin{aligned} \iota_{left}(a) &= \langle left, a\rangle \\ \iota_{right}(b) &= \langle right, b\rangle \end{aligned}$$

and partial projection functions $\pi_{left} : A + B \to A$ and $\pi_{right} : A + B \to B$ are defined

$$\begin{aligned} \pi_{left}(\langle left, a\rangle) &= a \\ \pi_{right}(\langle right, b\rangle) &= b\,. \end{aligned}$$

As with lifting, when $a \in A + B$ we write $a \sqsubseteq A$ or $a \sqsubseteq B$ if a 'is in' A or B respectively (i.e., $a \in \iota_{left} A$ or $a \in \iota_{right} B$ respectively). ∎

Because the two components of $\langle x, y\rangle$ are essentially independent (the fact that the left-hand component is x tells us nothing about y and vice-versa) the consistency relationship and entailment relation of $\mathbf{A} \times \mathbf{B}$ are inherited in a very simple way from $\mathbf{A}$ and $\mathbf{B}$.

Definition
If $\mathbf{A} = \langle A, Con_A, \vdash_A\rangle$ and $\mathbf{B} = \langle B, Con_B, \vdash_B\rangle$ are information systems, then $\mathbf{A} \times \mathbf{B}$ is defined to be $\langle A + B, Con, \vdash\rangle$, where

(a) $X \in Con \;\Leftrightarrow\; \pi_{left}(X) \in Con_A \wedge \pi_{right}(X) \in Con_B$

(b) $X \vdash a \;\Leftrightarrow\; a \sqsubseteq A \wedge \pi_{left}(X) \vdash_A \pi_{left}(a)$ or $a \sqsubseteq B \wedge \pi_{right}(X) \vdash_B \pi_{right}(a)$. ∎

4.1.2 PROPOSITION
If $\mathbf{A}$ and $\mathbf{B}$ are information systems then so is $\mathbf{A} \times \mathbf{B}$, and furthermore:

$$|\, \mathbf{A} \times \mathbf{B} \,| = \{x + y \;\mid\; x \in |\, \mathbf{A} \,| \wedge y \in |\, \mathbf{B} \,|\}\,.$$

PROOF
Let $\mathbf{A} \times \mathbf{B} = \langle A + B, Con, \vdash \rangle$ as in the definition above. The information system axioms are all trivial to verify. Consider Axiom 3, for example.

If $X \vdash a$ then, by symmetry, it is enough to consider the case when $a \sqsubseteq A$. By definition of $\vdash$ we know that $\pi_{left}(X) \vdash_A \pi_{left}(a)$ and hence (by Axiom 3 of $\mathbf{A}$) that $\pi_{left}(X \cup \{a\}) \in Con_A$. As $\pi_{right}(X \cup \{a\}) = \pi_{right}(X) \in Con_B$, we thus have $X \cup \{a\} \in Con$ by definition of Con.

We now turn to the proof of the second part of the proposition. If z is any subset of $A + B$ we have $z = \pi_{left}(z) + \pi_{right}(z)$. If z is an element of $| \mathbf{A} \times \mathbf{B} |$ then it is easy to see that $\pi_{left}(z)$ and $\pi_{right}(z)$ are consistent and deductively closed, so are elements of $| \mathbf{A} |$ and $| \mathbf{B} |$ respectively. This proves that the left-hand-side above is contained in the right-hand-side.

If $x \in | \mathbf{A} |$ and $y \in | \mathbf{B} |$ then, as $X \subseteq^{\text{fin}} x + y$ implies $\pi_{left}(X) \subseteq^{\text{fin}} x$ and $\pi_{right}(X) \subseteq^{\text{fin}} y$, we know $x + y$ is consistent. It remains to prove deductive closure. If $X \subseteq^{\text{fin}} x + y$ and $X \vdash a$ then, supposing without loss of generality that $a \sqsubseteq A$, $\pi_{left}(X) \vdash_A \pi_{left}(a)$ and hence $\pi_{left}(a) \in x$. This means that $a \in x + y$ as required.

Hence $x + y \in | \mathbf{A} \times \mathbf{B} |$ as required, completing the proof of the proposition. ∎

Clearly the map $pair : | \mathbf{A} | \times | \mathbf{B} | \to | \mathbf{A} \times \mathbf{B} |$ defined by

$$pair\langle x, y \rangle \;=\; x + y$$

is an order isomorphism. This proposition essentially justifies our construction of $\mathbf{A} \times \mathbf{B}$. The way we used a disjoint *union* of token sets to effect the *product* of elements has much in common with the natural set-theoretic isomorphism between $\mathcal{P}(A \cup B)$ and $\mathcal{P}(A) \times \mathcal{P}(B)$ when $A \cap B = \emptyset$.

EXAMPLES
It is easy to see that $\perp \times \perp = \perp$, because $\emptyset + \emptyset = \emptyset$. Indeed, for any information system $\mathbf{A}$, $\perp \times \mathbf{A}$ and $\mathbf{A} \times \perp$ are trivially isomorphic to $\mathbf{A}$.

$\perp_\uparrow \times \perp_\uparrow$ has two tokens $\{\langle left, lifted \rangle, \langle right, lifted \rangle\}$ and four elements. Figure 4.3 shows how these elements are ordered and how they coincide with $| \perp_\uparrow | \times | \perp_\uparrow |$. Notice the way that '*left*' and '*right*' tokens respectively tell us about the left-hand and right-hand components of a pair.

In general, $\perp_n \times \perp_m$ has $n + m$ tokens and $(n+1) \times (m+1)$ elements. Note that it is rather easier to use Proposition 4.1.2 to determine the domain structure of these systems than work them out directly. ∎

Clearly binary product construction can be iterated to produce the Cartesian product of any finite list of information systems, for example

$$\mathbf{A} \times (\mathbf{B} \times \ldots (\mathbf{Y} \times \mathbf{Z}) \ldots) .$$

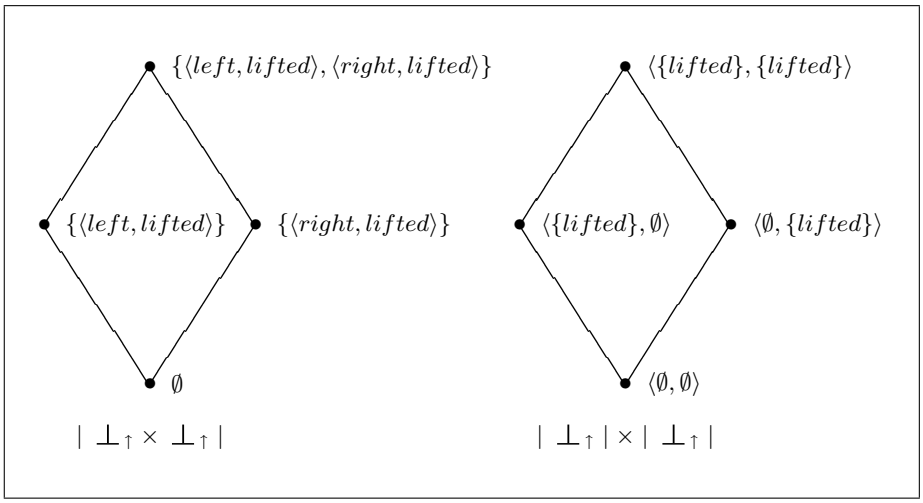

Figure 4.3 The equivalence of the two forms of '×'

If, on the other hand, we either wanted to avoid identifying n-tuples with iterated pairs, or to use potentially infinite product spaces such as (borrowing notation from Section 2.3)

$$\mathbf{A}^X \qquad \text{or} \qquad \prod_{\lambda\in\Lambda} \mathbf{A}_\lambda$$

the direct binary construction is easily adapted. The first[1] of these is X-many copies of the single system $\mathbf{A}$, while the second is the product of an indexed family of potentially different ones. The second (which is, of course, more general than the first) can be defined with token set $\{\langle\lambda, a\rangle \mid \lambda \in \Lambda \wedge a \in A_\lambda\}$ (where A_λ is the token set of $\mathbf{A}_\lambda$): the finite set X of these tokens is consistent if and only if $\pi_\lambda(X) = \{a \mid \langle\lambda, a\rangle \in X\} \in Con_\lambda$ for all λ, and $X \vdash \langle\lambda, b\rangle$ if and only if

$$\pi_\lambda(X) \vdash_\lambda b\,.$$

One can show without too much difficulty that the set of elements of this system is order isomorphic to

$$\prod_{\lambda\in\Lambda} |\, \mathbf{A}_\lambda \,|$$

(under its standard componentwise order), which is what we might have hoped.

[1]Depending one the context, it is sometimes better to write $\mathbf{A}^X$ as $X \to \mathbf{A}$, since an X-indexed tuple of elements of $|\, \mathbf{A} \,|$ is just a function from X to $|\, \mathbf{A} \,|$. However, if one does this it is essential that one carefully avoids confusion with the information system function-space constructor $\mathbf{B} \to \mathbf{A}$ which we will meet in Section 4.3. The only difference is that in one case the left hand argument is a set and in the other an information system.

Sum

The *sum* of two domains | **A** | and | **B** | should consist of separate copies of | **A** | and | **B** |. We have already seen one context where this construct is needed: in our second example language every expression denotes *either* a basic value *or* a function from expression values to expression values. We do not expect there will be any reason to prefer an element of one half of the sum, and indeed would expect that the two halves would be so distinct that there is no order between elements from opposite halves. This causes a significant problem when we try to form the sum as an information system, as the domain of elements of any such sum must have a least element – and clearly we cannot take one element of either half to be this bottom unless we change the assumptions made above.

There are two well-known symmetrical ways of summing a pair of domains so that the result has a minimum element. Inevitably, both involve an element of compromise. The first approach is to identify the least elements of the summands (which always exist in information systems), keeping all other elements distinct: this is known as a *coalesced* sum and will be written $\mathbf{A} \oplus \mathbf{B}$. The other is to add a bottom element that is not 'in' either of the summands, which remain completely separate above the new bottom, as though both were lifted above it: this is known as a *separated* sum and will be written $\mathbf{A} + \mathbf{B}$.[2] The effects of these two methods of summing are illustrated in Figure 4.4. If we are prepared to put up with asymmetry in the order treatment of **A** and **B** it is possible to get an exact 'disjoint union' of their elements. One solution is $\mathbf{A} \oplus \mathbf{B}_{\uparrow}$, where the bottom of | **B** | is placed above that of | **A** |. Another, rather more radical, solution may be found in Exercise 4.1.4.

We can expect the token set of either form of sum to contain copies of the token sets of the systems which are being summed. In order to avoid any confusion, these should be kept separate. We would expect an element of the | **A** | part of | $\mathbf{A} \oplus \mathbf{B}$ | or | $\mathbf{A} + \mathbf{B}$ | to consist only of A-tokens and vice-versa. Thus we can expect pairs of tokens from opposite sides to be inconsistent.

When constructing the coalesced sum we have to be careful of any tokens in $\overline{\emptyset}^A$ and $\overline{\emptyset}^B$ because we are identifying these two elements. If we retained these tokens in $\mathbf{A} \oplus \mathbf{B}$ they would presumably become part of $\overline{\emptyset}$ in the new system, and hence a subset of every element. Though this would not be incorrect, it would be inconsistent with the idea that no proposition about an element of **A** should be true about an element of **B**. We therefore choose the cleaner option of discarding all the (meaningless) propositions in $\overline{\emptyset}^A$ and $\overline{\emptyset}^B$ before constructing $\mathbf{A} \oplus \mathbf{B}$. This means that the bottom element of $\mathbf{A} \oplus \mathbf{B}$ is always $\emptyset$.

[2]Unfortunately, the literature is not consistent about which of the symbols + and $\oplus$ mean which type of sum: sometimes they are interchanged. We have followed what we believe is the more common notation.

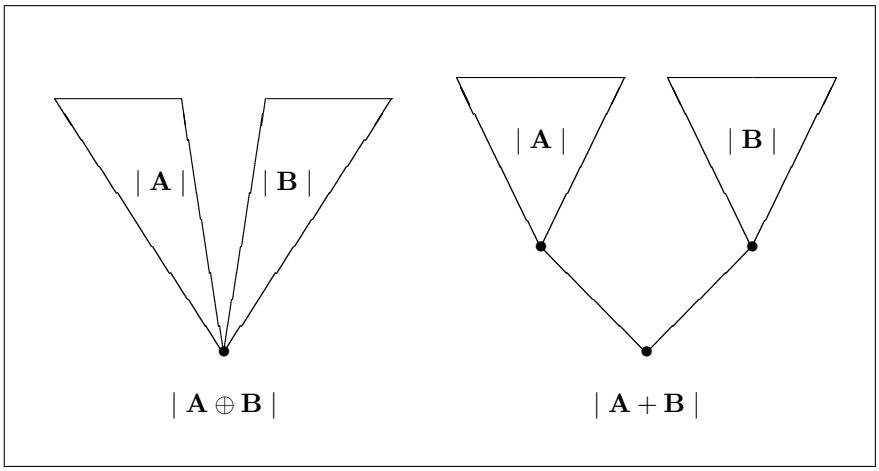

Figure 4.4 Coalesced and separated sums.

Definition
If $\mathbf{A}$ and $\mathbf{B}$ are information systems then $\mathbf{A} \oplus \mathbf{B}$ is defined to be $\langle A' + B', Con, \vdash \rangle$, where $A' = A - \overline{\emptyset}^A$, $B' = B - \overline{\emptyset}^B$ and

(a) $X \in Con \Leftrightarrow \pi_{left}(X) \in Con_A \wedge \pi_{right}(X) = \emptyset$ or
$\pi_{right}(X) \in Con_B \wedge \pi_{left}(X) = \emptyset$

(b) $X \vdash a \Leftrightarrow a \sqsubseteq A' \wedge \pi_{left}(X) \vdash_A \pi_{left}(a)$ or
$a \sqsubseteq B' \wedge \pi_{right}(X) \vdash_B \pi_{right}(a)$. ∎

The following proposition justifies this construction.

4.1.3 PROPOSITION
If $\mathbf{A}$ and $\mathbf{B}$ are information systems then so is $\mathbf{A} \oplus \mathbf{B}$. Furthermore:

$$| \mathbf{A} \oplus \mathbf{B} | = \{\iota_{left}(x - \overline{\emptyset}^A) \mid x \in | \mathbf{A} |\} \cup \{\iota_{right}(x - \overline{\emptyset}^B) \mid x \in | \mathbf{B} |\}$$

and the two parts of the right-hand-side have intersection $\{\emptyset\}$.

PROOF
The information system axioms are, as usual, easy but tedious to verify. The last line of the proposition is trivial. Thus we will prove only the set equality.

Observe first that any element of $| \mathbf{A} \oplus \mathbf{B} |$ is a subset of either $\iota_{left}(A')$ or $\iota_{right}(B')$, for otherwise it would not be consistent. By symmetry it will thus be

enough to prove that

$$\{z \in| \mathbf{A} \oplus \mathbf{B} || \ z \subseteq \iota_{left}(A')\} = \{\iota_{left}(x - \overline{\emptyset}^A) \mid x \in| \mathbf{A} |\}.$$

Suppose first that z is in the left-hand-side. Let $x = \pi_{left}(z) \cup \overline{\emptyset}^A$; certainly $z = \iota_{left}(x - \overline{\emptyset}^A)$. Claim $x \in| \mathbf{A} |$.

If $X \subseteq^{\text{fin}} x$ then, as $\iota_{left}(X - \overline{\emptyset}^A) \subseteq^{\text{fin}} z$, we have $X - \overline{\emptyset}^A \in Con_A$ by (a) above. But then $X \in Con_A$ by the axioms of information systems, since $X - \overline{\emptyset}^A \vdash_A a$ for each $a \in X \cap \overline{\emptyset}^A$.

Suppose $X \subseteq^{\text{fin}} x$ and $X \vdash_A a$. If $a \in \overline{\emptyset}^A$ then trivially $a \in x$. If $a \in A'$ then, by the same argument as in the last paragraph, $X - \overline{\emptyset}^A \vdash_A X$ and hence $X - \overline{\emptyset}^A \vdash_A a$ by Axiom 5. But $\iota_{left}(X - \overline{\emptyset}^A) \subseteq^{\text{fin}} z$ and, by (b) above, $\iota_{left}(X - \overline{\emptyset}^A) \vdash \iota_{left}(a)$ and hence $\iota_{left}(a) \in z$. Hence $a \in x$ as desired.

This shows that the left-hand-side is contained in the right-hand-side. The converse follows in a very similar way. ∎

Note that, since every $x \in| \mathbf{A} |$ has $\overline{\emptyset}^A \subseteq x$ and similarly for $\mathbf{B}$, the maps $left :| \mathbf{A} |\to| \mathbf{A} \oplus \mathbf{B} |$ and $right :| \mathbf{B} |\to| \mathbf{A} \oplus \mathbf{B} |$ defined

$$\begin{aligned} left(x) &= \iota_{left}(x - \overline{\emptyset}^A) \\ right(x) &= \iota_{right}(x - \overline{\emptyset}^B) \end{aligned}$$

are order preserving injections whose ranges cover $| \mathbf{A} \oplus \mathbf{B} |$ but meet only at $\emptyset$.

The separated sum constructor $+$ can be defined from first principles, like the ones above. However it is easier to define it in terms of $\oplus$ and lifting.

Definition
The *separated sum* $\mathbf{A} + \mathbf{B}$ of two information systems is $\mathbf{A}_\uparrow \oplus \mathbf{B}_\uparrow$. ∎

This works because the coalesced sum identifies the two new bottom elements introduced by lifting, leaving the original domains separate. Since the minimal element of any lifted domain is $\emptyset$, no tokens are removed from $A_\uparrow$ and $B_\uparrow$ in this construction. The following proposition follows easily from Propositions 4.1.1 and 4.1.3. Note that the well-definedness of $+$ follows trivially from that of $\oplus$ and lifting.

4.1.4 PROPOSITION
The elements of the separated sum $| \mathbf{A} + \mathbf{B} |$ are given by the formula:

$$| \mathbf{A} + \mathbf{B} | = \{\emptyset\} \cup \{\iota_{left}(x_\uparrow) \mid x \in| \mathbf{A} |\} \cup \{\iota_{right}(x_\uparrow) \mid x \in| \mathbf{B} |\}$$

and the three parts of the right-hand-side are disjoint. ∎

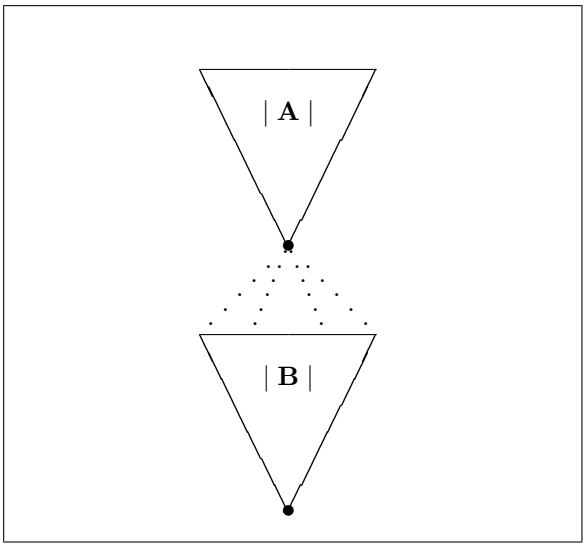

Figure 4.5 | **A** *over* **B** | – see Exercise 4.1.4.

Exercises

4.1.1 If **A** is an information system, define the approximable mappings $lift : \mathbf{A} \to \mathbf{A}_\uparrow$ and $push : \mathbf{A}_\uparrow \to \mathbf{A}$ whose actions on elements are

$$\begin{aligned} |\, lift \,|\,(x) &= x_\uparrow \\ |\, push \,|\,(x_\uparrow) &= x \\ |\, push \,|\,(\emptyset) &= \overline{\emptyset} \quad . \end{aligned}$$

4.1.2 Suppose **A**, **B** and **C** are information systems.

(i) Must $(\mathbf{A} + \mathbf{B}) + \mathbf{C} \cong \mathbf{A} + (\mathbf{B} + \mathbf{C})$?
(ii) Must $(\mathbf{A} \oplus \mathbf{B}) \oplus \mathbf{C} \cong \mathbf{A} \oplus (\mathbf{B} \oplus \mathbf{C})$?

What happens if you replace $\cong$ by $\equiv$?

4.1.3 Give a direct definition of the separated sum constructor $\mathbf{A} + \mathbf{B}$, and show that your definition makes $|\, \mathbf{A} + \mathbf{B} \,|$ order isomorphic to $|\, \mathbf{A}_\uparrow \oplus \mathbf{B}_\uparrow \,|$.

4.1.4 Define a domain constructor *over* so that | **A** *over* **B** | consists (up to isomorphism) of copies of | **A** | and | **B** |, with every element of | **A** | above the whole of | **B** |. (See Figure 4.5).
[Hint: you may well find it helpful to use the token set $A_\uparrow + B$; note that all

sets of B-tokens are consistent since there are elements (those of $\mathbf{A}$) that are greater than all those of $\mathbf{B}$.]

4.1.5 The Cartesian product of two partial orders P and Q can be given another order that is rather finer (in that it orders more things) than the one we have seen so far. This is defined:

$$\langle x, y\rangle \leq \langle x', y'\rangle \Leftrightarrow (x < x') \vee (x = x' \wedge y \leq y')$$

and is known as the *lexicographic* order because it is the order used in dictionaries: sorting is done first on the first component, the second only being used if the first components are equal.

(i) Show that the lexicographic order on the product of two linear orders is also a linear order. (Note that this is not true of the usual product order.)
(ii) Show that the lexicographic order is complete if P and Q are.
(iii) Find examples to show that the lexicographic product of two consistently complete cpos need not be consistently complete.

Is it possible to define a domain constructor which yields the Cartesian product of two information systems so that the result has the lexicographic order?

4.2 Solving equations

We now know what a domain equation like

$$\mathbf{A} \;=\; (\mathbf{A} \times \mathbf{A})_{\uparrow}$$

means: it is simply the statement that the information system $\mathbf{A}$ is a fixed point of a (compound) domain constructor. In other words, $\mathbf{A}$ is actually equal to the lifted version of the Cartesian product of two copies of $\mathbf{A}$. Of course any solution to this equation will have an interesting domain structure: naturally isomorphic to a lifted copy of its own square. In this section we show that this, and all similar equations, have solutions.

Definition
Suppose $\mathbf{A} = \langle A, Con_A, \vdash_A\rangle$ and $\mathbf{B} = \langle B, Con_B, \vdash_B\rangle$ are information systems, then $\mathbf{A} \trianglelefteq \mathbf{B}$ provided

(a) $A \subseteq B$,

(b) $X \in Con_A \;\Leftrightarrow\; X \subseteq A \;\wedge\; X \in Con_B$, and

(c) $X \vdash_A a \;\Leftrightarrow\; X \cup \{a\} \subseteq A \;\wedge\; X \vdash_B a$. ∎

$\mathbf{A} \trianglelefteq \mathbf{B}$ means that the structure of $\mathbf{A}$ is preserved in $\mathbf{B}$. That is, the relationship between $\mathbf{A}$'s tokens in $\mathbf{B}$ is the same as in $\mathbf{A}$. Exactly the same subsets of $\mathbf{A}$'s tokens are consistent in $\mathbf{A}$ and $\mathbf{B}$, and they have the same entailments among themselves. If $\mathbf{A} \trianglelefteq \mathbf{B}$, we say that '$\mathbf{B}$ extends $\mathbf{A}$' or that '$\mathbf{A}$ is a subsystem of $\mathbf{B}$'. ∎

Definition
Suppose $\mathbf{B} = \langle B, Con_B, \vdash_B \rangle$ is an information system and $A \subseteq B$. Define the *restriction of* $\mathbf{B}$ *to* A to be $\mathbf{B}|^{\backslash}A = \langle A, Con_A, \vdash_A \rangle$, where

(a) $X \in Con_A \Leftrightarrow X \subseteq A \wedge X \in Con_B$ and

(b) $X \vdash_A a \Leftrightarrow X \subseteq A \wedge a \in A \wedge X \vdash_B a$. ∎

The following proposition lists some elementary but useful facts about the $\trianglelefteq$ relation. They all follow directly from the definitions of $\trianglelefteq$ and $|^{\backslash}$.

4.2.1 LEMMA

(a) If $\mathbf{B} = \langle B, Con_B, \vdash_B \rangle$ is an information system and $A \subseteq B$, then $\mathbf{B}|^{\backslash}A$ is an information system and $\mathbf{B}|^{\backslash}A \trianglelefteq \mathbf{B}$.

(b) If $\mathbf{A} = \langle A, Con_A, \vdash_A \rangle$ and $\mathbf{B} = \langle B, Con_B, \vdash_B \rangle$ are information systems such that $\mathbf{A} \trianglelefteq \mathbf{B}$, then $\mathbf{A} = \mathbf{B}|^{\backslash}A$.

(c) If $\mathbf{A}$ and $\mathbf{B}$ are as in (b) and $A = B$, then $\mathbf{A} = \mathbf{B}$. ∎

Parts (a) and (b) here observe that there is exactly one subsystem of $\mathbf{B}$ for each subset of its tokens. Part (c) is a special case of (b).

4.2.2 LEMMA
Suppose $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are information systems, then

(a) $\bot = \langle \emptyset, \{\emptyset\}, \emptyset \rangle \trianglelefteq \mathbf{A}$,

(b) $\mathbf{A} \trianglelefteq \mathbf{B} \wedge \mathbf{B} \trianglelefteq \mathbf{A} \Leftrightarrow \mathbf{A} = \mathbf{B}$, and

(c) $\mathbf{A} \trianglelefteq \mathbf{B} \wedge \mathbf{B} \trianglelefteq \mathbf{C} \Rightarrow \mathbf{A} \trianglelefteq \mathbf{C}$.

PROOF
All three parts follow directly from Lemma 4.2.1. For example, suppose $\mathbf{A} \trianglelefteq \mathbf{B} \wedge \mathbf{B} \trianglelefteq \mathbf{C}$. Then necessarily $A \subseteq B \subseteq C$ and, by Lemma 4.2.1(b), $\mathbf{A} = \mathbf{B}|^{\backslash}A$ and

$\mathbf{B} = \mathbf{C}\!\upharpoonright\! B$. It follows immediately that $\mathbf{A} = \mathbf{C}\!\upharpoonright\! A$ and hence $\mathbf{A} \trianglelefteq \mathbf{C}$ by Lemma 4.2.1 (a). ∎

Thus $\trianglelefteq$ satisfies the axioms for a partial order with minimal element $\bot$. The information systems under $\trianglelefteq$ are not a partial order, however, because there are too many of them to be a set (rather, they form a *proper class*). This fine distinction is one we will largely be able to ignore. Note that the *set* of information systems whose tokens are all taken from a given set forms a partial order.

4.2.3 THEOREM
If Δ is a directed set of information systems, then $\mathbf{A}^* = \langle A^*, Con^*, \vdash^* \rangle$ is an information system, where

$$\begin{aligned} A^* &= \bigcup\{A \mid \langle A, Con_A, \vdash_A \rangle \in \Delta\} \\ Con^* &= \bigcup\{Con_A \mid \langle A, Con_A, \vdash_A \rangle \in \Delta\} \\ \vdash^* &= \bigcup\{\vdash_A \mid \langle A, Con_A, \vdash_A \rangle \in \Delta\}. \end{aligned}$$

Furthermore, $\mathbf{A}^*$ is the least upper bound of Δ with respect to $\trianglelefteq$.

PROOF
To prove that $\mathbf{A}^*$ is an information system, Axioms 1-5 should be verified. The first four are trivial and do not use the fact that Δ is directed: consider Axiom 3, for example.

If $X \vdash^* a$ then we know, by definition of $\vdash^*$, that $X \vdash_A a$ for some $\langle A, Con_A, \vdash_A \rangle$ in Δ. But then $X \cup \{a\} \in Con_A$, so that $X \cup \{a\} \in Con^*$, by definition of Con^*, as required.

For Axiom 5, suppose that $X, Y \in Con^*$, that $a \in Y \Rightarrow X \vdash^* a$ and that $Y \vdash^* b$. Since there are only finitely many entailments involved here there must be a finite subset $\mathcal{F}$ of Δ such that each entailment belongs to an element of $\mathcal{F}$. Since Δ is directed it contains an upper bound $\langle A, Con_A, \vdash_A \rangle$ for $\mathcal{F}$. By definition of $\trianglelefteq$ it follows that $X \vdash_A b$ and hence that $X \vdash^* b$ by definition of $\langle A^*, Con^*, \vdash^* \rangle$, as required.

Suppose that $\mathbf{A} \in \Delta$. Claim that $\mathbf{A} \trianglelefteq \mathbf{A}^*$. Trivially $A \subseteq A^*$, $Con_A \subseteq Con^*$ and $X \vdash_A a \Rightarrow X \vdash^* a$. It only remains to prove the reverse implications of the second and third clauses in the definition of $\trianglelefteq$.

So suppose $X \in Con^*$ and $X \subseteq A$. The definition of Con^* means there is $\mathbf{B} \in \Delta$ such that $X \in Con_B$. Then, as Δ is directed, there is some $\mathbf{C} \in \Delta$ such that $\mathbf{A} \trianglelefteq \mathbf{C}$ and $\mathbf{B} \trianglelefteq \mathbf{C}$. $X \in Con_C$ because $\mathbf{B} \trianglelefteq \mathbf{C}$, but then $X \in Con_A$ as $X \subseteq A$ and $\mathbf{A} \trianglelefteq \mathbf{C}$.

If $X \vdash^* a$ and $X \cup \{a\} \subseteq A$, then there is some $\mathbf{B} \in \Delta$ such that $X \vdash_B a$. The fact that $X \vdash_A a$ follows, as in the last paragraph, from the existence of $\mathbf{C} \in \Delta$

such that $\mathbf{A} \trianglelefteq \mathbf{C}$ and $\mathbf{B} \trianglelefteq \mathbf{C}$.

This completes the proof that $\mathbf{A} \trianglelefteq \mathbf{A}^*$. This means that $\mathbf{A}^*$ is *an* upper bound for Δ. If $\mathbf{A}'$ were another then trivially

$$A' \supseteq \bigcup\{A \mid \langle A, Con_A, \vdash_A\rangle \in \Delta\} = A^* .$$

For any finite subset Y of A^* we can find an element $\mathbf{B}$ of Δ such that $Y \subseteq B$ and thus $\mathbf{A}^*\!\upharpoonright\! Y = \mathbf{B}\!\upharpoonright\! Y$ and $\mathbf{A}'\!\upharpoonright\! Y = \mathbf{B}\!\upharpoonright\! Y$ (as $\mathbf{B} \trianglelefteq \mathbf{A}^*$ and $\mathbf{B} \trianglelefteq \mathbf{A}'$ respectively). Thus (respectively choosing $Y = X$ and $Y = X \cup \{a\}$) we have

- $X \in Con^* \Leftrightarrow X \subseteq A^* \wedge X \in Con'$, and
- $X \vdash^* a \Leftrightarrow X \cup \{a\} \subseteq A^* \wedge X \vdash' a$,

which establishes that $\mathbf{A}^* \trianglelefteq \mathbf{A}'$. Thus $\mathbf{A}^*$ is, as claimed, the *least* upper bound of Δ. ∎

This result says that the 'partial order' of information systems is complete.

One would expect that, when one system extends another in the strict way meant by $\trianglelefteq$, there would be a natural relationship between the corresponding domains of elements. This is in fact so, as is shown by the following proposition which shows there is a natural embedding of $|\ \mathbf{A}\ |$ into $|\ \mathbf{B}\ |$.

4.2.4 LEMMA
If $\mathbf{A} \trianglelefteq \mathbf{B}$, then the maps $\rightarrowtail : |\ \mathbf{A}\ | \to |\ \mathbf{B}\ |$ and $\pi : |\ \mathbf{B}\ | \to |\ \mathbf{A}\ |$ defined

$$\begin{aligned} \rightarrowtail(x) &= \overline{x} \quad \text{(deductive closure in } \mathbf{B}) \\ \pi(x) &= x \cap A \end{aligned}$$

are well-defined and continuous. Furthermore $\pi \circ \rightarrowtail = id_{|\mathbf{A}|}$ (the identity map on $|\ \mathbf{A}\ |$) and $\rightarrowtail \circ \pi(y) \subseteq y$ for each $y \in |\ \mathbf{B}\ |$.

PROOF
That $\rightarrowtail$ is well-defined follows from Lemma 3.1.3 and the fact that any element x of $|\ \mathbf{A}\ |$ is trivially consistent in $\mathbf{B}$, as the relationships of elements of A are the same in $\mathbf{A}$ as in $\mathbf{B}$.

That $\pi(y)$ is consistent is trivial. That it is deductively closed follows since, if $a \in A$, $X \subseteq^{\text{fin}} x \cap A$ and $X \vdash_A a$, $X \vdash_B a$ by definition of $\trianglelefteq$ and hence $a \in x$, which trivially implies $a \in \pi(x)$.

The monotonicity of $\rightarrowtail$ and π is trivial, as is the continuity of π. If $\Delta \subseteq |\ \mathbf{A}\ |$ is directed then $\rightarrowtail(\bigcup\Delta) \supseteq \bigcup\{\rightarrowtail(x) \mid x \in \Delta\}$ by monotonicity. If $b \in \rightarrowtail(\bigcup\Delta)$ then there is some $X \subseteq^{\text{fin}} \bigcup\Delta$ such that $X \vdash_B b$; but by directedness of Δ there is then $x \in \Delta$ such that $X \subseteq^{\text{fin}} x$ and hence $b \in \rightarrowtail(x) \subseteq \bigcup\{\rightarrowtail(x) \mid x \in \Delta\}$. Thus $\rightarrowtail$ is continuous.

If $x \in |\ \mathbf{A}\ |$, trivially $x \subseteq \pi(\rightarrowtail(x))$. If $a \in \pi(\rightarrowtail(x))$ then $a \in A$ and there exists $X \subseteq^{\text{fin}} x$ such that $X \vdash_B a$. But then $X \vdash_A a$ by definition of $\trianglelefteq$ so that $a \in x$. Hence $x = \pi(\rightarrowtail(x))$ as required. Thus $\rightarrowtail$ is an order preserving injection from $|\ \mathbf{A}\ |$ to $|\ \mathbf{B}\ |$.

If $y \in |\ \mathbf{B}\ |$ then $\rightarrowtail(\pi(x)) = \overline{y \cap A}$, which is easily seen to be a subset of $y\ (= \overline{y})$. ∎

The injections and projections described in Lemma 4.2.4 tie in well with the idea that tokens are propositions about elements. When $\mathbf{A} \trianglelefteq \mathbf{B}$ we know that B is a richer language of propositions than A, but that the elements of A retain their 'meaning' in B. To inject an element from $|\ \mathbf{A}\ |$ to $|\ \mathbf{B}\ |$ we simply take all the propositions in the richer language which are entailed by what we know about the element from A. (B can easily contain new tokens which are weaker than some elements of A.) To project an element from $|\ \mathbf{B}\ |$ to $|\ \mathbf{A}\ |$ we just retain those propositions true of it which are in the more restricted language.

These injections and projections are extremely well-behaved and have many useful properties. The following lemma records a few of these, all of which are easy to prove.

4.2.5 LEMMA
If $\mathbf{A} \trianglelefteq \mathbf{B} \trianglelefteq \mathbf{C}$ then, if $\rightarrowtail_{XY}$ and π_{XY} represent the injection and projection functions from $|\ \mathbf{X}\ |$ to $|\ \mathbf{Y}\ |$ for suitable $\mathbf{X}, \mathbf{Y}$, the following identities all hold:

$$\rightarrowtail_{BC} \circ \rightarrowtail_{AB} = \rightarrowtail_{AC}, \qquad \pi_{BA} \circ \pi_{CB} = \pi_{CA}, \qquad \pi_{CB} \circ \rightarrowtail_{AC} = \rightarrowtail_{AB}\ .$$ ∎

Given a directed set Δ of information systems, we have $\mathbf{A} \trianglelefteq \bigsqcup \Delta$ for all $\mathbf{A} \in \Delta$. Thus there are projection and injection maps between the limit of Δ and each of its elements. Also, because Δ is directed, there is a complex system of projections and injections between its elements. The following result shows how the structure of $|\ \bigsqcup \Delta\ |$ is determined by these.

4.2.6 THEOREM
Suppose Δ is a directed set of information systems under $\trianglelefteq$ with limit $\mathbf{A}^* = \langle A^*, Con^*, \vdash^* \rangle$ and, for each $\mathbf{A} \in \Delta$, $\rightarrowtail^*_A$ and π^*_A are the injection and projection between $|\ \mathbf{A}\ |$ and $|\ \mathbf{A}^*\ |$. Then, whenever $x \in |\ \mathbf{A}^*\ |$,

$$x\ =\ \bigcup \{ \rightarrowtail^*_A (\pi^*_A(x))\ \mid\ \mathbf{A} \in \Delta \}\,.$$

The π_A realise a natural correspondence between $|\ \mathbf{A}^*\ |$ and the set of all *cones* of elements from Δ: choices of an $x_A \in |\ \mathbf{A}\ |$ for each $\mathbf{A} \in \Delta$ such that $\pi_{BA}(x_B) = x_A$ whenever $\mathbf{A} \trianglelefteq \mathbf{B}$. Each cone maps to its union (which is always an element of $|\ \mathbf{A}^*\ |$) and each $x \in |\ \mathbf{A}^*\ |$ maps to $\{\pi^*_A(x) \mid \mathbf{A} \in \Delta\}$; these maps being inverses.

Furthermore, if $x, y \in| \mathbf{A}^* |$ then $x \subseteq y$ if and only if $\forall \mathbf{A} \in \Delta. \pi^*_A(x) \subseteq \pi^*_A(y)$. Thus the order on $| \mathbf{A}^* |$ is determined by the natural order on the cones.

PROOF
The first part of this result is easy, for we know by Lemma 4.2.4 that $\rightarrowtail^*_A (\pi^*_A(x)) \subseteq x$ whenever $x \in| \mathbf{A}^* |$ and $\mathbf{A} \in \Delta$. And, since $A^* = \bigcup\{A \mid \langle A, Con, \vdash\rangle \in \Delta\}$,

$$x = \bigcup\{x \cap A \mid \langle A, Con, \vdash\rangle \in \Delta\} \subseteq \bigcup\{\rightarrowtail^*_A (\pi^*_A(x)) \mid \mathbf{A} \in \Delta\},$$

which completes this part of the proof.

It is an immediate consequence of Lemma 4.2.5 that, if $x \in| \mathbf{A}^* |$, then the vector $\langle \pi_A(x) \mid \mathbf{A} \in \Delta\rangle$ is a cone, since if $\mathbf{A}, \mathbf{B} \in \Delta$ and $\mathbf{A} \trianglelefteq \mathbf{B}$ then $\pi^*_A = \pi_{BA} \circ \pi^*_B$. And since the union of this cone is x (by the same argument as in the last paragraph), the cone uniquely determines x.

Given any cone $\langle x_A \mid \mathbf{A} \in \Delta\rangle$, claim that the union $x^* = \bigcup\{x_A \mid \mathbf{A} \in \Delta\}$ has the property that $x^* \cap A = x_A$ for all $\mathbf{A} \in \Delta$. That $x_A \subseteq x^* \cap A$ is trivial; and if $a \in x^* \cap A$ then there must be $\mathbf{B} \in \Delta$ such that $a \in x_B$. Since Δ is directed there is $\mathbf{C} \in \Delta$ such that $\mathbf{A}, \mathbf{B} \trianglelefteq \mathbf{C}$. Since $\pi_{CA}(x_C) = x_A$ and $\pi_{CB}(x_C) = x_B$ it follows that $a \in x_A$, completing the proof that $x_A = x^* \cap A$.

x^* is consistent in $\mathbf{A}^*$ since, given any $X \subseteq^{\text{fin}} x^*$, there is $\mathbf{A} \in \Delta$ such that $X \subseteq A$. By the last paragraph we know that $X \subseteq^{\text{fin}} x_A$ and hence $X \in Con_A \subseteq Con^*$. It is deductively closed by a very similar argument. The last paragraph shows that a cone is determined by its union, so union is a well-defined injective map from cones to $| \mathbf{A}^* |$. We have already seen that it is the inverse of the map from $| \mathbf{A}^* |$ to cones.

Given these maps, the last part of the theorem (that the order on $| \mathbf{A}^* |$ is determined by the order on the cones) follows immediately. ∎

This result is invaluable in working out the partial order structure of the limits of directed sets of domains, since it makes it possible to deduce these without working out the full structure of the limit system itself. We shall see this in the next chapter when we examine the solutions to some domain equations in detail.

EXAMPLE
It is very easy to see that the systems $\{ \perp_n \mid n \in \mathbb{N}\}$ of Figure 4.2 (page 85) form a directed set, with $\perp_n \trianglelefteq \perp_m$ when $n \leq m$. The limit of this directed set is $\perp_\omega = \langle A, Con, \vdash\rangle$, where

(a) $A = \{a_0, a_1, a_2, \ldots\}$ (as defined in the example on page 84),

(b) $X \in Con$ for all finite subsets of A and

(c) $X \vdash a_n \Leftrightarrow X$ contains an a_m with $m \geq n$.

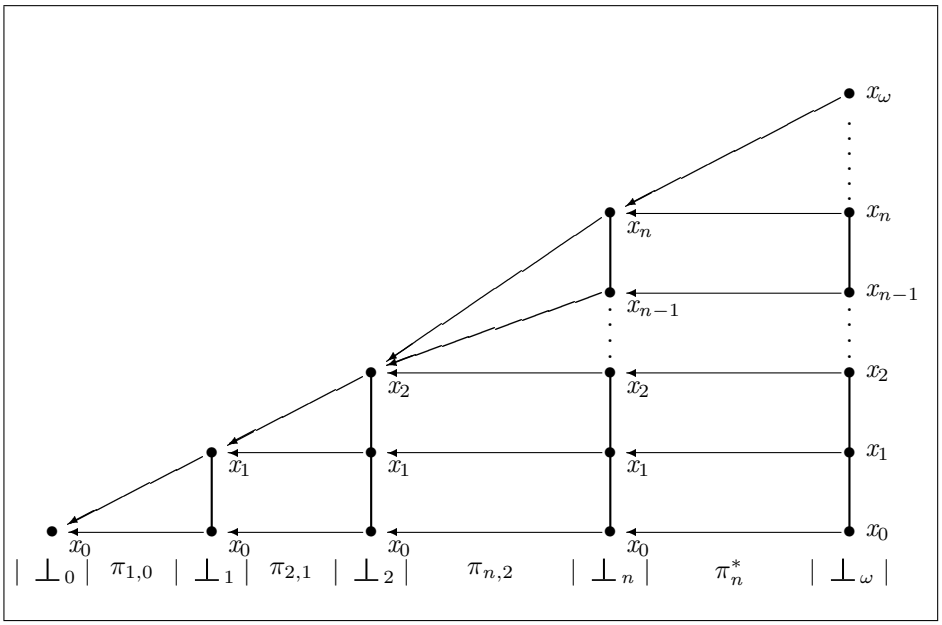

Figure 4.6 Projection functions

It is easy to see that $\mid \perp_\omega \mid$ is just $\{x_0, x_1, x_2, \ldots\} \cup \{x_\omega\}$, where $x_n = \{a_i \mid i < n\}$ and $x_\omega = A$, for any subset of A is consistent, and it will be deductively closed precisely when the indices of the a_i it contains are left-closed. Thus, in this case, the limit domain consists of all the elements of the limiting domains together with a single 'limit' element.

Denote the injections and projections between these domains by $\rightarrowtail_{m,n}$, $\pi_{m,n}$, $\rightarrowtail_n^*$ and π_n^*, each having the obvious meaning. All the injection functions are the identity (onto a subset of their codomains) because, in this example, the only tokens in a large domain that are not in a small one are strictly stronger than those of the small one. Figures 4.6 and 4.7 illustrate the projections and injections respectively.

Notice that the injection maps between $\mid \perp_k \mid$ and $\mid \perp_{k+1} \mid$ are different from the ones induced by lifting ($\perp_{k+1} = (\perp_k)_\uparrow$). This is because the construction of $\pi_{k+1,k}$ and $\rightarrowtail_{k,k+1}$ does not use the fact that $\perp_k$'s tokens are lifted in producing $\perp_{k+1}$. They see the way the domains look, not how they were put together.

It is easy to see from Figure 4.6 how the elements of $\mid \perp_\omega \mid$ correspond to cones of elements from the approximating domains: x_ω corresponds to $\langle x_0, x_1, \ldots, x_k, \ldots\rangle$, while x_k corresponds to the eventually constant sequence $\langle x_0, x_1, \ldots, x_{k-1}, x_k, x_k, x_k, \ldots\rangle$. ∎

Recall the definitions of *monotonic* and *continuous* functions from the Chap-

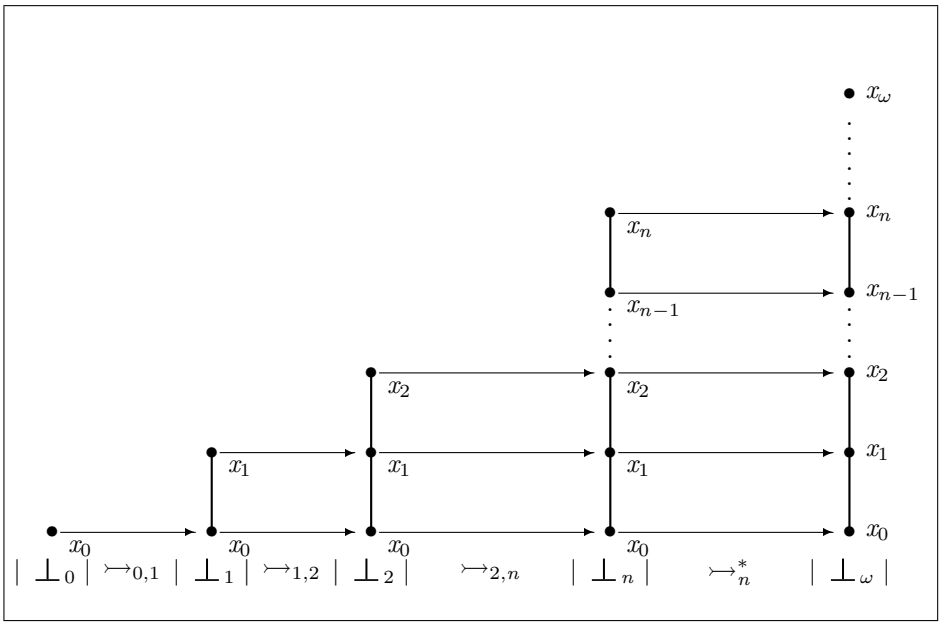

Figure 4.7 Injection functions

ter 2. These definitions apply equally to domain constructors. We extend the relation $\trianglelefteq$ to tuples of information systems in the obvious way: $\langle \mathbf{A}_1, \ldots, \mathbf{A}_n \rangle \trianglelefteq \langle \mathbf{A}'_1, \ldots, \mathbf{A}'_n \rangle$ if and only if $\mathbf{A}_1 \trianglelefteq \mathbf{A}'_1 \wedge \mathbf{A}_2 \trianglelefteq \mathbf{A}'_2 \wedge \ldots \wedge \mathbf{A}_n \trianglelefteq \mathbf{A}'_n$.

Definition
The n to m domain constructor F is *monotonic* if, whenever $\underline{\mathbf{A}}$ and $\underline{\mathbf{A}}'$ are two n-tuples such that $\underline{\mathbf{A}} \trianglelefteq \underline{\mathbf{A}}'$

$$F(\underline{\mathbf{A}}) \trianglelefteq F(\underline{\mathbf{A}}') \,.$$

Notice that, if Δ is directed and F is monotonic, then $\{F(\mathbf{A}) \mid \mathbf{A} \in \Delta\}$ is directed.

Definition
The n to m domain constructor F is *continuous* if, whenever Δ is a directed set of n-tuples of information systems,

$$F(\bigsqcup \Delta) = \bigsqcup \{F(\underline{\mathbf{A}}) \mid \underline{\mathbf{A}} \in \Delta\} \,.$$

We will see shortly that all the domain constructors given so far are monotonic and continuous.

All the expected results about composing monotonic and continuous constructors hold, the proofs being identical to those for functions over true partial orders.

4.2.7 LEMMA
Suppose F and G are respectively n to m and m to k domain constructors.

(a) If F and G are both monotonic then so is $G \circ F$.

(b) If F and G are both continuous then so is $G \circ F$.

(c) $F(X_1, X_2, \ldots, X_n)$ is monotonic (continuous) if, and only if, it is monotonic (continuous) in each component of its argument separately.

(d) $F(\underline{X}) = \langle F_1(\underline{X}), \ldots, F_m(\underline{X})\rangle$ is monotonic (continuous) if, and only if, each of its component functions F_i is. ∎

Thus, assuming that all basic domain constructors are continuous, we know that all compositions of them are.

The following is the crucial result which allows us to solve domain equations.

4.2.8 THEOREM
If F is a continuous unary domain constructor, then there is an information system $\mathbf{A}$ such that

(a) $\mathbf{A} = F(\mathbf{A})$, and

(b) if $\mathbf{B}$ is any information system such that $\mathbf{B} = F(\mathbf{B})$, then $\mathbf{A} \trianglelefteq \mathbf{B}$.

PROOF
The proof of this theorem is essentially the same as that of Theorem 2.2.5, but we include it here in full because of its great importance[3].

Consider the systems $\bot$, $F(\bot)$, $F^2(\bot), \ldots, F^n(\bot), \ldots$. Since $\bot$ is minimal among the information systems, we know that $\bot \trianglelefteq F(\bot)$. Suppose for induction that $F^k(\bot) \trianglelefteq F^{k+1}(\bot)$. Then, applying F to both sides, $F^{k+1}(\bot) \trianglelefteq F^{k+2}(\bot)$ by monotonicity, proving this result for all k. Transitivity of $\trianglelefteq$ (Lemma 4.2.2 (c)) then shows, for all $n, m \in \mathbb{N}$,

$$n \leq m \;\Rightarrow\; F^n(\bot) \trianglelefteq F^m(\bot)$$

[3]However the stronger result proved in Section 2.4, namely that *monotone* functions on complete partial orders have least fixed points, does *not* have an analogue here. This is because the information systems are a proper class and so the iteration $F^\alpha(\bot)$ need never reach a fixed point.

so that $\{F^n(\perp) \mid n \in \mathbb{N}\}$ is a directed set. Continuity of F gives

$$\begin{aligned} F(\bigsqcup\{F^n(\perp) \mid n \in \mathbb{N}\}) &= \bigsqcup\{F(F^n(\perp)) \mid n \in \mathbb{N}\} \\ &= \bigsqcup\{F^{n+1}(\perp) \mid n \in \mathbb{N})\} \\ &= \bigsqcup\{F^n(\perp) \mid n \in \mathbb{N})\} \end{aligned}$$

(Because $\bigsqcup\Delta = \bigsqcup(\Delta \cup \{\perp\})$ for any directed Δ.) This means that $\mathbf{A} = \bigsqcup\{F^n(\perp) \mid n \in \mathbb{N}\}$ is a fixed point of F. If $\mathbf{B}$ is another fixed point then

$\perp \trianglelefteq \mathbf{B}$ by minimality of $\perp$, and

$$\begin{aligned} F^n(\perp) \trianglelefteq \mathbf{B} \Rightarrow \quad F^{n+1}(\perp) &\trianglelefteq F(\mathbf{B}) && \text{by monotonicity} \\ &= \mathbf{B} && \text{as } \mathbf{B} \text{ is a fixed point.} \end{aligned}$$

By induction, then, $F^n(\perp) \trianglelefteq \mathbf{B}$ for all n. $\mathbf{B}$ is thus an upper bound for $\{F^n(\perp) \mid n \in \mathbb{N}\}$: since $\mathbf{A}$ is the *least* upper bound we know $\mathbf{A} \trianglelefteq \mathbf{B}$ as required. ∎

This proposition is easily generalised to let us solve mutual domain equations such as

$$\begin{aligned} \mathbf{A} &= \mathbf{A} + \mathbf{B} \\ \mathbf{B} &= (\mathbf{A} \times \mathbf{B})_\uparrow . \end{aligned}$$

The proof is essentially the same (using n to n constructors rather than 1 to 1).

4.2.9 THEOREM
If F is a continuous n to n domain constructor, then there is an n-tuple $\underline{\mathbf{A}}$ of information systems such that

(a) $\underline{\mathbf{A}} = F(\underline{\mathbf{A}})$, and

(b) if $\underline{\mathbf{B}}$ is any n-tuple such that $\underline{\mathbf{B}} = F(\underline{\mathbf{B}})$, then $\underline{\mathbf{A}} \trianglelefteq \underline{\mathbf{B}}$. ∎

EXAMPLE
Assuming for the moment that lifting is continuous, the systems $\perp_n$ of Figure 4.2 (page 85) are exactly the iterates used by the above proposition in solving the domain equation $\mathbf{A} = \mathbf{A}_\uparrow$. Therefore their limit $\perp_\omega$ is the minimal solution to the equation $\mathbf{A} = \mathbf{A}_\uparrow$. This is easy to check.

The token set of $\perp_\omega$ is $\{a_0, a_1, a_2, \ldots\}$. Hence the token set of $(\perp_\omega)_\uparrow$ is

$$\{lifted\} \cup \{\langle old, a_0\rangle, \langle old, a_1\rangle, \ldots\} = \{a_0\} \cup \{a_1, a_2, \ldots\},$$

which is the same. Because any finite subset of $\bot_\omega$'s tokens are consistent, the same is true of $(\bot_\omega)_\uparrow$ (so the two systems have the same consistency relation). Finally, from the definition of lifting, $X \vdash a$ in $(\bot_\omega)_\uparrow$ if and only if

$$a = a_0 \wedge X \neq \emptyset, \quad \text{or}$$
$$a = \langle old, a_n \rangle\ (= a_{n+1}) \wedge$$
$$\exists\, m \geq n . a_m \in \pi_\uparrow(X)\ (= \{a_r \mid a_{r+1} \in X\})\ .$$

This is easily seen to be the same relation as in $\bot_\omega$. ∎

There is still one important thing left to do, namely to justify our claim that the existing constructors are all continuous. This will be made rather easier by the following definition and lemma, which allow us to avoid some unnecessary work.

Definition
The monotonic, unary domain constructor F is said to be *continuous on token sets* if, whenever Δ is a directed set of information systems, the token set of $F(\bigsqcup \Delta)$ is equal to the union of those of $\{F(\mathbf{A}) \mid \mathbf{A} \in \Delta\}$. ∎

4.2.10 LEMMA
The unary domain constructor F is continuous if and only if it is monotonic and is continuous on token sets.

PROOF
Suppose Δ is directed. Then $F(\bigsqcup \Delta) \trianglerighteq \bigsqcup\{F(\mathbf{A}) \mid \mathbf{A} \in \Delta\}$ by the usual argument from monotonicity. But now these two information systems are ordered but (by continuity on token sets) have the same set of tokens. Lemma 4.2.1 (c) implies they are in fact equal, as required. ∎

4.2.11 PROPOSITION
Lifting, $\times$, $\oplus$ and $+$ are all continuous.

PROOF
By virtue of Lemmas 4.2.10 and 4.2.7 it is sufficient to prove that all domain constructors are, in each argument separately, monotonic and continuous on token sets. The proofs of these results are all very alike. We present here only the most interesting one, namely that for coalesced sum. By symmetry it will clearly be sufficient to consider only the left-hand argument.
Monotonicity. Suppose $\mathbf{A}$ is any information system and that $\mathbf{B} \trianglelefteq \mathbf{C}$. We must show that $\mathbf{D} \trianglelefteq \mathbf{E}$, where $\mathbf{D} = \mathbf{B} \oplus \mathbf{A}$ and $\mathbf{E} = \mathbf{C} \oplus \mathbf{A}$.

The token set of $\mathbf{D}$ is $B' + A'$, where $B' = B - \overline{\emptyset}^B$ and $A' = A - \overline{\emptyset}^A$. Now, if

$b \in \overline{\emptyset}^C \cap B$, then $\emptyset \vdash_C b$ and hence $\emptyset \vdash_B b$ (as $\emptyset \cup \{b\} \subseteq B$) and so $b \in \overline{\emptyset}^B$. Hence, if $C' = C - \overline{\emptyset}^C$, $C' \supseteq B'$, and so by definition of $+$ (on sets),

$$B' + A' \subseteq C' + A'$$

which is the first thing we must prove for monotonicity.

Let $Con'_A = \{X \in Con_A \mid X \subseteq A'\}$, and similarly for Con'_B and Con'_C. The definitions of $\mathbf{B} \oplus \mathbf{A}$ and $\trianglelefteq$ then give

$$\begin{array}{rcl} Con_D & = & \{\iota_{left}(X) \mid X \in Con'_B\} \cup \{\iota_{right}(X) \mid X \in Con'_A\} \\ & = & \{\iota_{left}(X) \mid X \in Con'_C \wedge X \subseteq B'\} \cup \\ & & \{\iota_{right}(X) \mid X \in Con'_A\} \\ & = & (\{\iota_{left}(X) \mid X \in Con'_C\} \cap \\ & & \quad \{Y \subseteq \iota_{left}(C') \mid \pi_{left}(Y) \subseteq^{\text{fin}} B'\}) \\ & & \cup \{\iota_{right}(X) \mid X \in Con'_A\} \\ & = & \{X \in Con_E \mid X \subseteq D\} \end{array}$$

which is what is required of Con_D and Con_E. A similar argument shows that $X \vdash_D d \Leftrightarrow (X \cup \{d\} \subseteq D \wedge X \vdash_E d)$, completing the proof.

Continuity on token sets. It is clear that, whenever A is a set and D a set of sets, then $(\bigcup D) + A = \bigcup\{B + A \mid B \in D\}$. Thus to prove continuity on token sets it will be enough to show that, whenever Δ is a directed set of information systems with limit $\mathbf{C} = \bigsqcup \Delta$, $\overline{\emptyset}^C = \bigcup\{\overline{\emptyset}^B \mid \mathbf{B} \in \Delta\}$. This itself follows easily from the construction of the limit in Theorem 4.2.3: the entailment relation of the limit is just the union of the entailment relations of the approximations and so

$$\{b \mid \emptyset \vdash_C b\} = \bigcup\{\{b \mid \emptyset \vdash_B b\} \mid \mathbf{B} \in \Delta\},$$

which is exactly what we needed. ∎

We introduced two general Cartesian product constructors earlier:

$$\mathbf{A}^X \qquad \text{and} \qquad \prod_{\lambda \in \Lambda} \mathbf{A}_\lambda$$

which respectively denote X copies of a single information system $\mathbf{A}$ and the product of many, possibly different, $\mathbf{A}_\lambda$. As we observed earlier, the second is essentially just a generalisation of the first. But actually, when we come to prove their monotonicity and continuity, they are rather different. $\mathbf{A}^X$ is (for any X) a unary constructor, while the general product maps vectors indexed by Λ to a single information system.

The case of $\mathbf{A}^X$ is straightforward, the proof following exactly the same pattern as before. The general product is a little more interesting, since (as was observed in Section 2.3) it is not sufficient to prove it monotone and continuous in

each argument separately. However it is monotone and continuous – see Exercise 2.2.11.

There are some exercises on the solution to domain equations below. The reader who has difficulty solving them now would do well to return to them after reading the first section of the next chapter. There we will see some examples of the solutions to more complex domain equations – this will both let us see how our theory works out in practice and will develop various techniques that are useful in understanding the structures of the solutions that our theory gives. This is necessary both when we want to know how some solution domain looks and when we want to find the 'right' domain equation to give us some particular result when we solve it.

Exercises

4.2.1 If $\mathbf{A} = \langle A, Con_A, \vdash_A \rangle$ is an information system, show that $\mathbf{A}^{\backslash}(A - \overline{\emptyset}) \cong \mathbf{A}$.

4.2.2 Prove that the product constructor $\times$ on information systems is monotonic and continuous.

4.2.3 Describe the partial order structures of the minimal solutions of the following domain equations:

(i) $\mathbf{D} = \perp \oplus \mathbf{D}$
(ii) $\mathbf{D} = \perp + \mathbf{D}$
(iii) $\mathbf{D} = \perp \times \mathbf{D}$
(iv) $\mathbf{D} = \mathbf{A} + \mathbf{D}$
(v) $\mathbf{D} = \perp_{\uparrow} \times \mathbf{D}$

($\mathbf{A}$ being any fixed information system.)

4.2.4 Suppose that F and G are continuous unary domain constructors and that $\mathbf{A}$ and $\mathbf{B}$ are respectively the least solutions of the domain equations

$$\mathbf{A} = F(G(\mathbf{A})) \quad \text{and} \quad \mathbf{B} = G(F(\mathbf{B})).$$

Prove that $\mathbf{A} = F(\mathbf{B})$.

4.2.5 Say that two tokens a, b of the information system $\mathbf{A}$ are *equivalent* if $a \in x \Leftrightarrow b \in x$ for all $x \in |\mathbf{A}|$.

(i) Prove that a and b are equivalent if and only if $\overline{\{a\}} = \overline{\{b\}}$.
(ii) Show that there is an information system $\mathbf{A}'$ such that $\mathbf{A}' \trianglelefteq \mathbf{A}$, $\mathbf{A}' \cong \mathbf{A}$ and $\mathbf{A}'$ has no pairs of equivalent tokens.

4.2.6 Suppose $\mathbf{A} \trianglelefteq \mathbf{B} \trianglelefteq \mathbf{C}$ and $\mathbf{A} \cong \mathbf{C}$. Need $\mathbf{A} \cong \mathbf{B}$? What would the answer have been if $\cong$ had been replaced by $\equiv$?

4.2.7 Suppose that $\mathbf{A} \trianglelefteq \mathbf{B}$ and that $\mathbf{B}'$ is a system isomorphic to $\mathbf{B}$ via the bijection $\theta : B' \to B$. If $A' \subseteq B'$ and $\mathbf{A}'$ is isomorphic to $\mathbf{A}$ via $\theta|\!\setminus A'$, prove that $\mathbf{A}' \trianglelefteq \mathbf{B}'$.

4.2.8 If $\mathbf{A}$ is an arbitrary information system, determine the partial order structure of the least solution of the domain equation

$$\mathbf{D} = \mathbf{A} \oplus \mathbf{D} .$$

Hence or otherwise find a domain equation whose least solution has the form of (countably) infinitely many disjoint and incomparable copies of $|\ \mathbf{A}\ |$ above a single bottom element.

4.2.9 Define a continuous domain constructor $\hat{\mathbf{A}}$ whose effect is to place an extra top (maximum) element above the elements of $|\ \mathbf{A}\ |$. (It is thus the opposite of lifting in some sense.) Determine the structures of the least solutions to the domain equations

$$\mathbf{D} = \hat{\mathbf{D}} \qquad \text{and} \qquad \mathbf{E} = (\hat{\mathbf{E}})_{\uparrow} .$$

4.2.10 Recall the constructor *over* defined in Exercise 4.1.?. Consider the domain equation

$$\mathbf{C} = \mathbf{C}\ \mathit{over}\ \mathbf{C} .$$

If $\mathbf{C}_0 = \ \perp$ and $\mathbf{C}_{n+1} = \mathbf{C}_n \mathit{over} \mathbf{C}_n$, show that $|\ \mathbf{C}_n\ |$ has exactly 2^n elements and that, the projection $\pi_{n+1,n}$ maps exactly two elements of $|\ \mathbf{C}_{n+1}\ |$ to each element of $|\ \mathbf{C}_n\ |$. Hence prove that $|\ \mathbf{C}\ |$ is order-isomorphic to the Cantor set. [The Cantor set consists of all real numbers in $[0, 1]$ with a ternary (base 3) expansion consisting only of 0s and 2s. It is what is left if the open middle third of $[0, 1]$ is removed, then the middle thirds of the two remaining intervals, and so on for ever.]

4.2.11 Recall the definition of the general Cartesian product on page 88. Show that it is monotone, in other words, if $\underline{\mathbf{A}}$ and $\underline{\mathbf{B}}$ are two Λ-vectors of information systems such that $\forall\, \lambda \in \Lambda . \mathbf{A}_\lambda \trianglelefteq \mathbf{B}_\lambda$, then

$$\textstyle\prod_{\lambda \in \Lambda} \mathbf{A}_\lambda \trianglelefteq \prod_{\lambda \in \Lambda} \mathbf{B}_\lambda .$$

Show also that it is continuous on token sets and hence continuous (being careful to justify this last step, since it is not explicitly covered by Lemma 4.1.10).

4.3 The function space constructor

Cantor's theorem[4] tells us we will have to compromise in our choice of function space constructor: there can be no hope of our finding a continuous constructor such that $|\ \mathbf{A} \to \mathbf{B}\ |$ consists (up to isomorphism) of all the functions from $|\ \mathbf{A}\ |$ to $|\ \mathbf{B}\ |$ for, by the theory above,

$$\mathbf{A} = \ \bot + (\mathbf{A} \to \mathbf{A})$$

would have a solution! Our compromise is only to represent those functions which it is reasonable to represent as computable: ones where every piece of information about $f(x)$ is deducible from a finite amount of information about x.

Thus the functions we want are the continuous ones and the tokens we use are, for $\mathbf{A} \to \mathbf{B}$, $Con_A \times B$. Fortunately we already know a great deal about how continuous functions are represented by this sort of token: we can expect an element of $|\ \mathbf{A} \to \mathbf{B}\ |$ to be an approximable mapping from $\mathbf{A}$ to $\mathbf{B}$. We must devise a suitable consistency relationship and entailment relation to make this happen. This is done as follows.

Definition
If $\mathbf{A}$ and $\mathbf{B}$ are information systems, then $\mathbf{A} \to \mathbf{B} = \langle Con_A \times B, Con, \vdash \rangle$, where

- $\{\langle X_i, b_i\rangle \mid 1 \le i \le n\} \in Con \Leftrightarrow \forall\, F \subseteq \{1, \ldots, n\}.\ \bigcup\{X_i \mid i \in F\} \in Con_A$
  $\Rightarrow\ \{b_i \mid i \in F\} \in Con_B$
- $\{\langle X_i, b_i\rangle \mid 1 \le i \le n\} \vdash \langle X, b\rangle \ \Leftrightarrow\ \{b_i \mid X \vdash_A X_i\} \vdash_B b$. ∎

The first clause here says that a finite set of tokens is consistent if, whenever a subset have consistent left-hand sides (and so can be telling us information about the application of the function to a single argument), the information they tell us about the result (their right-hand sides) must be consistent as well. The next proposition tells us that the elements of $|\ \mathbf{A} \to \mathbf{B}\ |$ are familiar under another name.

4.3.1 PROPOSITION
If $\mathbf{A}$ and $\mathbf{B}$ are information systems then so is $\mathbf{A} \to \mathbf{B}$, $\to$ is a continuous domain constructor and furthermore $|\ \mathbf{A} \to \mathbf{B}\ |$ is precisely the set of approximable mappings from $\mathbf{A}$ to $\mathbf{B}$.

---

[4]Cantor's theorem states that there cannot be a surjective map from any set to its powerset. Since, if B is a set with at least two elements (b, b', say), there is a surjective map from the full function space $A \to B$ to $\mathcal{P}(A)$ given by $\psi(f) = \{a \mid f(a) = b\}$, there can be no surjective map from A to $A \to B$.

PROOF

Axioms 1, 2 and 4 are trivial.

For Axiom 3 we must show that if $\{\langle X_1, b_1\rangle, \ldots, \langle X_n, b_n\rangle\} \vdash \langle X, b\rangle$, then necessarily $\{\langle X_1, b_1\rangle, \ldots, \langle X_n, b_n\rangle, \langle X, b\rangle\} \in Con$.

Suppose then that $X \cup \bigcup\{X_i \mid i \in F\} \in Con_A$. Then, by Lemma 3.1.2 and Axiom 1 of $\mathbf{A}$, $\bigcup\{X_i \mid X \vdash_A X_i\} \cup \bigcup\{X_i \mid i \in F\} \in Con_A$. Hence $\{b_i \mid X \vdash_A X_i\} \cup \{b_i \mid i \in F\} \in Con_B$, by definition of Con. Thus $\{b\} \cup \{b_i \mid i \in F\} \in Con_B$ by the definition of $\vdash$ and Lemma 3.1.2 applied to $\mathbf{B}$. This is what was required.

For Axiom 5 we must show that if $\{\langle X_1, b_1\rangle, \ldots, \langle X_n, b_n\rangle\} \vdash \langle Y_i, c_i\rangle$ for $1 \leq i \leq m$, and $\{\langle Y_1, c_1\rangle, \ldots, \langle Y_m, c_m\rangle\} \vdash \langle Z, d\rangle$ then necessarily $\{\langle X_1, b_1\rangle, \ldots, \langle X_n, b_n\rangle\} \vdash \langle Z, d\rangle$.

So suppose the premises hold. Then, by definition of $\vdash$, $\{c_i \mid Z \vdash_A Y_i\} \vdash_B d$. If $Z \vdash_A Y_i$ then $\{b_j \mid Y_i \vdash_A X_j\} \subseteq \{b_j \mid Z \vdash_A X_j\}$ and hence $\{b_j \mid Z \vdash_A X_j\} \vdash_B c_i$ (since $\{b_j \mid Y_i \vdash_A X_j\} \vdash_B c_i$ by the first premise). Hence

$$\{b_j \mid Z \vdash_A X_j\} \vdash_B \{c_i \mid Z \vdash_A Y_i\} \vdash_B d$$

so that $\{b_j \mid Z \vdash_A X_j\} \vdash_B d$ (by Axiom 5 of $\mathbf{B}$) as required.

This completes the proof that $\rightarrow$ is well-defined.

As before, to prove continuity it will suffice to prove monotonicity and token-set continuity of each argument separately. We will concentrate on the left-hand argument (which has the harder proofs). So suppose $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are any information systems such that $\mathbf{A} \trianglelefteq \mathbf{B}$. Let $\mathbf{D} = \mathbf{A} \rightarrow \mathbf{C}$ and $\mathbf{E} = \mathbf{B} \rightarrow \mathbf{C}$. We must prove $\mathbf{D} \trianglelefteq \mathbf{E}$.

Since trivially $Con_A \subseteq Con_B$, we have $D = Con_A \times C \subseteq Con_B \times C = E$.

If $Z = \{\langle X_1, c_1\rangle, \ldots, \langle X_n, c_n\rangle\} \in Con_D$ then trivially $Z \subseteq D$, and

$$\begin{array}{rcll} \bigcup\{X_i \mid i \in F\} \in Con_B & \Rightarrow & \bigcup\{X_i \mid i \in F\} \in Con_A & \text{as } \mathbf{A} \trianglelefteq \mathbf{B} \\ & \Rightarrow & \{c_i \mid i \in F\} \in Con_C & \text{as } Z \in Con_D \end{array}$$

so $Z \in Con_E$. The converse is very similar.

If $Z \vdash_E \langle X, c\rangle$, where $Z \cup \{\langle X, c\rangle\} \subseteq D$ and $Z = \{\langle X_1, c_1\rangle, \ldots, \langle X_n, c_n\rangle\}$, then we have, for each $1 \leq i \leq n$,

$$X \vdash_B X_i \Leftrightarrow X \vdash_A X_i$$

as $\mathbf{A} \trianglelefteq \mathbf{B}$ and $X \cup X_i \subseteq A$. Thus, as $c \in C$ we have

$$\{c_i \mid X \vdash_B X_i\} \vdash_C c \Leftrightarrow \{c_i \mid X \vdash_A X_i\} \vdash_C c$$

so $Z \vdash_D \langle X, c\rangle$ as required. The above argument also essentially establishes the converse.

This completes the proof that $\rightarrow$ is monotonic in its first argument. (The proof for the second argument is essentially the same but slightly easier.)

That it is continuous on token sets is an easy consequence of the fact that if Δ is a directed set of information systems then the Con set of $\bigsqcup \Delta$ is just the union of those of the members of Δ. The proof for the second argument is the same.

To prove the final part of the proposition we will first show that every approximable mapping is in $|\ \mathbf{A} \rightarrow \mathbf{B}\ |$, and then that the reverse holds.

So suppose $r : \mathbf{A} \rightarrow \mathbf{B}$ is an approximable mapping. r is consistent as a subset of $Con_A \times B$ because, if $\{\langle X_1, b_1\rangle, \ldots, \langle X_n, b_n\rangle\} \subseteq r$ and $X = \bigcup_{i=1}^{n} X_i \in Con_A$ (it is clearly enough to consider this case), $X \vdash_A X_i$ for each i and so Xrb_i by the third axiom of approximable mappings. $\{b_1, \ldots, b_n\} \in Con_B$ then follows by the first axiom.

r is deductively closed as if $Z = \{\langle X_1, b_1\rangle, \ldots, \langle X_n, b_n\rangle\} \subseteq^{\text{fin}} r$ and $Z \vdash \langle X, b\rangle$, then $\bigcup\{b_i \mid X \vdash_A X_i\} \vdash_B b$. Thus Xrb_i whenever $X \vdash_A X_i$ (by the third axiom of approximable mappings) and so Xrb by the second axiom.

If r is an element of $|\ \mathbf{A} \rightarrow \mathbf{B}\ |$ then it satisfies the first axiom of approximable mappings trivially by consistency. If $\{\langle X, b_1\rangle, \ldots, \langle X, b_n\rangle\} \subseteq r$ and $\{b_1, \ldots, b_n\} \vdash_B b$, then $\langle X, b\rangle \in r$ is a trivial consequence of deductive closure. Thus the second axiom holds. The third similarly follows trivially from deductive closure. ∎

Since we know that the approximable mappings from $\mathbf{A}$ to $\mathbf{B}$ are naturally order-isomorphic to the continuous functions from $|\ \mathbf{A}\ |$ to $|\ \mathbf{B}\ |$, (see Theorem 3.3.2) we can thus justifiedly claim that $\rightarrow$ is indeed a domain constructor yielding the domain of continuous functions. We will of necessity study this constructor in depth in later chapters, since it is the central one for producing the domains we require to give semantics to most of the languages treated there.

Note that, as was the case with the other constructors given earlier in this chapter, the *existence* of the domain constructor $\rightarrow$ (as justified by Proposition 4.3.1 and the above remarks), together with the results of Chapter 3, provides a proof that the space of continuous functions from one algebraic, consistently complete cpo to another also has those properties.

There will be further discussion of the function space constructor in Section 5.2, and we will also gain more insight into it when we use it to define semantic domains in Chapter 7 and after.

Exercises

4.3.1 Prove that, for any information systems $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, the information systems $(\mathbf{A} \times \mathbf{B}) \rightarrow \mathbf{C}$ and $\mathbf{A} \rightarrow (\mathbf{B} \rightarrow \mathbf{C})$ are isomorphic (i.e., that $(\mathbf{A} \times \mathbf{B}) \rightarrow \mathbf{C} \equiv \mathbf{A} \rightarrow (\mathbf{B} \rightarrow \mathbf{C})$). What does this tell you about $\mathbf{A} \rightarrow (\mathbf{B} \rightarrow \mathbf{C})$ and $\mathbf{B} \rightarrow (\mathbf{A} \rightarrow \mathbf{C})$?

4.3.2 Use the above constructor and the results of Chapter 3 to identify the pofinite elements of the partial order of continuous functions from one algebraic, consistently complete cpo to another.

4.3.3 Devise a continuous domain constructor $\multimap$ such that $|\ \mathbf{A} \multimap \mathbf{B}\ |$ essentially consists of all *strict* continuous functions from $|\ \mathbf{A}\ |$ to $|\ \mathbf{B}\ |$ (i.e., functions f such that $f(\overline{\emptyset}_A) = \overline{\emptyset}_B$.
Show that the domains of elements of $(\mathbf{A}+\mathbf{B}) \multimap \mathbf{C}$ and $(\mathbf{A} \to \mathbf{C}) \times (\mathbf{A} \to \mathbf{C})$ are always order isomorphic. Under your definition, are the information systems isomorphic too?

4.3.4 Recall our definition of general product spaces on page 88. Given an information system $\mathbf{A}$ and a set X, show that $\mathbf{A}^X$ (which we remarked might be written $X \to \mathbf{A}$) is isomorphic to a subsystem of $\mathbf{flat}(X) \to \mathbf{A}$.
What can you say about $\mathbf{A}^X$ and $\mathbf{flat}(X) \multimap \mathbf{A}$, where $\multimap$ is the strict function space constructor you defined in the previous exercise.

Chapter 5

More about domain equations

In this chapter we illustrate the theory developed in the last chapter by examining some domains and domain equations in detail. In doing this we will both develop techniques for discovering the structures of the solutions to domain equations we might have been given and try to develop the insight needed to decide on the right domain equation to give us domains we may want. From time to time we will find we need to add to our range of domain constructors.

5.1 Lists

A list is an ordered collection of objects from some type. Lists are very common in functional programming languages and have a wide variety of uses there. If we want to construct a semantic domain for one of these languages we will need to know how to create an appropriate domain of lists. An examination of some such languages will however reveal that we need not a single list constructor but many, depending on the way lists are stored and the evaluation strategy of the language. For in some languages lists are always finite while in others they may also be infinite. And the required *strictness* (degree to which the whole list becomes undefined if some element is) varies widely: some languages (in practice ones with only finite lists) identify all lists which have an undefined component; others identify all those that have an undefined *first* element; while others are not strict at all.

To discuss lists accurately it is useful to have a good notation for them. The one given here is that used in various functional programming languages – see [BW] for example. Throughout we will suppose that we are trying to create a domain of lists with elements taken from $|\ \mathbf{A}\ |$. The least element $\overline{\emptyset}$ of $|\ \mathbf{A}\ |$ will be denoted $\perp_A$. Typical elements of $|\ \mathbf{A}\ |$ will be written α, to distinguish them from tokens from A (written a as usual) and elements of the list domains (either written x, y,

z as usual, or l to denote a list).

$[\alpha_0, \alpha_1, \ldots, \alpha_n]$ will denote the finite list containing the elements α_0, $\alpha_1, \ldots$, α_n in that order. Thus $[\,]$ is the empty list, and $[\alpha]$ is the list containing only α. If l is a list (of objects from the domain $|\ \mathbf{A}\ |$, say) and $\alpha \in |\ \mathbf{A}\ |$ then $\alpha : l$ is the list whose first element is α and whose subsequent elements are those of l. Thus $\alpha_0 : [\alpha_1, \alpha_2, \ldots, \alpha_n] = [\alpha_0, \alpha_1, \ldots, \alpha_n]$.

We will usually want to distinguish $[\,]$ from $\perp$, the undefined list. For one can tell that $[\,]$ is empty and use the fact in computation, whereas $\perp$ will typically denote the 'result' of some nonterminating or erroneous calculation. The list $\alpha : \perp$ may or may not be identified with $\perp$. We would want to make the identification in a language where the whole of the list is always computed when any part is needed. But in a *lazy* language, where only those parts of the list actually required are computed, we could either observe that $\alpha : \perp$ is nonempty or that its first element is α without entering the calculation of $\perp$. Similar considerations apply to the decision of whether $\perp_A : l$ should be identified with $\perp$: the answer depends on the precise capabilities of the language under consideration.

In defining the semantics of these languages we will therefore need to be versatile. Our mathematical models should not contain whole classes of object that can never be of use (such as infinite lists in a language without any) and nor should they distinguish between objects that, for a given language, can never be distinguished – for they would then fail to identify programs that ought to be identified. This section is devoted to examining a number of domain equations that yield domains of lists and thereby developing a repertoire of list constructors for different occasions.

The simplest domain equation that looks as though it might give a domain of lists of $\mathbf{A} = \langle A, Con, \vdash \rangle$ is

$$\mathbf{D} = \mathbf{A} \times \mathbf{D}\,.$$

The least solution to this equation is given by $\bigsqcup\{\mathbf{D}_n \mid n \in \mathbb{N}\}$, where $\mathbf{D}_0 = \perp\!\!\!\perp$ and $\mathbf{D}_{n+1} = \mathbf{A} \times \mathbf{D}_n$. (We define D_n to be the token-set of $\mathbf{D}_n$.) Thus a typical element of $|\ \mathbf{D}_n\ |$ can be identified (thanks to Proposition 4.1.2) with an object of the form

$$\langle \alpha_0, \langle \alpha_1, \langle \ldots, \langle \alpha_{n-1}, \emptyset \rangle \ldots \rangle \rangle \rangle$$

where all the α_i are in $|\ \mathbf{A}\ |$. (From now on we will tend to make this identification for the sake of clarity.) The final $\emptyset$ represents the only element of $|\ \perp\!\!\!\perp\ |$. We also know from Proposition 4.1.2 that the order structure of $|\ \mathbf{D}_n\ |$ is the standard symmetrical co-ordinatewise one.

It is convenient to think of the final $\emptyset$ as something that is as yet unknown, and this view is reinforced by an examination of the projection function $\pi_{n,m}$ that

maps $|\,\mathbf{D}_n\,|$ to $|\,\mathbf{D}_m\,|$ when $m \leq n$. For, in $\mathbf{D}_n$, information about the component α_r $(r < n)$ is conveyed by tokens of the form $\langle right, \langle \ldots, \langle right, \langle left, a\rangle\rangle \ldots\rangle\rangle$ $(a \in A)$ where there are exactly r *right*s. Thus, since in general $\pi_{n,m}(x) = x \cap D_m$, the element representing $\langle \alpha_0, \langle \alpha_1, \langle \ldots, \langle \alpha_{n-1}, \emptyset\rangle \ldots\rangle\rangle\rangle$ in $\mathbf{D}_n$ is mapped by $\pi_{n,m}$ to the shorter 'list' $\langle \alpha_0, \langle \alpha_1, \langle \ldots, \langle \alpha_{m-1}, \emptyset\rangle \ldots\rangle\rangle\rangle$. We can think of the last $n - m + 1$ components of the first of these as being 'collapsed' into the final 'unknown' component of the second.

Now that we have identified the approximating domains $\mathbf{D}_n$ and the corresponding projection functions, the structure of the limit domain $\mathbf{D}$ is easy to identify. For it is clear that the cones of elements from the approximating domains are in natural 1-1 correspondence with the set of infinite lists $\{[\alpha_0, \alpha_1, \ldots] \mid \forall\, i.\alpha_i \in |\,\mathbf{A}\,|\}$ (the nth element of the cone being $\langle \alpha_0, \langle \alpha_1, \langle \ldots, \langle \alpha_{n-1}, \emptyset\rangle \ldots\rangle\rangle\rangle$). We have thus constructed the domain of infinite lists of elements of $|\,\mathbf{A}\,|$. We know that two elements of this domain will be ordered (by $\subseteq$) if and only if all the corresponding components of their cones are ordered. It follows that the order on $|\,\mathbf{D}\,|$ corresponds to the natural, componentwise order on the sequences:

$$[\alpha_0, \alpha_1, \ldots] \subseteq [\alpha'_0, \alpha'_1, \ldots] \Leftrightarrow \forall\, i.\alpha_i \subseteq \alpha'_i\,.$$

Note that the least element of this domain is just the infinite sequence consisting of the least element $\perp_A = \overline{\emptyset}$ of $|\,\mathbf{A}\,|$, and that this is different from, say, a sequence whose first element is $\perp_A$ but whose second is not. We have thus succeeded in defining the domain of infinite lists of a given type that is not strict.

Suppose we want a domain that includes finite lists. There are two different sorts of finite lists one might meet. The first is a list which contains a number of elements and can be seen not to have any more. In our notation this is written $[\alpha_0, \alpha_1, \ldots, \alpha_n]$. The second has a number of elements and then produces an error state or loop if one looks at it further and is denoted $\alpha_0 : \alpha_1 : \ldots \alpha_n : \perp$. (Though it only makes sense to call this second sort finite if the infinite list of $\perp_A$s is different from $\perp$.) This sort of list is only finite in a very negative way: there is nothing useful one can do with one that cannot be done with an infinite extension. Certainly the first type of finite list is the one we will most often want. Consider the empty list: there is a piece of information true of it that is true of no other (namely that it is empty), and information about every other non-$\perp$ list not true of it (either information about some element or simply the information that the list is not empty). Therefore we can expect the empty list to be an element of our domain incomparable to all others except $\perp$. Every other non-$\perp$ list can be expected to be of the form $\alpha : l$ for some $\alpha \in |\,\mathbf{A}\,|$ and some list l.

This leads us to expect that the domain of lists will look something like

$$\mathbf{K} = \perp\!\!\!\perp_{\uparrow} \oplus (\mathbf{A} \times \mathbf{K})$$

or like

$$\mathbf{L} = \perp\!\!\!\perp + (\mathbf{A} \times \mathbf{L})\,.$$

Both these equations create a separate point for $[\,]$; the obvious difference between them is that the first identifies $\perp_A : \perp$ with $\perp$ while the second does not. It is instructive to work out what $|\,\mathbf{K}\,|$ and $|\,\mathbf{L}\,|$ look like in more detail.

Let us first consider $\mathbf{K}$, and define $\mathbf{K}_0 = \perp\!\!\!\perp$ and $\mathbf{K}_{n+1} = \perp\!\!\!\perp_{\uparrow} \oplus (\mathbf{A} \times \mathbf{K}_n)$. Thus $|\,\mathbf{K}_0\,|$ has one element (which we can identify with the undefined list $\perp$) and the elements of $|\,\mathbf{K}_{n+1}\,|$ can be identified with $[\,]$ and $\alpha : x$ for $\alpha \in|\, \mathbf{A}\,|$ and $x \in|\, \mathbf{K}_n\,|$. This means that the elements of $|\,\mathbf{K}_n\,|$ have two forms: either they are $[\alpha_0, \alpha_1, \ldots, \alpha_{k-1}]$ for some $0 \leq k < n$ and $\alpha_i \in|\, \mathbf{A}\,|$ *or* they have the form $\alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \perp$. A little analysis reveals that finite lists (of the form $[\alpha_0, \alpha_{n-1}, \ldots, \alpha_{k-1}]$) of different lengths are incomparable, with the usual component-wise order among ones of the same length. 'Unfinished' lists of the form $\alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \perp$ are ordered component-wise among themselves. In general

$$\alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \perp \subseteq [\alpha'_0, \alpha'_1, \ldots, \alpha'_{k-1}]$$

if and only if $\alpha_i \subseteq \alpha'_i$ for $i < k$ and $\alpha_i = \perp_A$ when $i \geq k$. No finite list is below an unfinished one. (All of the order structure that we have just set out can easily be proved by induction from the properties of the domain constructors involved in the equation.)

Let K_n be the token-set of $\mathbf{K}_n$. Noting that, in general, the tokens of a domain $\mathbf{B}$ appear in $\perp\!\!\!\perp_{\uparrow} \oplus (\mathbf{A} \times \mathbf{B})$ with a double *right* tag, we see that K_n contains two types of token. These are $\iota^{2r+1}_{right}(\iota_{left}(a))$ $(r < n)$ where $a \in A$, conveying information about α_r in either type of element, and $\iota^{2r}_{right}(\iota_{left}(lifted))$ $(r < n)$ denoting that the list has finite length r. It is thus apparent that the projection map $\pi_{n+1,n}$ has the following effects, since it 'forgets' about information on α_n and whether a list of length n is finished:

$$\begin{array}{rcll} \pi_{n+1,n}(\alpha_0 : \alpha_1 : \ldots \alpha_n : \perp) & = & \alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \perp & \\ \pi_{n+1,n}[\alpha_0, \alpha_1, \ldots \alpha_{n-1}] & = & \alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \perp & \\ \pi_{n+1,n}[\alpha_0, \alpha_1, \ldots \alpha_{r-1}] & = & [\alpha_0, \alpha_1, \ldots \alpha_{r-1}] & (r < n). \end{array}$$

The cones $\langle x_n \mid n \in \mathbb{N}\rangle$ such that $x_n \in|\, \mathbf{K}_n\,|$ and $\pi_{n+1,n}(x_{n+1}) = x_n$ have two forms: *either* all x_n are 'unfinished', in which case there is an infinite sequence $\langle \alpha_i \mid i \geq 0\rangle$ of elements of $|\,\mathbf{A}\,|$ such that for all n $x_n = \alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \perp$, *or* some first

x_n is 'finite'. In the latter case notice that $\pi_{n,n-1}(x_n)$ can only be unfinished (as is certain since x_n is the first finite one) if $x_n = [\alpha_0, \ldots, \alpha_{n-2}]$ for some α_i. Since there is only one element of $|\ \mathbf{K}_m\ |$ mapping under $\pi_{m+1,m}$ to $[\alpha_0, \ldots, \alpha_{n-1}]$ when $m > n$ it follows that $x_r = \alpha_0 : \alpha_1 : \ldots : \alpha_{r-1} : \perp$ when $r < n$ and $x_r = [\alpha_0, \ldots, \alpha_{n-2}]$ when $r \geq n$.

We will identify the first sort of cone with the infinite sequence $[\alpha_i \mid i \geq 0]$. The second sort of cone will be identified with the finite sequence at which the cone stabilises. Note that there is exactly one element of $|\ \mathbf{K}\ |$ corresponding to each finite or infinite sequence of $|\ \mathbf{A}\ |$. The order on $|\ \mathbf{K}\ |$ is easily deduced (via the cones) from those on the $|\ \mathbf{K}_n\ |$:

- Finite lists of different lengths are incomparable.
- $[\alpha_0, \ldots, \alpha_{n-1}] \subseteq [\alpha'_0, \ldots, \alpha'_{n-1}]$ iff $\alpha_i \subseteq \alpha'_i$ for all i.
- $[\alpha_i \mid i \geq 0] \subseteq [\alpha'_i \mid i \geq 0]$ iff $\alpha_i \subseteq \alpha'_i$ for all i.
- $[\alpha_i \mid i \geq 0] \subseteq [\alpha'_0, \ldots, \alpha'_{n-1}]$ iff $\alpha_i \subseteq \alpha'_i$ for all $i < n$ and $\alpha_i = \perp_A$ for all $i \geq n$.
- $[\alpha_0, \ldots, \alpha_{n-1}] \not\subseteq [\alpha'_i \mid i \geq 0]$.

In this domain, as in the domain of infinite sequences above, we are identifying the totally undefined list with an infinite list of $\perp_A$s. What this identification means is that, while the fact that a list is finished is positive information that is recorded among the tokens, the mere fact that it is not finished is not. The only way we can tell that a list is not empty is to see information about one of its members (not necessarily the first). This rather curious position does not accurately model the domain of lists as modelled in real programming languages. It is much more likely either that it should be possible to see that a list is nonempty without seeing positive information about a member, or that a list whose first element is $\perp_A$ should be identified with $\perp$.

Let us now examine the domain $\mathbf{L}$. Here we do have a positive piece of information that a list is not empty (the token that lifts it into the right-hand side of the separated sum $\perp\!\!\!\perp + (\mathbf{A} \times \mathbf{L})$) that tells us nothing about any of the elements of the list. Let $\mathbf{L}_n$ be the iterates of $\mathbf{L}$ defined in the same way as $\mathbf{K}_n$. Again the only element of $|\ \mathbf{L}_0\ |$ can be denoted $\perp$, and this time the elements of $|\ \mathbf{L}_{n+1}\ |$ are $\perp$ (the bottom added by the separated sum), $[\,]$ (the single element in the left-hand side of the separated sum) and $a : x$ for all elements $a \in |\ \mathbf{A}\ |$ and $x \in |\ \mathbf{L}_n\ |$, now all different from $\perp$. Thus the elements of $|\ \mathbf{L}_n\ |$ are precisely $\alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \perp$ (with $0 \leq r \leq n$) and $[\alpha_0, \alpha_1, \ldots, \alpha_{r-1}]$ (with $0 \leq r < n$). We will re-christen lists of the first sort 'partial' rather than 'unfinished' for reasons that will become apparent, though it is arguably better to keep the old name for those where $r = n$. The second sort will again be called 'finite'. The partial order on $|\ \mathbf{L}_n\ |$ is as follows:

- Finite lists of different lengths are incomparable.
- $[\alpha_0, \ldots, \alpha_{r-1}] \subseteq [\alpha'_0, \ldots, \alpha'_{r-1}]$ iff $\alpha_i \subseteq \alpha'_i$ for all i.
- $\alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot \subseteq \alpha'_0 : \alpha'_1 : \ldots \alpha'_{s-1} : \bot$ iff $r \leq s$ and $\alpha_i \subseteq \alpha'_i$ when $i < r$.
- $\alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot \subseteq [\alpha'_0, \alpha'_1, \ldots, \alpha'_{s-1}]$ iff $r \leq s$ and $\alpha_i \subseteq \alpha'_i$ when $i < r$.
- No finite list is below a partial one.

The projection functions are very similar to those for the $\mathbf{L}_n$ and are as follows:

$$\begin{array}{rcll}
\pi_{n+1,n}(\alpha_0 : \alpha_1 : \ldots \alpha_n : \bot) & = & \alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \bot & \\
\pi_{n+1,n}(\alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot) & = & \alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot & (r \leq n) \\
\pi_{n+1,n}[\alpha_0, \alpha_1, \ldots \alpha_{n-1}] & = & \alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \bot & \\
\pi_{n+1,n}[\alpha_0, \alpha_1, \ldots \alpha_{r-1}] & = & [\alpha_0, \alpha_1, \ldots \alpha_{r-1}] & (r < n).
\end{array}$$

Note that there are now two varieties of element in $|\ \mathbf{L}_n\ |$ with only one thing from $|\ \mathbf{L}_{n+1}\ |$ projecting onto them: the finite lists and the partial ones with length less than n. There are now three sorts of cone. The first is where each x_n is a partial list of length n; necessarily there is an infinite sequence $\langle \alpha_i \mid i \geq 0 \rangle$ such that, for all n, $x_n = \alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \bot$. We will identify this cone with the infinite list $[\alpha_i \mid i \geq 0]$.

If any x_n has either of the two other possible forms, finite or shorter partial, then all subsequent x_m must be identical to x_n. In the first case there must be α_0, $\alpha_1, \ldots$, α_{n-2} such that $r \geq n \Rightarrow x_r = [\alpha_1, \alpha_2, \ldots, \alpha_{n-2}]$, and $r < n \Rightarrow x_r = \alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot$. We identify this cone with the finite list $[\alpha_0, \alpha_1, \ldots, \alpha_{n-2}]$.

In the second case there must be α_0, $\alpha_1, \ldots$, α_{n-1} such that $r \geq n \Rightarrow x_r = \alpha_0 : \alpha_1 : \ldots : \alpha_{n-2} : \bot$ and $r < n \Rightarrow x_r = \alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot$. We identify this cone with the partial list $\alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \bot$. These three types are all the elements of $|\ \mathbf{L}\ |$. The partial order is easily deduced from those of $|\ \mathbf{L}_n\ |$.

- Finite lists of different lengths are incomparable.
- $[\alpha_0, \ldots, \alpha_{r-1}] \subseteq [\alpha'_0, \ldots, \alpha'_{r-1}]$ iff $\alpha_i \subseteq \alpha'_i$ for all i.
- $[\alpha_i \mid i \geq 0] \subseteq [\alpha'_i \mid i \geq 0]$ iff $\alpha_i \subseteq \alpha'_i$ for all i.
- Finite lists are incomparable with infinite ones.
- $\alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot \subseteq \alpha'_0 : \alpha'_1 : \ldots \alpha'_{s-1} : \bot$ iff $r \leq s$ and $\alpha_i \subseteq \alpha'_i$ when $i < r$.
- $\alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot \subseteq [\alpha'_0, \alpha'_1, \ldots, \alpha'_{s-1}]$ iff $r \leq s$ and $\alpha_i \subseteq \alpha'_i$ when $i < r$.
- $\alpha_0 : \alpha_1 : \ldots \alpha_{r-1} : \bot \subseteq [\alpha'_i \mid i \geq 0]$ iff $\alpha_i \subseteq \alpha'_i$ when $i < r$.
- No finite or infinite list is below a partial one.

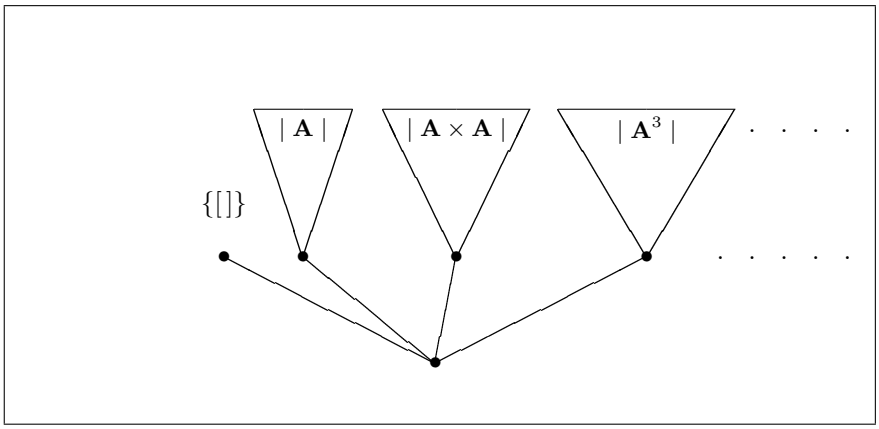

Figure 5.1 | **F** | – the domain of finite lists

| **L** | thus contains all three varieties of list identified at the start of the chapter, with no identifications or additional elements. It is precisely the domain of lists required for most lazy functional languages, including that of [BW].

All the domains of lists we have constructed so far contain all infinite lists. Not all programming languages encompass these, and as we remarked before it is undesirable to create domains which contain objects irrelevant to a language.

Now, instead of writing down some plausible domain equations and calculating the result, we will try to derive an equation that yields a domain consisting of only finite lists (with ones of different lengths being incomparable) plus a single bottom element $\perp$. Figure 5.1 shows the domain we are aiming for; let us call it **F**. The obvious domains approximating **F** are ones $\mathbf{F}_n$ whose elements consist of finite lists of length $n-1$ or less. (We get $n-1$ rather than n here because there is a list with no elements $[\,]$, and | $\mathbf{F}_0$ | will contain no complete objects.) Experience suggests that, under the projection from | $\mathbf{F}_{n+1}$ | to | $\mathbf{F}_n$ |, a list of length less than n should map to itself. There are two plausible places that a list $[\alpha_0, \alpha_1, \ldots, \alpha_{n-1}]$ of length n could map to: either to $[\alpha_0, \alpha_1, \ldots, \alpha_{n-2}]$ or to $\perp$. If we adopted the first of these options there would be, as in **K** and **L**, cones of elements corresponding to each infinite sequence. But we do not want infinite lists in | **F** |, so the second option must apply. Indeed if we do adopt this projection map it is easy to see that the only possible cones $\langle x_i \mid i \geq 0 \rangle$ are the one with all $x_i = \perp$, and, for each finite sequence $[\alpha_0, \alpha_1, \ldots, \alpha_{n-1}]$, the cone with $x_i = \perp$ for $i \leq n$ and $x_i = [\alpha_0, \alpha_1, \ldots, \alpha_{n-1}]$ for $i > n$. This leads to exactly the domain we required.

Unfortunately there is no way of producing $\mathbf{F}_{n+1}$ from $\mathbf{F}_n$ via any of the domain constructors we have seen so far. This is because, when we take the product $\mathbf{A} \times \mathbf{F}_n$, we get an element $\langle \alpha, \perp \rangle$ for every $\alpha \in$| **A** |. This is not part of $\mathbf{F}_{n+1}$ since

the latter only has one element that is not a 'complete' sequence. It turns out that we need to identify all of these elements: we need a *right strict* product constructor $\times\!\!\!\!\searrow$ with the property that the elements of $|\ \mathbf{A} \times\!\!\!\!\searrow \mathbf{B}\ |$ correspond to the pairs $\langle x, y\rangle$ such that $y \neq \bot_B$ plus a single bottom element. If we had such a constructor then $\mathbf{A} \times\!\!\!\!\searrow \mathbf{F}_n$ would consist of all lists of length 1 to n plus a single bottom element. Thus $\bot_\uparrow \oplus (\mathbf{A} \times\!\!\!\!\searrow \mathbf{F}_n)$ would have the correct shape for $\mathbf{F}_{n+1}$. This suggests that the domain equation generating the finite lists is

$$\mathbf{F} = \bot_\uparrow \oplus (\mathbf{A} \times\!\!\!\!\searrow \mathbf{F})\,.$$

However we cannot tell if the projection maps are correct until we know more about the constructor.

Strict product constructors

In fact there are three different strict versions of Cartesian product. $\times\!\!\!\!\searrow$, the one discussed above, is strict in its right-hand component so that all pairs of the form $\langle x, \bot\rangle$ are identified. $\swarrow\!\!\!\!\times$ is the symmetric version that is strict in its left-hand argument. $\swarrow\!\!\!\!\times\!\!\!\!\searrow$ is strict in both: namely all elements with the form $\langle x, \bot\rangle$ or $\langle \bot, y\rangle$ are identified.

We will start by defining the symmetric version $\swarrow\!\!\!\!\times\!\!\!\!\searrow$. When we first considered the ordinary product constructor $\mathbf{A} \times \mathbf{B}$, tokens of the form $\langle a, b\rangle$ (a and b being respectively tokens from $\mathbf{A}$ and $\mathbf{B}$) were rejected. For one could not then give information about the first component of a pair without giving information about the second, and there might be no information about the second component (when this equalled $\emptyset$). But of course this is exactly what is required for a strict constructor: there can be no information about one component without knowing that the other is not $\bot$. There is, as in the case of $\oplus$, the complication that there may or may not be tokens true of the least element $\overline{\emptyset}$, since to make this idea work we must ensure that the least elements of $|\ \mathbf{A}\ |$ and $|\ \mathbf{B}\ |$ have no tokens true of them. Therefore we make the following definition.

Definition
$\mathbf{A} \swarrow\!\!\!\!\times\!\!\!\!\searrow \mathbf{B} = \langle (A - \overline{\emptyset}^A) \times (B - \overline{\emptyset}^B), Con, \vdash\rangle$ where

- $X \in Con \Leftrightarrow \pi_1(X) \in Con_A \wedge \pi_2(X) \in Con_B$
- $X \vdash \langle a, b\rangle \Leftrightarrow \pi_1(X) \vdash_A a \wedge \pi_2(X) \vdash_B b.$

Here, $\pi_1(X) = \{a \mid \langle a, b\rangle \in X\}$ and $\pi_2(X) = \{b \mid \langle a, b\rangle \in X\}$ are the functions that extract information about the first and second components of elements of X. ∎

PROPOSITION 5.1.1
$\times\!\!\!\!\!\!\diagup$ is well-defined and continuous. Furthermore, if **A** and **B** are any two information systems then $|\ \mathbf{A} \times\!\!\!\!\!\!\diagup \mathbf{B}\ |$ is given by

$$\{\emptyset\} \cup \{(x - \overline{\emptyset}^A) \times (y - \overline{\emptyset}^B) \mid x \in (|\ \mathbf{A}\ | - \{\overline{\emptyset}^A\}) \wedge y \in (|\ \mathbf{B}\ | - \{\overline{\emptyset}^B\})\}\ .$$

Thus $|\ \mathbf{A} \times\!\!\!\!\!\!\diagup \mathbf{B}\ |$ is order isomorphic to the symmetric strict product of $|\ \mathbf{A}\ |$ and $|\ \mathbf{B}\ |$. ∎

The order isomorphism here is easily deduced using the fact that, if x, x', y, y' are all nonempty sets, then $x \times y = x' \times y'$ if and only if $x = x'$ and $y = y'$.

The situation with the left- and right-strict operators is similar. For example in $|\ \mathbf{A} \times\!\!\!\!\diagup \mathbf{B}\ |$ there can be no information about the left-hand component without enough information to ensure that the right-hand component is not bottom. However there should be information about the right-hand component accessible without knowing anything about the left-hand one. Perhaps the cleanest way of achieving this is to make sure that there are meaningless tokens in the left-hand system. Unfortunately we cannot guarantee they will be there in a given **A**, just as we could not guarantee before that they would not be! Therefore we 'add' a token to **A**, just as we did when lifting; however it seems appropriate to give the new token a different name than 'lifted': we will call it Δ[1].

Definition
$\mathbf{A} \times\!\!\!\!\diagup \mathbf{B} = \langle C, Con, \vdash \rangle$, where

- $C = (\{\Delta\} \cup \{\langle old, a\rangle \mid a \in A\}) \times (B - \overline{\emptyset}^B)$
- $X \in Con \ \Leftrightarrow\ \pi_1'(X) \in Con_A \wedge \pi_2(X) \in Con_B$
- $X \vdash \langle \Delta, b\rangle \ \Leftrightarrow\ \pi_2(X) \vdash_B b$
  $X \vdash \langle\langle old, a\rangle, b\rangle \ \Leftrightarrow\ \pi_1'(X) \vdash_A a \wedge \pi_2(X) \vdash_B b$

  π_2 is as before and $\pi_1'(X) = \{a \mid \langle\langle old, a\rangle, b\rangle \in X\}$. ∎

Note that Δ conveys no information about the first component of a pair; its only function is to allow pairs of the form $\langle \Delta, b\rangle$ to give information about the second component. (We do not want our constructor to be strict in its first argument; therefore it must be possible to give information about the second even when there is nothing to tell about the first.) The following result shows that this definition works.

[1] We have chosen this name because, in Scott's original presentation of information systems where *all* systems had a weakest token that was true of everything, that token was called Δ.

PROPOSITION 5.1.2.
$\leftthreetimes$ is well-defined and continuous. If $\mathbf{A}$ and $\mathbf{B}$ are any information systems then

$$| \mathbf{A} \leftthreetimes \mathbf{B} | = \{\emptyset\} \cup \{\iota_\Delta(x) \times (y - \overline{\emptyset}^B) \mid x \in | \mathbf{A} | \wedge y \in (| \mathbf{B} | - \{\overline{\emptyset}^B\})\}$$

where $\iota_\Delta(x) = \{\Delta\} \cup \{\langle old, a\rangle \mid a \in x\}$. Thus $| \mathbf{A} \leftthreetimes \mathbf{B} |$ is order isomorphic to the right-strict product of $| \mathbf{A} |$ and $| \mathbf{B} |$. ∎

The order isomorphism follows from the same fact used for $\times$. It is important that $\iota_\Delta(x)$ is never empty.

Plainly there is no need for us to define the *left*-strict constructor $\rightthreetimes$ here, since it is just the symmetric version of $\leftthreetimes$.

Let us now return to the problem that generated this discussion, namely the domain $\mathbf{F}$ of finite lists. We suspected that the equation

$$\mathbf{F} = \perp_\uparrow \oplus (\mathbf{A} \leftthreetimes \mathbf{F})$$

was what was required. We already know from the discussion above that the shape of $| \mathbf{F}_n |$ is a bottom element plus disjoint and incomparable copies of the domains of sequences of all lengths $k < n$. Thus the elements of this domain are either $\perp$ or $[\alpha_1, \ldots, \alpha_k]$ for some $k < n$ (and all such are distinct). To work out the projection $\pi_{n+1,n}$ we need to know more about the token-set F_n of $\mathbf{F}_n$. $F_0 = \emptyset$, and for all n,

$$\begin{aligned} F_{n+1} \quad = \quad & \{\langle left, lifted\rangle\} \cup \{\langle right, \langle \Delta, c\rangle\rangle \mid c \in F_n\} \cup \\ & \{\langle right, \langle \langle old, a\rangle, c\rangle\rangle \mid a \in A \wedge c \in F_n\} \end{aligned}$$

Note that we have not needed to remove meaningless tokens here, either in forming the strict product or the coalesced sum. This is because the way the $\mathbf{F}_n$ are built up means there are never any in either circumstance; for note that no coalesced sum or (semi-) strict product *ever* has a meaningless token. The set-theoretic clutter makes it a little hard to see what is going on here; it is much better if we re-write the above equation as a BNF-style definition (where the disjointness is assumed rather than elaborately enforced). Thus, if c_n represents a typical element of F_n, we have

$$c_{n+1} ::= [\,] \mid \Delta : c_n \mid a : c_n$$

(where the order of the clauses is exactly as in the earlier equation). We have chosen the list notation for tokens because, as can be seen from the definitions of $\leftthreetimes$ and $\oplus$, in $\mathbf{F}_n$ information about lists of the form $[\alpha_1, \ldots, \alpha_r]$ is provided by tokens of the form $[b_1, \ldots, b_r]$ of the same length, with tokens of different lengths being inconsistent. (Here, b_i is either in $A \setminus \overline{\emptyset}^A$ or is Δ.)

Recall that the projection maps are defined $\pi_{n+1,n}(x) = x \cap F_n$. But the only elements in $F_{n+1} - F_n$ are the tokens about lists of length n. Thus, if $x \in |\ \mathbf{F}_{n+1}\ |$ represents a shorter list we have $\pi_{n+1,n}(x) = x$ (for x contains no token not in F_n) and if x represents a list of length n we have $\pi_{n+1,n}(x) = \emptyset$ (for x contains no token in F_n). Thus the projections are exactly as we needed, so the limit domain has exactly the form we wanted.

We could have told this without looking at the projection maps, since the tokens of $\mathbf{F}$ are just the union of the token sets F_n. This is just the set defined, in BNF style, by

$$c\ ::=\ [\,] \mid \Delta : c \mid a : c\,,$$

namely all finite lists of Δs and as. And the consistency and entailment relations are just the limits of those of the $\mathbf{F}_n$, so tokens of different lengths are inconsistent, with the ones of a given length having just the right structure to tell us about lists of elements of $|\ \mathbf{A}\ |$ of that length. Note that $[\Delta, \ldots, \Delta]$ (n Δs) is the weakest token among those of length n.

A variant on this domain that one might well want is the one which is strict in all $\mathbf{A}$ components: namely any finite list with a bottom in any component becomes identified with the bottom of the whole domain. This is the domain of finite lists found in several eager functional languages. The way to construct it is shown in Exercise 5.1.3.

Domain constructors defined by fixed points

In the discussion of lists above we always started with a domain $\mathbf{A}$ of list 'atoms' and ended up with some domain of lists of elements of $|\ \mathbf{A}\ |$. All these list-domain forming operations are of course just domain constructors. For example $\mathbf{L}$ is a constructor (which we might better write $List(\mathbf{A})$ or similar) which, given any domain $\mathbf{A}$, returns the appropriate domain of lists. It would be useful to know if these constructors had the same properties of monotonicity and continuity as the directly defined ones earlier. For if this is the case we will be able to use them in domain equations such as

$$\mathbf{D}\ =\ \mathbf{B} \oplus (\mathbf{D} \to \mathbf{D})_{\uparrow} \oplus (List(\mathbf{D}))_{\uparrow}$$

which might be required for a functional language with lists.

In fact they do all have these desirable properties, as a consequence of the following general result.

5.1.3 THEOREM
Suppose F is a continuous binary domain constructor (i.e., 2 to 1). Then the unary domain constructor G formed by defining $G(\mathbf{A})$ to be the least solution to $\mathbf{B} = F(\mathbf{A}, \mathbf{B})$ is itself continuous.

PROOF
We know from Theorem 4.2.8 that $G(\mathbf{A}) = \bigsqcup\{F^n_{\mathrm{A}}(\bot) \mid n \in \mathbb{N}\}$ where $F_{\mathrm{A}}(\mathbf{B}) = F(\mathbf{A}, \mathbf{B})$. Suppose $\mathbf{A} \trianglelefteq \mathbf{A}'$. Now $F^0_{\mathrm{A}}(\bot) \trianglelefteq F^0_{\mathrm{A}'}(\bot)$ trivially, and if $F^n_{\mathrm{A}}(\bot) \trianglelefteq F^n_{\mathrm{A}'}(\bot)$ then $F^{n+1}_{\mathrm{A}}(\bot) \trianglelefteq F^{n+1}_{\mathrm{A}'}(\bot)$ by monotonicity of F. Hence $F^n_{\mathrm{A}}(\bot) \trianglelefteq F^n_{\mathrm{A}'}(\bot)$ for all n by induction. Thus

$$\bigsqcup\{F^n_{\mathrm{A}}(\bot) \mid n \in \mathbb{N}\} \trianglelefteq \bigsqcup\{F^n_{\mathrm{A}'}(\bot) \mid n \in \mathbb{N}\}$$

and so G is monotone.

Next, suppose Δ is a directed set of information systems. Since G is monotone we know that $\bigsqcup\{G(\mathbf{A}) \mid \mathbf{A} \in \Delta\} \trianglelefteq G(\bigsqcup\Delta)$ and hence that the token-set of the right-hand side contains that of the left-hand side. So suppose a is a token of $G(\bigsqcup\Delta) = \bigsqcup\{F^n_{\sqcup\Delta}(\bot) \mid n \in \mathbb{N}\}$. Necessarily there is some n such that a is a token of $F^n_{\sqcup\Delta}(\bot)$.

Claim that $F^n_{\sqcup\Delta}(\bot) = \bigsqcup\{F^n_{\mathrm{A}}(\bot) \mid \mathbf{A} \in \Delta\}$ (i.e., $F^n_{\mathrm{A}}(\bot)$ is continuous as a function of $\mathbf{A}$). We will prove this by induction on n. It is trivial if $n = 0$, and if it is true for n then

$$\begin{array}{rcll} F^{n+1}_{\sqcup\Delta}(\bot) & = & F(\bigsqcup\Delta, F^n_{\sqcup\Delta}(\bot)) & \\ & = & \bigsqcup\{F(\mathbf{A}, F^n_{\mathrm{B}}(\bot)) \mid \mathbf{A}, \mathbf{B} \in \Delta\} & \text{as } F \text{ is continuous} \\ & & & \text{and by induction} \\ & = & \bigsqcup\{F(\mathbf{A}, F^n_{\mathrm{A}}(\bot)) \mid \mathbf{A} \in \Delta\} & \Delta \text{ directed} \\ & = & \bigsqcup\{F^{n+1}_{\mathrm{A}}(\bot) \mid \mathbf{A} \in \Delta\}\,, & \end{array}$$

so it is true for $n+1$ and hence all n by induction.

Thus, if n is as above, there is some $\mathbf{A} \in \Delta$ such that a is a token of $F^n_{\mathrm{A}}(\bot)$ and hence of $G(\mathbf{A})$. This proves that G is continuous on token sets and hence continuous. ∎

This result shows that all of the list constructors seen earlier are indeed continuous. Of course there is a more general version of this result which applies to more than just 2 to 1 constructors. We state it here without proof.

5.1.4 THEOREM
Suppose $F(\underline{\mathbf{A}}, \underline{\mathbf{B}})$ is a continuous $n+m$ to m domain constructor. (Here, $\underline{\mathbf{B}}$ denotes an m-tuple and $\underline{\mathbf{A}}$ an n-tuple.) Then the n to m domain constructor G formed by defining $G(\underline{\mathbf{A}})$ to be the least solution to $\underline{\mathbf{B}} = F(\underline{\mathbf{A}}, \underline{\mathbf{B}})$ is itself continuous. ∎

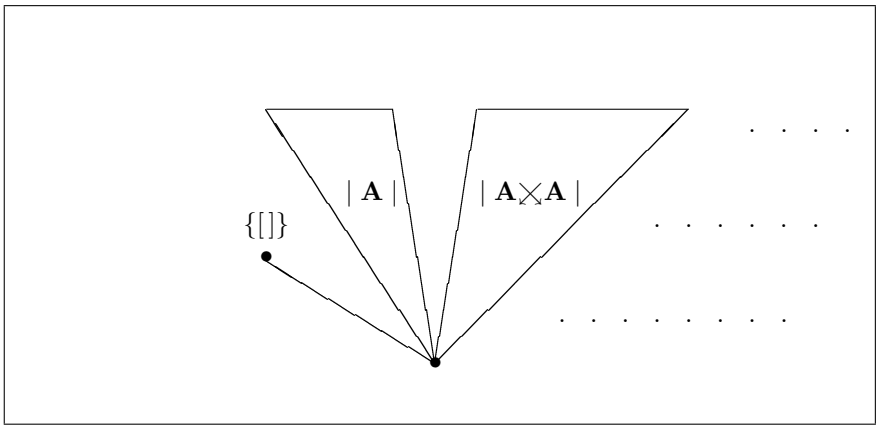

Figure 5.2 | **S** | – the domain of strict finite lists

Exercises

5.1.1 Compute the structures of the least solutions to the domain equations

$$\mathbf{D} = \mathbf{A} \otimes \mathbf{D} \qquad \text{and} \qquad \mathbf{E} = \mathbf{A} \otimes \mathbf{E}$$

for an arbitrary information system **A**.

5.1.2 Can you find a domain equation whose solution looks like | **K** | and | **L** | defined in the text above except that it is strict in the first element of a list. The elements should therefore be

- finite lists $[\alpha_0, \ldots, \alpha_{n-1}]$ $(n \geq 0)$ with all $\alpha_i \neq \bot_A$,
- infinite lists $[\alpha_i \mid i \geq 0]$ such that all $\alpha_i \neq \bot_A$, and
- 'partial' lists $\alpha_0 : \alpha_1 : \ldots \alpha_{n-1} : \bot$ $(n \geq 0)$ with all $\alpha_i \neq \bot_A$.

The order between these elements should be exactly as in **L** (and exactly as in **K** if one substitutes $\bot$ by an infinite list of $\bot_A$s).

5.1.3 Show that the least solution of the domain equation

$$\mathbf{S} = \bot_{\uparrow} \oplus (\mathbf{A} \otimes \mathbf{S})$$

is the *strict* domain of finite sequences referred to in the text above where sequences of different lengths are incomparable except that any sequence containing a $\bot_A$ is identified with the overall $\bot$. This domain is illustrated in Figure 5.2.

5.1.4 The following are all domain equations attempting to describe binary trees of elements of $|\ \mathbf{A}\ |$. In each case establish the shape of the domain generated.

(a) $\mathbf{D} = \mathbf{A} \times (\mathbf{D} \times \mathbf{D})$
(b) $\mathbf{T} = \ \perp + (\mathbf{A} \times (\mathbf{T} \times \mathbf{T}))$
(c) $\mathbf{N} = \mathbf{A} + (\mathbf{N} \times \mathbf{N})$
(d) $\mathbf{F} = \ \perp_{\uparrow} \oplus (\mathbf{A} \swarrow\!\!\!\times\!\!\!\searrow (\mathbf{F} \swarrow\!\!\!\times\!\!\!\searrow \mathbf{F}))$
(e) $\mathbf{G} = \mathbf{A}_{\uparrow} \oplus (\mathbf{G} \swarrow\!\!\!\times\!\!\!\searrow \mathbf{G})$

5.1.5 Can you find a way of generating a domain of finite-branching trees with nodes labelled by elements of $|\ \mathbf{A}\ |$? Each node should contain a label from $|\ \mathbf{A}\ |$ and an arbitrary length finite list of subtrees.

5.1.6 Suppose F and G are two continuous binary domain constructors. We can define domains $\mathbf{A}$ and $\mathbf{B}$ by the mutual definition

$$\begin{aligned} \mathbf{A} &= F(\mathbf{A}, \mathbf{B}) \\ \mathbf{B} &= G(\mathbf{A}, \mathbf{B})\,. \end{aligned}$$

Alternatively we can use Theorem 5.1.3 to define a continuous domain constructor $G_1(\mathbf{A})$ whose value is the least solution (for fixed $\mathbf{A}$) of the equation $\mathbf{B} = G(\mathbf{A}, \mathbf{B})$. We can then define $\mathbf{A}'$ to be the least solution to the equation

$$\mathbf{A}' = F(\mathbf{A}', G_1(\mathbf{A}'))\,.$$

Prove that $\mathbf{A} = \mathbf{A}'$.

5.1.7 There is seemingly a lot of redundancy in the tokens of the strict product constructors. For example in $\langle x, y\rangle \in |\ \mathbf{A} \swarrow\!\!\!\times\!\!\!\searrow \mathbf{B}\ |$ all tokens of the form $\langle a, b\rangle$, with b a piece of information about y, tell us exactly the same thing (namely a) about x. Why do you think this is? Can it be avoided?
[Hint: You might find it helpful to consider the following question: given a token a of A, need there be a least element of $|\ \mathbf{A} \swarrow\!\!\!\times\!\!\!\searrow \mathbf{B}\ |$ which has a true of its first component?]

*5.1.8 Suppose that the information systems $\mathbf{A}_n$ are such that $\mathbf{A}_n \trianglelefteq \mathbf{A}_{n+1}$ and $\mathbf{A}_n \equiv \mathbf{A}_m$ for all n, m. If each of the token sets A_n is finite, prove that the limit $\mathbf{A}^* = \bigsqcup\{\mathbf{A}_n \mid n \in \mathbb{N}\}$ satisfies $\mathbf{A}_n \equiv \mathbf{A}^*$. Need this still hold if the A_n are infinite?

5.2 Function space constructors

The equation that to some extent inspired the whole subject of domain theory was $\mathbf{D} = \mathbf{D} \to \mathbf{D}$, since solutions to this equation were required as models for

the pure λ-calculus (see Chapter 7). Strangely enough, this equation turns out to behave rather differently from all others we will meet. For in fact it has a very simple solution: it is easy to verify that $\bot = \bot \to \bot$. This simply says that there is only one function from a singleton set to itself; so the set can be regarded as isomorphic to its own function space. Plainly $\bot$ is the least solution to the equation, but equally it is not the solution we are looking for: a one-point domain is not an interesting semantic domain.

This is the only case we will meet where the *least* solution to a domain equation is not the *right* one. The reasons why we run into this problem will be discussed later when we meet the λ-calculus. In some sense the solution $\bot$ is the only one that is *not* right since, as is discussed in exercises at the end of Section 7.2, any nontrivial system $\mathbf{D}$ such that $\mathbf{D} \equiv (\mathbf{D} \to \mathbf{D})$ gives a useful model for the pure λ-calculus. Notice that we have replaced our usual equality here by isomorphism. That this is necessary is demonstrated by the following result, which at first sight seems to kill off all hope of finding the domains we are after.

5.2.1 PROPOSITION
$\bot$ is the only solution of the domain equation $\mathbf{D} = \mathbf{D} \to \mathbf{D}$.

PROOF
The set-theoretic axiom of foundation[2] tells us that there does not exist an infinite sequence a_i of sets with $a_0 \ni a_1 \ni a_2 \ni \cdots a_n \ni a_{n+1} \ni \cdots$. Any solution $\langle D, Con, \vdash \rangle$ to the domain equation with nonempty token set D would violate this. For assume a_0 is any element of D. Then, since $D = Con \times D$ by assumption, there are X_1 and a_1 such that $a_0 = \langle X_1, a_1 \rangle$. Similarly there are X_2 and a_2 such that $a_1 = \langle X_2, a_2 \rangle$ and in general X_{n+1} and a_{n+1} such that $a_n = \langle X_{n+1}, a_{n+1} \rangle$.

In set theory, the pair $\langle a, b \rangle$ is identified with the set $\{\{a\}, \{a, b\}\}$. We therefore have the following:

$$a_0 \ni \{X_1, a_1\} \ni a_1 \ni \{X_2, a_2\} \ldots \ni a_n \ni \{X_{n+1}, a_{n+1}\} \ni a_{n+1} \ni \ldots$$

in contravention of the axiom of foundation. Thus $\langle D, Con, \vdash \rangle$ cannot exist. ∎

However there is a way around this. The solution is to find another function space constructor $\mapsto$ such that $\mathbf{A} \mapsto \mathbf{B} \equiv \mathbf{A} \to \mathbf{B}$ but which does not have this problem. Before we try to find this constructor it will be nice to know what we want of it.

---

[2]The axiom of foundation (or regularity) states that any nonempty set x contains an element y such that $x \cap y = \emptyset$. This implies that there can be no sequence $\langle a_i \mid i \in \mathbb{N} \rangle$ such that $a_{i+1} \in a_i$ for all i. For the set $x = \{a_i \mid i \in \mathbb{N}\}$ would violate the axiom: $a_{i+1} \in x \cap a_i$ for all i.

It has been argued by Aczel and others that it would be more natural to base domain theory on a non-standard set theory *without* the axiom of foundation.

Looking back at the proof of Theorem 4.2.8, where we proved domain equations had solutions, the only property of $\perp$ that we used was that $\perp \trianglelefteq F(\perp)$ – this got the induction going to create the chain $\langle F^n(\perp) \mid n \in \mathbb{N}\rangle$. If $\mathbf{A}$ is *any* system such that $\mathbf{A} \trianglelefteq F(\mathbf{A})$ then exactly the same argument produces a chain $\langle F^n(\mathbf{A}) \mid n \in \mathbb{N}\rangle$ whose limit $\bigsqcup\{F^n(\mathbf{A}) \mid n \in \mathbb{N}\}$ is a fixed point.

Thus what we would like is a nontrivial $\mathbf{A}$ and a constructor $\mapsto$ as above such that $\mathbf{A} \trianglelefteq (\mathbf{A} \mapsto \mathbf{A})$. To achieve this we obviously need $A \subseteq A'$, where A' is the token set of $\mathbf{A} \mapsto \mathbf{A}$. In $\mathbf{B} \to \mathbf{C}$, the tokens are $Con_B \times C$ – we are looking for a way of replacing some of these by either the elements of B or of C. (The fact that $\mathbf{A}$ appears on both sides of $\mathbf{A} \mapsto \mathbf{A}$ gives us that choice.) If we consider the case where $\mathbf{C} = \perp$, the token set $Con_B \times C$ is empty however large B might be, which indicates that it might be difficult to embed the tokens of $\mathbf{B}$ into those of $\mathbf{B} \to \mathbf{C}$.

The situation is much better when we try to embed tokens of $\mathbf{C}$. For whatever the shape of $\mathbf{B}$, the set Con_B always contains the empty set $\emptyset$ – so $Con_B \times C$ always contains $\langle \emptyset, c\rangle$ for all $c \in C$. The replacement of $\langle \emptyset, c\rangle$ by c even squares well with our basic intuition about the function space constructor: we are simply saying that a piece of information we can get about the result $f(x)$ of applying a function without knowing anything about x is recorded as itself rather than paired with $\emptyset$. And our hopes for embedding $\mathbf{C}$ in $\mathbf{B} \to \mathbf{C}$ in this way are improved when we observe that, in general

$$(\mathbf{B} \to \mathbf{C})\big|\big\backslash(\{\emptyset\} \times C) \equiv \mathbf{C}$$

via the obvious bijection that sends $\langle \emptyset, c\rangle$ to c.

There is one problem with this, the same problem we encountered when we first defined the lifting constructor, namely that of confusion between tokens. For what are we to do if one of the tokens of $\mathbf{C}$ has the form $\langle X, c\rangle$ for $\emptyset \neq X \in Con_B$ and $c \in C$? We cannot adopt the same solution as before, which was to tag distinct names onto the two types of token we wished to keep apart, since then the tokens of $\mathbf{C}$ would generally not be a subset of those of the function space. And however elaborate a name we choose to tag onto the other tokens, we cannot stop ones of that form turning up in C.

One solution to this dilemma is only to expect $\mathbf{C} \trianglelefteq (\mathbf{B} \mapsto \mathbf{C})$ when the tokens of $\mathbf{C}$ are such as not to allow confusion, for example because C contains no ordered pairs.

Definition
If X is any set, define

$$\Phi(X) = \{a \in X \mid a = \langle b, c\rangle \text{ for any objects } b \text{ and } c\}$$
$$\Psi(X) = X - \Phi(X) \ .$$

($\Phi(X)$ is the set of all ordered pairs in X, $\Psi(X)$ is the set of all elements of X that are not ordered pairs.) In general, $X = \Phi(X) \cup \Psi(X)$ and $\Phi(X) \cap \Psi(X) = \emptyset$. ∎

The final solution we adopt is only to replace the $\mathbf{B} \to \mathbf{C}$ token $\langle \emptyset, c \rangle$ by c when $c \in \Psi(C)$ and could therefore not be confused with any other token.

Definition
The alternative function space constructor is defined as follows. If $\mathbf{B} = \langle B, Con_B, \vdash_B \rangle$ and $\mathbf{C} = \langle C, Con_C, \vdash_C \rangle$ are two information systems, then we define $\mathbf{B} \mapsto \mathbf{C}$ to be the triple $\langle A, Con_A, \vdash_A \rangle$, where

$$\begin{array}{rcl} A & = & \Psi(C) \cup ((Con_B \times C) - (\{\emptyset\} \times \Psi(C))) \\ Z \in Con_A & \Leftrightarrow & \forall \{c_1, \ldots, c_r, \langle X_1, c'_1 \rangle, \ldots, \langle X_s, c'_s \rangle\} \subseteq Z. \\ & & \bigcup_{i=1}^{s} X_i \in Con_B \;\Rightarrow\; \{c_1, \ldots, c_r, c'_1, \ldots, c'_s\} \in Con_C \end{array}$$

and, if $Z = \{c_1, \ldots, c_r, \langle X_1, c'_1 \rangle, \ldots, \langle X_s, c'_s \rangle\} \in Con_A$,

$$\begin{array}{rcl} Z \vdash_A c & \Leftrightarrow & \{c_1, \ldots, c_r\} \cup \{c'_i \mid \emptyset \vdash_B X_i\} \vdash_C c \\ Z \vdash_A \langle X, c \rangle & \Leftrightarrow & \{c_1, \ldots c_r\} \cup \{c'_i \mid X \vdash_B X_i\} \vdash_C c\,. \end{array}$$

Where, above, it is in a place properly occupied by an element of A, c denotes any element of $\Psi(C)$. ∎

5.2.2 PROPOSITION
$\mapsto$ is a well-defined continuous domain constructor such that, for all $\mathbf{B}$ and $\mathbf{C}$,

$$\mathbf{B} \mapsto \mathbf{C} \equiv \mathbf{B} \to \mathbf{C}\,.$$

PROOF
The last part of this result follows from the way $\mapsto$ was put together. If $\langle A, Con, \vdash \rangle$ is as in the definition above, we know that the map $\theta : A \to Con_B \times C$ defined

$$\theta(a) = \begin{cases} \langle \emptyset, a \rangle & \text{if } a \in \Psi(C) \\ a & \text{otherwise} \end{cases}$$

is a bijection. And it is clear that it is an isomorphism between $\langle A, Con, \vdash \rangle$ and the usual function space $\mathbf{B} \to \mathbf{C}$, since in the definition of $\mapsto$ above, the tokens in $\Psi(C)$ play exactly the roles in the consistency and entailment relations that their images under θ do in $\mathbf{B} \to \mathbf{C}$. Since we already know that $\to$ is well-defined, the existence

of this isomorphism actually proves that $\langle A, Con, \vdash\rangle$ is an information system, so that $\mapsto$ is well-defined.

The only thing left to prove is that $\mapsto$ is continuous. Last time we studied the function space constructor we concentrated on the left-hand argument. This time we will concentrate on the right-hand one, since the operation of the left-hand one has not really changed and it is the right-hand one that we are applying the Ψ trick to.

So suppose that $\mathbf{B}$ is fixed and that $\mathbf{C} \trianglelefteq \mathbf{C}'$. Now it is easy to see that, since $C \subseteq C'$, we get

$$\Phi(C) \subseteq \Phi(C') \quad \text{and} \quad \Psi(C) \subseteq \Psi(C')\,.$$

The token-sets A and A' of the domains $\mathbf{B} \mapsto \mathbf{C}$ and $\mathbf{B} \mapsto \mathbf{C}'$ can be written

$$\begin{array}{ll} \Psi(C) \cup (\{\emptyset\} \times \Phi(C)) \cup ((Con_B - \{\emptyset\}) \times C) & \text{and} \\ \Psi(C') \cup (\{\emptyset\} \times \Phi(C')) \cup ((Con_B - \{\emptyset\}) \times C') & \end{array}$$

from which it is clear that $A \subseteq A'$. Rather than prove the Con and $\vdash$ clauses of monotonicity directly we will again appeal to our isomorphisms $\theta : A \to Con_B \times C$ and $\theta' : A' \to Con_B \times C'$; for it is obvious from their definitions that $\theta = \theta'\restriction A$. We can then appeal to the result of Exercise 4.2.7 to show that $\mapsto$ is monotone in its second argument.

The continuity (on token sets) of the second argument follows straightforwardly when we observe that, if $\mathcal{C}$ is any set of sets,

$$\begin{array}{lcl} \Phi(\bigcup \mathcal{C}) & = & \bigcup\{\Phi(C) \mid C \in \mathcal{C}\} \quad \text{and} \\ \Psi(\bigcup \mathcal{C}) & = & \bigcup\{\Psi(C) \mid C \in \mathcal{C}\}\,. \end{array}$$

This completes our proof of the proposition. ∎

We are aiming to get $\mathbf{C} \trianglelefteq (\mathbf{C} \mapsto \mathbf{C})$ for some nontrivial $\mathbf{C}$. That this is possible is shown by the following pair of lemmas.

5.2.3 LEMMA
If $\mathbf{B}$ and $\mathbf{C}$ are any information systems, then

$$(\mathbf{B} \mapsto \mathbf{C})\restriction \Psi(C) = \mathbf{C}\restriction \Psi(C)\,.$$

PROOF
To see this all we have to do is to examine the consistency and entailment relations of the left-hand side, which are just those from the definition of $\mapsto$ above with all

references to pairs $\langle X, c\rangle$ removed:

$$\{c_1, \ldots, c_r\} \in Con \Leftrightarrow \{c_1, \ldots, c_r\} \in Con_C$$

$$\{c_1, \ldots, c_r\} \vdash c \Leftrightarrow \{c_1, \ldots, c_r\} \vdash_C c\ .$$

These are plainly those of the right-hand side since the tokens c, c_i in question are those of $\Psi(C)$. ∎

5.2.4 LEMMA
If $\mathbf{A}$ is any information system then there is another, $\mathbf{C}$, such that $\mathbf{A} \equiv \mathbf{C}$ and $\Psi(C) = C$.

PROOF
All we have to do is to find a way of systematically altering the elements of A so that they remain distinct and are never mapped to ordered pairs. One way of doing this is to map each a to $\{0, 1, \{2, a\}\}$ which cannot be an ordered pair because it is always a three-element set, while the ordered pair $\{\{x\}, \{x, y\}\}$ has at most two. ∎

This discussion has all been leading up to the following result, which shows that domains of the type we are looking for really do exist.

5.2.5 THEOREM
If P is any algebraic, consistently complete cpo then there is another one Q such that Q is order isomorphic to the continuous function space $Q \to Q$ and there is an (order-preserving) embedding of P into Q.

PROOF
By Theorem 3.2.5 and Lemma 5.2.4, there is an information system $\mathbf{C}_0$ such that $|\ \mathbf{C}_0\ |$ is order isomorphic to P and $\Psi(C_0) = C_0$. Lemma 5.2.3 then tells us that $\mathbf{C}_0 \trianglelefteq \mathbf{C}_0 \mapsto \mathbf{C}_0 = \mathbf{C}_1$. Define $\mathbf{C}_{n+1} = \mathbf{C}_n \mapsto \mathbf{C}_n$ for all n.

Then, since the domain constructor $F(\mathbf{A}) = \mathbf{A} \mapsto \mathbf{A}$ is monotone, we get $\mathbf{C}_n \trianglelefteq \mathbf{C}_{n+1}$ for all n exactly as in the proof of Theorem 4.2.8. Thus $\{\mathbf{C}_n \mid n \in \mathbb{N}\}$ is a chain. Again, as in Theorem 4.2.8, the limit $\mathbf{C}^*$ of this chain must be a fixed point of F, namely $\mathbf{C}^* = \mathbf{C}^* \mapsto \mathbf{C}^*$.

Now $\mathbf{C}^* \mapsto \mathbf{C}^* \equiv \mathbf{C}^* \to \mathbf{C}^*$ by Proposition 5.2.2. It follows from Proposition 4.3.1 and Theorem 3.3.2 that $|\ \mathbf{C}^* \mapsto \mathbf{C}^*\ |$ is isomorphic to the continuous function space $|\ \mathbf{C}^*\ | \to |\ \mathbf{C}^*\ |$. Thus, setting $Q = |\ \mathbf{C}^*\ |$ we get what we want, observing that, since $\mathbf{C}_0 \trianglelefteq \mathbf{C}^*$, the required embedding map exists. ∎

It is interesting to examine the projection and injection maps between the $|\ \mathbf{C}_n\ |$ and the resultant cones of elements. To do this we need to understand how the token sets C_n of these domains evolve.

C_0 consists of non-pairs, and C_{n+1} consists of the non-pairs of C_n together with some pairs. It follows that $\Psi(C_n) = C_0$ for all n. Therefore, for all n,

$$C_{n+1} = C_0 \cup ((Con_n \times C_n) - (\{\emptyset\} \times C_0)) .$$

If $x \in| \mathbf{C}_0 |$ then it is just in our starting domain – there is no *a priori* reason to think of it as a function or anything else.

If $x \in| \mathbf{C}_{n+1} |$ then it represents the function from $| \mathbf{C}_n |$ to $| \mathbf{C}_n |$ given by

$$f_x(y) = \{a \mid a \in x \cap C_0 \vee \exists\, X \subseteq^{\text{fin}} y.\langle X, a\rangle \in x\} .$$

Let us first consider $\pi_{1,0}$ and $\iota_{0,1}$. $\pi_{1,0}(x) = x \cap C_0$ which, looking at the function f_x defined above, is precisely $f_x(\overline{\emptyset}^0)$ (where $\overline{\emptyset}^0$ is the least element of $| \mathbf{C}_0 |$). (This is true even if $\overline{\emptyset}^0 \neq \emptyset$ because of the definition of $\mapsto$.)

The definitions of the injection function and of $\mapsto$ give

$$\begin{array}{rcll} \iota_{0,1}(y) & = & \overline{y} & \\ & = & y \cup \{\langle X, c\rangle \mid X \neq \emptyset \wedge y \vdash_0 c\} & \\ & = & y \cup \{\langle X, c\rangle \mid X \neq \emptyset \wedge c \in y\} & \text{as } y \text{ deductively closed} \end{array}$$

The f_x corresponding to this is clearly the constant function which maps everything to y.

So suppose $n > 0$, $x \in| \mathbf{C}_{n+1} |$ and $y = \pi_{n+1,n}(x) = x \cap C_n$. We know that $f_y :| \mathbf{C}_{n-1} | \to | \mathbf{C}_{n-1} |$ is then given by

$$f_y(z) = \{a \mid a \in (x \cap C_n) \cap C_0 \vee \exists\, X \subseteq^{\text{fin}} z.\langle X, a\rangle \in (x \cap C_n)\} .$$

Now $\langle X, a\rangle \in (x \cap C_n)$ iff $\langle X, a\rangle \in x$ and $X \cup \{a\} \subseteq C_{n-1}$. Since $X \subseteq^{\text{fin}} z \in| \mathbf{C}_{n-1} |$ implies $X \subseteq^{\text{fin}} C_{n-1}$ we can re-write the above as

$$f_y(z) = \{a \mid a \in x \cap C_0 \vee \exists\, X \subseteq^{\text{fin}} z.\langle X, a\rangle \in x\} \cap C_{n-1} .$$

Suppose $z \in| \mathbf{C}_{n-1} |$ and $X \subseteq^{\text{fin}} \iota_{n-1,n}(z) = \overline{z}$. Then there is $X' \subseteq^{\text{fin}} z$ such that $X' \vdash_n X$, and hence if $a \in C_n$ then $\{\langle X, a\rangle\} \vdash_{n+1} \langle X', a\rangle$. It follows that if $X \subseteq^{\text{fin}} \iota_{n-1,n}(z)$ and $\langle X, a\rangle \in x$ then, by deductive closure of x, there is $X' \subseteq^{\text{fin}} z$ such that $\langle X', a\rangle \in x$. It follows that we can re-write the above in the following form.

$$f_y(z) = \{a \mid a \in x \cap C_0 \vee \exists\, X \subseteq^{\text{fin}} \iota_{n-1,n}(z).\langle X, a\rangle \in x\} \cap C_{n-1} ,$$

which is in turn equivalent to

$$f_y = \pi_{n,n-1} \circ f_x \circ \iota_{n-1,n} .$$

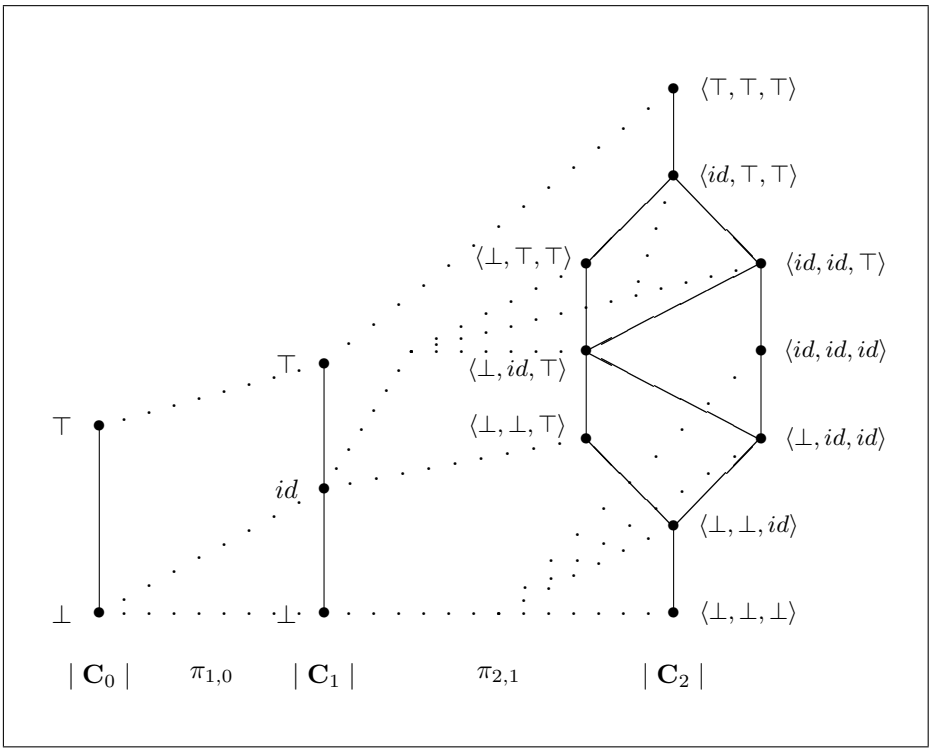

Figure 5.3 The first iterates to $\mathbf{C} = \mathbf{C} \mapsto \mathbf{C}$.

Thus the function represented by $\pi_{n+1,n}(x)$ can be determined from that represented by x in an elegant way via the previous projection and injection.

Also, if $y \in |\ \mathbf{C}_n\ |$ and $x \in |\ \mathbf{C}_{n+1}\ |$ are such that $x = \iota_{n,n+1}(y)$ then a similar analysis shows that

$$f_x = \iota_{n-1,n} \circ f_y \circ \pi_{n,n-1}\ .$$

The domains $|\ \mathbf{C}_n\ |$ always grow very rapidly in size except when $|\ \mathbf{C}_0\ |$ has one element (and so do all $|\ \mathbf{C}_n\ |$). The first three domains for the case when $|\ \mathbf{C}_0\ |$ has two elements, and the associated projection functions, are illustrated in Figure 5.3. The elements of $|\ \mathbf{C}_1\ |$ are $\bot$ and $\top$ the constant functions that map both elements of $|\ \mathbf{C}_0\ |$ to its lesser and greater element respectively, plus *id*, the identity function on $|\ \mathbf{C}_0\ |$. The elements of $|\ \mathbf{C}_2\ |$ are denoted by a triple: $\langle a, b, c\rangle$ denoting the values of the corresponding function at $\bot$, *id* and $\top$. There are no less than 120549 elements in $|\ \mathbf{C}_3\ |$.

There is one feature of the above development which is very interesting from a historical angle. For in fact, it brings domain theory round full circle to Scott's

first models of the λ-calculus from which the subject of domain theory evolved. His construction there was based on a complete lattice D_0. He then defined D_{n+1} to be the continuous function space $D_n \to D_n$. The limit domain D_∞ was found by *defining* projection and injection functions $\pi_{m,n}$, $\iota_{n,m}$ to be precisely the same functions as were given to us above by our usual theory:

$$\begin{array}{rcll}
\pi_{1,0}(f) & = & f(\bot) & \text{where } \bot \text{ is minimal in } D_0 \\
\iota_{0,1}(x) & = & \lambda\, y.x & \text{the constant function} \\
\pi_{n+2,n+1}(f) & = & \pi_{n+1,n} \circ f \circ \iota_{n,n+1} & \\
\iota_{n+1,n+2}(f) & = & \iota_{n,n+1} \circ f \circ \pi_{n+1,n} & \\
\pi_{n,n} & = & id_n & \text{the identity function} \\
\iota_{n,n} & = & id_n & \text{the identity function} \\
\iota_{n,m} & = & \iota_{m-1,m} \circ \ldots \circ \iota_{n,n+1} & \text{for } n < m \\
\pi_{m,n} & = & \pi_{n+1,n} \circ \ldots \circ \pi_{m,m-1} & \text{for } n < m.
\end{array}$$

Scott then identified D_∞ with the set of all cones in this system and proved that D_∞ is order isomorphic to the continuous function space $D_\infty \to D_\infty$. The curious thing is that Scott ended up with exactly the same partial order (D_∞) starting from a given D_0 as we did ($| \mathbf{C}^* |$) starting from a $| \mathbf{C}_0 |$ order isomorphic to D_0. We can tell this because we know inductively that each $| \mathbf{C}_n |$ is order isomorphic to the corresponding D_n; and we know that the projection functions that define D_∞ are (under these isomorphisms) exactly the same projections that characterise $| \mathbf{C}^* |$.

This discussion of solutions to the equivalence $\mathbf{C} \equiv \mathbf{C} \to \mathbf{C}$ should have helped the reader to see more deeply into the structures of information systems. It should have brought home the fact that, in information systems and their domain constructors, we have rather concrete representations of the partial orders and the constructions over them. For we have seen that there are different choices for these representations that lead to quite different conclusions.

Everything that has been done in this section for the function space constructor could have been done for the other ones. The definitions we have given of all the standard versions of constructors are sufficiently 'constructive' in a set-theoretic sense that domain equations involving them will tend only to have a single solution. If, for any reason, it is desired to find further solutions to the underlying partial order isomorphisms, it will generally be possible to find some coding trick like the one used in this section to re-define the constructors appropriately (see Exercise 5.2.5 below). However, except for the above example we know of no legitimate reason for wanting to do this. This is just as well, since it is at least inelegant to vary the definitions of the various domain constructors to get the solutions we want.

Exercises

5.2.1 Show that the construction mapping each a to $\{0, 1, \{2, a\}\}$ used in the proof of Lemma 5.2.4 gives rise to a continuous domain constructor.

5.2.2 Given a domain $\mathbf{A}$, Lemma 5.2.4 and Theorem 5.2.5 represent the construction of a domain $\mathbf{C}^* = C(\mathbf{A})$ which is equal to $\mathbf{C}^* \mapsto \mathbf{C}^*$ and which has $\mathbf{A}$ isomorphic to a subspace.

(a) Show that, if $\mathbf{A} \equiv \mathbf{A}'$ then $C(\mathbf{A}) \equiv C(\mathbf{A}')$ and that if $\mathbf{A} \cong \mathbf{A}'$ then $C(\mathbf{A}) \cong C(\mathbf{A}')$.

*(b) Show that there is a non-trivial $\mathbf{A}$ such that $\mathbf{A} \equiv C(\mathbf{A})$.

[Hint: modify the construction in Lemma 5.2.4 so that it is possible that $\mathbf{A} \trianglelefteq C(\mathbf{A})$ *for non-trivial* $\mathbf{A}$*.]*

5.2.3 Let $\mathbf{D}$ be the least solution to the equation

$$\mathbf{D} = (\mathbf{D} \to \mathbf{D})_\uparrow \,.$$

and let $\mathbf{D}_n$ be the approximations to $\mathbf{D}$ defined in the usual way. Discover the structure of the projection and injection maps between the $\mid \mathbf{D}_n \mid$ by analysis similar to that done above for $\mapsto$.
Figure 5.4 is a picture of the domain $\mid \mathbf{D}_3 \mid$. Find out which function each point (other than the least one) represents from $\mid \mathbf{D}_2 \mid$ to itself, and hence determine the parts of the picture that project back to each of the four points of $\mid \mathbf{D}_2 \mid$.

5.2.4 Write a computer program to verify that the next domain in the sequence begun in Figure 5.3 has 120549 elements. How many of these project back to each of the 10 elements of $\mid \mathbf{C}_2 \mid$?

5.2.5 Show that the two domain equations

$$\mathbf{D} = \mathbf{D} \times \mathbf{D} \quad \text{and} \quad \mathbf{E} = \mathbf{E} \oplus \mathbf{E}$$

have only the trivial solution $\perp$. Find non-trivial domains $\mathbf{D}'$ and $\mathbf{E}'$ such that

$$\mathbf{D} \equiv \mathbf{D} \times \mathbf{D} \quad \text{and} \quad \mathbf{E} \equiv \mathbf{E} \oplus \mathbf{E} \,.$$

[Hint: It is not necessary to define alternative versions of the constructors to solve this question!]

*5.2.6 Give alternative versions of $\oplus$ and $\times$ which allow you to construct a non-trivial $\mathbf{A}$ such that

$$\mathbf{A} \equiv \mathbf{A} \oplus (\mathbf{A} \times \mathbf{A}) \,.$$

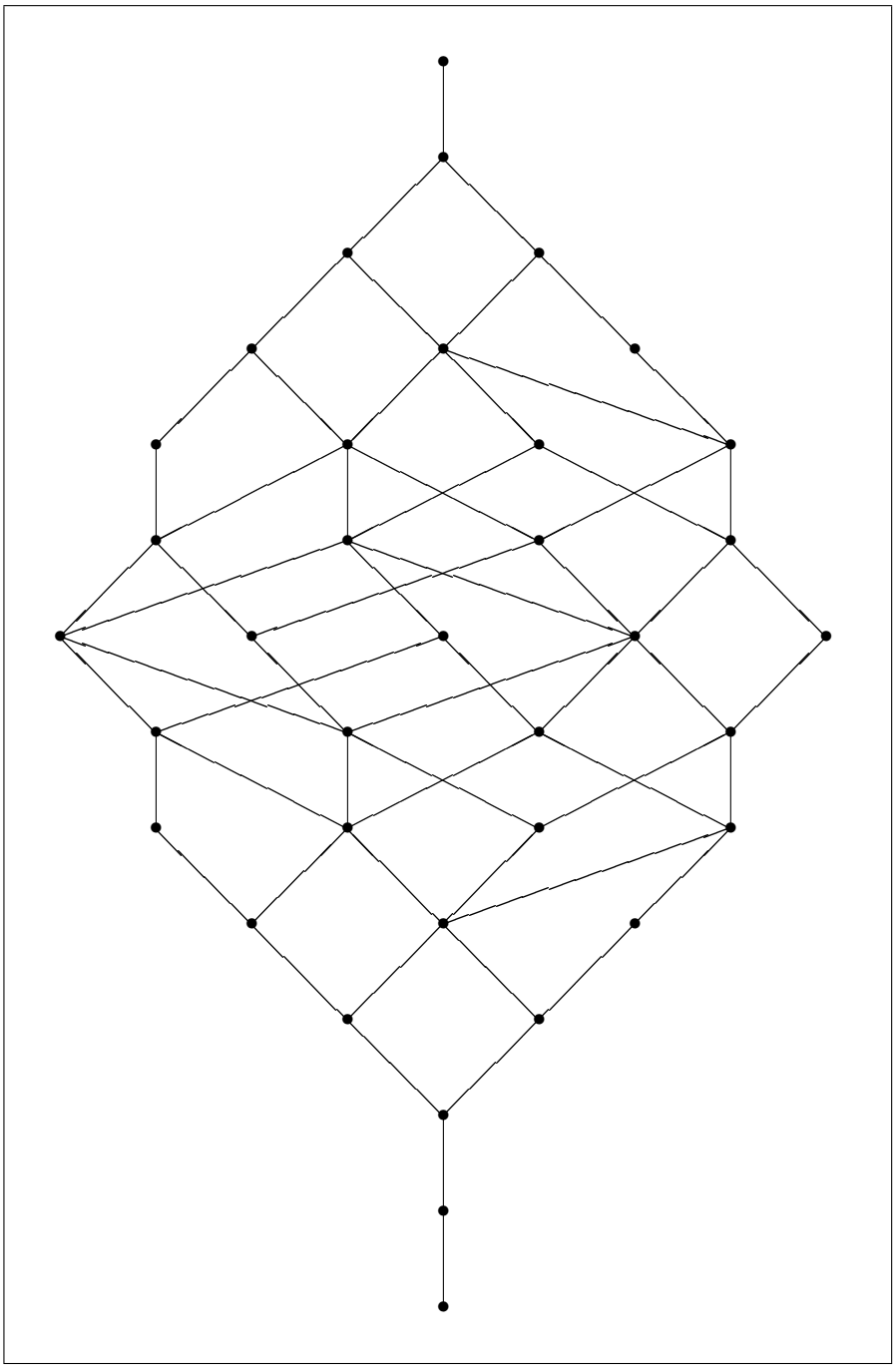

Figure 5.4 The third iterate to $\mathbf{D} = (\mathbf{D} \to \mathbf{D})_{\uparrow}$ – see Exercise 5.2.3.

Chapter 6

Powerdomains

At the start of Chapter 4, when we identified a number of domain constructors which it would be useful to have, one of them was powerset or *powerdomain*. The definition of powerdomain constructors turns out to be rather more difficult than any of the others we have met so far. Therefore this whole rather long chapter is devoted to the subject. In the first section we introduce the various orders used for powerdomains, and the philosophies behind them. Then, in the following sections, we devise and analyse the corresponding domain constructors.

Necessarily the mathematics in the later parts of this chapter is rather deeper than that needed earlier. The reader who is new to domain theory may prefer to omit it on the first reading of the book.

6.1 Introduction: powerdomain orders

The main application of powerdomains in semantics is in the study of *nondeterminism*. A program is said to be nondeterministic if it is unpredictable; if the outcome of running the program is not fully determined by its inputs. There are a number of sources of nondeterminism in real languages, notably parallel ones. The point is, that to give a semantics to a nondeterministic program, we really have to describe it by the set of its possible outcomes.

Consider, for example, a language describing processes that interact with their users via a succession of inputs and outputs. A process starts in some initial state. Whatever state it is in, on each cycle it inspects the value of its inputs then gives some output and takes up some new state. (The output at each stage and the new state may depend both on the old state and on the input.) If these processes are always deterministic then an appropriate semantic domain is given by the equation

$$\mathbf{D} = \mathbf{I} \to (\mathbf{O} \times \mathbf{D})$$

where **I** and **O** are respectively the domains of inputs and outputs. Here **D** is the domain of states: a state is a function from inputs to output/state pairs. However, if they are nondeterministic with either the new state or both the output and new state possibly varying unpredictably then we would want one of the following domains

$$\mathbf{D}_1 = \mathbf{I} \to (\mathbf{O} \times \mathcal{P}(\mathbf{D}_1))$$

$$\mathbf{D}_2 = \mathbf{I} \to \mathcal{P}(\mathbf{O} \times \mathbf{D}_2)$$

where $\mathcal{P}(\cdot)$ is some powerdomain constructor. For applications like this we thus seem to need to be able to solve domain equations through the powerdomain constructor.

Cantor's theorem tells us that no set can be put into 1-1 correspondence with its powerset. But any continuous powerdomain constructor $\mathcal{P}(\cdot)$ necessarily leads to a solution to the domain equation

$$\mathbf{A} = \mathcal{P}(\mathbf{A})\,.$$

It follows that, as was the case with function space, we have to settle for a reduced powerset – we can only hope to model a class of subsets analogous to the continuous functions. In terms of information systems a continuous function is one whose result on any argument can be determined from its behaviour on finite sets of information about it. It will turn out that we can only model sets S such that whether x is a member of S or not can be determined from which finite sets of information are true of x.

There is another rather more annoying problem that we have to confront in devising powerdomain constructors, namely that of finding an appropriate partial order to give to a powerdomain. This has not worried us in the past since we could, for example, simply choose the pointwise order on function spaces: $f \leq g$ if and only if $\forall\, x.f(x) \leq g(x)$. But, given a partial order Q, there seems to be no entirely satisfactory order we can give to the powerset $\mathcal{P}(Q)$. In this chapter we will study at least four different orders, each with its advantages and disadvantages.

There are of course the subset ($\subseteq$) and superset ($\supseteq$) orders, but these totally ignore the underlying order on Q. For example, it seems natural to suppose that, if $x, y \in Q$ are such that $x \leq y$, then we should have $\{x\} \leq \{y\}$ in $\mathcal{P}(Q)$ (a property that neither $\subseteq$ nor $\supseteq$ can boast). Since these orders did not have all the properties one might have hoped for, people devising powerdomains have generally looked for orders that *were* related to that on the underlying domain. But that has not led to a fully satisfactory answer either, as we shall see. One of the powerdomains we will define later will be based on reverse inclusion ($\supseteq$) (which seems to be more satisfactory than $\subseteq$).

Just how troublesome the quest for the right order is can be seen when we look at the sets $S = \{x, z\}$ and $T = \{y\}$, where $x < y < z$ are three elements of Q. We might reasonably claim that S is weaker than T because each element of T has an element of S less than it. But equally each element of T has an element of S greater than it, so perhaps T is weaker than S. Or, given this confusion, had we better conclude that S and T are in fact incomparable. In fact, all three answers have been arrived at by people constructing powerdomains which are based on the orders of the underlying domains, for there are three different such orders commonly used. Also, these orders (and the associated powerdomains) are each commonly given at least two different names. The first order alluded to above is formalised as

$$S \sqsubseteq_S T \Leftrightarrow \forall\, y \in T. \exists\, x \in S. x \leq y\ .$$

This is commonly called the *Smyth* order, after M.B. Smyth who first considered it as a powerdomain order. It is also called the order of *dæmonic* nondeterminism since it is only interested in the worst elements of a set – it judges whether S is weaker than T only by looking at sufficiently small elements of S and T. The philosophy underlying this order is that we always assume the worst: if we have a set of possible outcomes then we are only concerned that the worst ones are good enough. In particular, for any nonempty set S we have $S \cup \{\bot\} \sqsubseteq_S \{\bot\}$ as well as $\{\bot\} \sqsubseteq_S S \cup \{\bot\}$, so that all sets that have the least element in them are as bad as each other.

The dual of this order is the following

$$S \sqsubseteq_H T \Leftrightarrow \forall\, x \in S. \exists\, y \in T. x \leq y\ .$$

It is said to be the order of *angelic* nondeterminism since it is only interested in the best (maximal) elements of S and T. In particular, for a nonempty set S we always have $S \sqsubseteq_H S \cup \{\bot\}$ as well as the more obvious $S \cup \{\bot\} \sqsubseteq_H S$. Thus this order seems to say that we always assume that a program will behave as well as possible. Not surprisingly this order has found less uses than $\sqsubseteq_S$ in a world where we are usually concerned with proving correctness in every case of a nondeterministic process – and are therefore more likely to be interested in what the *worst* elements of a set are than the *best*. One case where this is not true is if we are only interested in the *partial* correctness of a program – namely everything that is observable of it is correct – and we are not interested in whether it actually does anything. Here, an object in $|\ \mathbf{A}\ |$ with more observable of it (i.e., which is higher up the order) is less likely to be correct, and so perversely the best possible element is the least in the partial order on $|\ \mathbf{A}\ |$. To test whether a nondeterministic program is partially correct it is thus enough to inspect the maximal elements of its set of outcomes.

It was because this philosophy was related to that of work by Hoare on partial correctness that G.D. Plotkin christened this order the *Hoare* order.

In fact the first order to be used (by Plotkin) was the combination of these two. It is usually called the *Egli-Milner* order after its inventors.

$$S \mathrel{\underline{\lesssim}}_{EM} T \Leftrightarrow S \mathrel{\underline{\lesssim}}_S T \wedge S \mathrel{\underline{\lesssim}}_H T \,.$$

Clearly this is a sharper order than either of the other two. The resulting powerdomain has been called that of *erratic* nondeterminism (presumably because it is no longer assumed to be angelic or dæmonic). The powerdomain derived from this order is often called the *Plotkin* powerdomain after its inventor.

For all three of these orders it is true that there are distinct sets S and T such that $S \mathrel{\underline{\lesssim}} T$ and $T \mathrel{\underline{\lesssim}} S$. For example, if $x < y < z$, $S = \{x, z\}$ and $T = \{x, y, z\}$, then this is true of all three orders. However they are all transitive, namely

$$R \mathrel{\underline{\lesssim}} S \wedge S \mathrel{\underline{\lesssim}} T \Rightarrow R \mathrel{\underline{\lesssim}} T \,.$$

Consider, for example, the case of $\mathrel{\underline{\lesssim}}_S$. If $R \mathrel{\underline{\lesssim}}_S S$, $S \mathrel{\underline{\lesssim}}_S T$ and $x \in T$ then $\exists\, y \in S . y \leq x$. Hence there exists $z \in R$ such that $z \leq y$, and hence $z \leq x$ as required to prove $R \mathrel{\underline{\lesssim}}_S T$. The fact that $\mathrel{\underline{\lesssim}}_H$ is transitive follows by symmetry, and that $\mathrel{\underline{\lesssim}}_{EM}$ is then comes from the fact that $\mathrel{\underline{\lesssim}}_S$ and $\mathrel{\underline{\lesssim}}_H$ are. All three orders are trivially reflexive in that $S \mathrel{\underline{\lesssim}} S$ for all S.

A transitive reflexive relation like these is called a *preorder* or sometimes a *quasi-order*. Given a preorder $\mathrel{\underline{\lesssim}}$ one can define an equivalence relation $\sim$ by setting

$$S \sim T \Leftrightarrow S \mathrel{\underline{\lesssim}} T \wedge T \mathrel{\underline{\lesssim}} S \,.$$

The three equivalences derived from the powerdomain orders will be denoted $\sim_S$, $\sim_H$ and $\sim_{EM}$ respectively. Two sets are equivalent under one of these if they are indistinguishable under the philosophy of the appropriate order, for example pessimistic in the case of the Smyth order. Given any preorder $\mathrel{\underline{\lesssim}}$ on a set Q, one can define a partial order $\sqsubseteq$ on $Q/\!\sim$, the set of all equivalence classes under $\sim$, by defining $\overline{x} \sqsubseteq \overline{y}$ if and only if $x \mathrel{\underline{\lesssim}} y$ – this is easily seen to be unambiguous since $\mathrel{\underline{\lesssim}}$ is transitive. An order on equivalence classes is not a particularly attractive thing to work with, so where possible one tries to pick some special representative of each class and recast the order as an order on these. Fortunately it is possible to find these for all three orders.

Definition

If Q is a partial order and $S \subseteq Q$, then we define

$$\begin{array}{rcl} S\uparrow & = & \{x \in Q \mid \exists\, y \in S.y \leq x\} \\ S\downarrow & = & \{x \in Q \mid \exists\, y \in S.y \geq x\} \\ ccl(S) & = & \{x \in Q \mid \exists\, y, z \in S.y \leq x \leq z\} \end{array}$$

$S\uparrow$ is the *upwards* closure of S, $S\downarrow$ is its *downwards* closure and $ccl(S)$ is its *convex* closure. S is said to be upwards-, downwards- or convex-closed if $S = S\uparrow$, $S = S\downarrow$ or $S = ccl(S)$ respectively.

6.1.1 LEMMA
If Q is a partial order and $S, T \subseteq Q$, then we have the following

(a) $S \sim_S S\uparrow$, and $S \sim_S T$ if and only if $S\uparrow = T\uparrow$;

(b) $S \sim_H S\downarrow$, and $S \sim_H T$ if and only if $S\downarrow = T\downarrow$;

(c) $S \sim_{EM} ccl(S)$, and $S \sim_{EM} T$ if and only if $ccl(S) = ccl(T)$.

PROOF
It is immediately clear that $S\uparrow \sqsubseteq_S S$ since $S\uparrow \supseteq S$, and the fact that $S \sqsubseteq_S S\uparrow$ follows directly from the definition of $S\uparrow = \{x \in Q \mid \exists\, y \in S.y \leq x\}$. This proves that $S \sim_S S\uparrow$. From this and the fact that $\sim_S$ is an equivalence relation it follows that if $S\uparrow = T\uparrow$ then $S \sim_S T$. So suppose $S \sim_S T$ and $x \in S\uparrow$. Then there is $y \in S$ such that $y \leq x$. Now since $T \sqsubseteq_S S$ it follows that there is $z \in T$ such that $z \leq y$, which proves $x \in T\uparrow$. Hence $S\uparrow \subseteq T\uparrow$, and so $S\uparrow = T\uparrow$ by symmetry. This completes the proof of (a).

The proof of (b) is precisely symmetric to that of (a).

For (c), it is clear that $ccl(S)\uparrow = S\uparrow$ and $ccl(S)\downarrow = S\downarrow$. It follows from (a) and (b) that $ccl(S) \sim_S S$ and $ccl(S) \sim_H S$ and hence $ccl(S) \sim_{EM} S$. As in (a) it follows that if $ccl(S) = ccl(T)$ then $S \sim_{EM} T$. If $S \sim_{EM} T$ then certainly $S\uparrow = T\uparrow$ (by (a)) and $S\downarrow = T\downarrow$ (by (b)). But it is not hard to see that in general $ccl(S) = S\uparrow \cap S\downarrow$, so that $ccl(S) = ccl(T)$ as required. This completes the proof. ∎

This result identifies special members of the equivalences of all three preorders: every equivalence class of $\sim_S$ contains one upwards-closed member; each of $\sim_H$ contains one downwards-closed member; and each of $\sim_{EM}$ contains one convex member. Rather than thinking of the three powerdomains as consisting of sets of equivalence classes it is easier to think of them as sets of (respectively) upwards-closed, downwards-closed and convex-closed sets. In future we will generally adopt this simplification. Since the orders are thus turned into true partial orders they will now be denoted $\sqsubseteq_S$, $\sqsubseteq_H$ and $\sqsubseteq_{EM}$. Figure 6.1 illustrates typical elements of the three powerdomains.

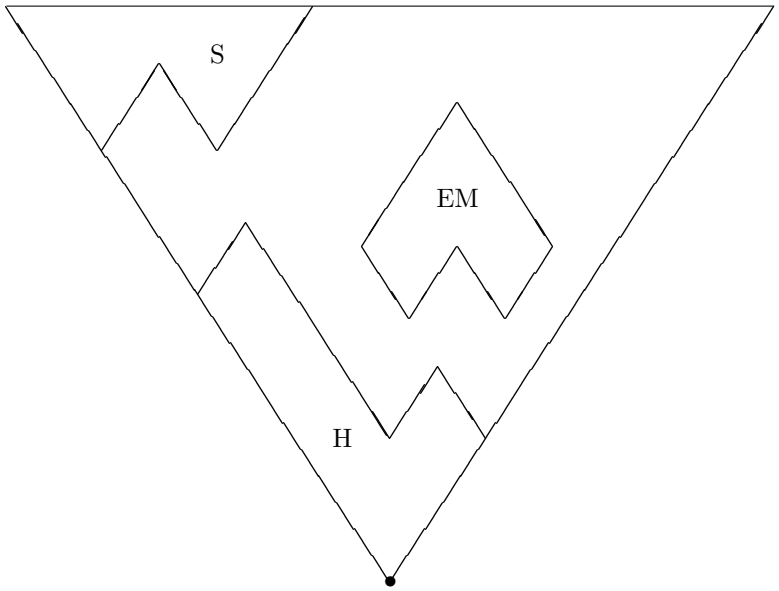

Figure 6.1 Typical elements of the three powerdomains

Since powerdomains are usually used to measure the range of nondeterminism of processes, it is often thought better to exclude the empty set ($\emptyset$) from consideration, for every process must do something (even if that thing is to fail to terminate). In the case of the Plotkin powerdomain it is necessary to exclude it, for if S is a nonempty set we clearly neither have $S \sqsubseteq_{EM} \emptyset$ nor $\emptyset \sqsubseteq_{EM} S$ (and so if the empty set were present there could be no least element). In the other two cases it turns out to be optional. If we were to allow the empty set then it would be the least element of the Hoare powerdomain and the greatest element of the Smyth one. We will generally *exclude* $\emptyset$ from our powerdomains.

As discussed earlier we will need a further property analogous to continuity in functions before we can define the domain constructors themselves. When we do so these will be denoted

$$\mathcal{P}_S(\mathbf{A}) \qquad \mathcal{P}_H(\mathbf{A}) \qquad \mathcal{P}_{EM}(\mathbf{A})$$

with the additional constructors

$$\mathcal{P}_S^{\emptyset}(\mathbf{A}) \qquad \text{and} \qquad \mathcal{P}_H^{\emptyset}(\mathbf{A})$$

for the versions of these domains including the empty set. However it is possible to gain a lot of information about what might or might not be possible to achieve with these constructors by examining what the partial orders of upwards- downwards- and convex-closed *arbitrary* subsets of a given partial order look like. For now we will use the same notation for these partial order constructions, replacing $\mathcal{P}$ by $\mathcal{Q}$. Thus, given a partial order X, $\mathcal{Q}_{EM}(X)$ is the set of all nonempty convex subsets of X ordered by $\sqsubseteq_{EM}$.

We can expect that any *finite* subset will be 'continuous', so that these coarse partial order constructions ought to coincide with the real ones for finite partial orders, at least. The exercises below demonstrate some properties and limitations of these orders, and also show that some further restriction really is necessary.

This concludes our preliminary discussion of powerdomains. The next few sections all discuss a specific powerdomain. In each one the powerdomain is defined as a domain constructor, we then define and prove the correspondence between the elements of the information system thus created and the powerdomain itself, and finally define the basic set-theoretic operations over the powerdomain.

Exercises

6.1.1 Let $X = \{x, y, z\}$ be ordered $x < y < z$. Show that there is no partial order on the full powerset $\mathcal{P}(X)$ that makes both the singleton set operator $(X \to \mathcal{P}(X))$ and the union operator $((\mathcal{P}(X) \times \mathcal{P}(X)) \to \mathcal{P}(X))$ monotone.

6.1.2 Let X be an arbitrary partial order. Identify the least elements of $\mathcal{Q}_S(X)$, $\mathcal{Q}_H(X)$ and $\mathcal{Q}_{EM}(X)$. Under what circumstances does each of them have a greatest element?

6.1.3 Show by means of example that $\mathcal{Q}_S(X)$ need not be complete when X is a cpo.

6.1.4 If X is a finite linear order, show that both $\mathcal{Q}_H(X)$ and $\mathcal{Q}_S(X)$ are order-isomorphic to X. In each case show by means of example that the isomorphism need not hold if X is a linearly ordered infinite cpo.

*6.1.5 Is there a cpo X which is not linearly ordered but which is order isomorphic to $\mathcal{Q}_S(X)$?

*6.1.6 Show that there is no cpo X which is order isomorphic to $\mathcal{Q}_S^{\emptyset}(X)$ or $\mathcal{Q}_H^{\emptyset}(X)$. *[Hint: consider the order type of the region at the top or bottom of X, respectively.]*

6.1.7 Show that if $S \subseteq \mathcal{Q}_H(X)$ is nonempty then $\bigcup S$ is the least upper bound of S. Find a similar formula for least upper bounds in $\mathcal{Q}_S(X)$.

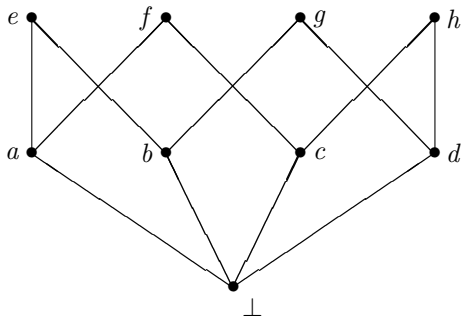

Figure 6.2 Domain for Exercise 6.1.8

6.1.8 Consider the domain X in Figure 6.2. Find two subsets of $\{a, b, c, d\}$ that have an upper bound in $\mathcal{Q}_{EM}(X)$ but no *least* upper bound. This exercise shows that the Plotkin domain constructor does not preserve the property of consistent completeness, which means that we cannot expect to define it, at least in a pure form, over information systems.

6.1.9 Show that no partial order X with more than one element is order isomorphic to $\mathcal{Q}_{EM}(X)$.
[Hint: consider the largest antichain in X (an antichain is a subset no two members of which are comparable).]

6.2 The Hoare powerdomain

In this section we meet the first, and easier, of the two classical powerdomains that we can properly define. Before we introduce it we ought to meet the analogue of the 'continuity' property of functions that was alluded to above.

If $\mathbf{A}$ is an information system, then $S \subseteq |\ \mathbf{A}\ |$ is said to be *finitely generable* if the following holds

$$\forall\, x \in |\ \mathbf{A}\ | \,.(\forall\, Z \subseteq^{\text{fin}} A.\, \exists\, x' \in S.x \cap Z = x' \cap Z) \Rightarrow x \in S\,.$$

This says that, if it impossible to tell from looking at finite pieces of information about x that it is *not* in S, then $x \in S$. If we accept that only a finite amount of information is observable about any object in a finite time, this is a reasonable restriction to place on the sets that we wish to model in a powerdomain.

We can justify this claim as follows. Suppose S is some set being produced by a computation and x satisfies the above. It is impossible to tell finitely either by looking at positive information about x or at what information is not true of it,

that x does not belong to S. So if we think of the value of S as the limit of what it observable about it in a finite time, then naturally $x \in S$. Another possible situation is where a process is making internal decisions the sum of which will determine its overall behaviour. The set of all possible behaviours will be S. If we postulate that the process can only make finitely many decisions in a finite time, each decision being represented by the outcome either definitely having or definitely not having some token true of it, then König's Lemma (see Section 2.4) tells us that the set of all possible outcomes is finitely generable.

Thus we identify finitely generable sets as the 'computable' ones, just as we identified continuous functions as the suitable subcollection to consider there. Of course, it is precisely this restriction that allows one to avoid the adverse consequences of Cantor's theorem referred to earlier.

This is also a very natural condition to place on sets for which we can hope to have a continuous domain constructor. For the token set of a powerdomain $\mathcal{P}(\mathbf{A})$ can only be built up from finite constructions over the tokens, etc., of $\mathbf{A}$.

The definition of finitely generable sets given above is specific to information systems. In case the reader is interested in how it relates to the classical definition we have included a discussion of this in the Appendix to this chapter. One important result which is proved there is that the finitely generable subsets of the partial order represented by an information system do not depend on the exact structure of the information system, only on the partial order. In other words, if $\mathbf{A} \cong \mathbf{B}$ then the order isomorphism between $|\ \mathbf{A}\ |$ and $|\ \mathbf{B}\ |$ also induces a 1-1 correspondence between the finitely generable elements.

While the definition of finitely generable sets is fresh in our minds it is useful to make a second, related definition that will be needed later.

Definition
If $S \subseteq |\ \mathbf{A}\ |$ then we define $\overline{S}^{fg}$, the *fg*-closure of S, to be $\{x \in |\ \mathbf{A}\ || \forall\, Z \subseteq^{\text{fin}} A.\, \exists\, x' \in S.x \cap Z = x' \cap Z\}$. ∎

The important properties of this operator are described in the following lemma, whose proof is trivial. Its significance will become clearer later in this chapter when we understand more about the mathematics of finitely generable sets.

6.2.1 LEMMA
$\overline{S}^{fg}$ is always finitely generable and is the smallest finitely generable set containing S. ∎

In the Hoare powerdomain we can expect to be interested only in the maximal elements of a set, since this domain consists only of downwards-closed subsets of $|\ \mathbf{A}\ |$. A typical token should give us information about these maximal elements. In this domain the bigger a set is, the higher up the order it is (i.e., $S \sqsubseteq_H T$ if and only

if $S \subseteq T$). Since elements higher up an information system contain more tokens, this leads us to expect that the tokens of $\mathcal{P}_H(\mathbf{A})$ should convey *positive* information about elements of a hypothetical set S. Just about the simplest possibility here turns out to be what works: a token of $\mathcal{P}_H(\mathbf{A})$ is an element X of Con_A conveying the information that some element of S has all of X true of it. Since the whole set $\mid \mathbf{A} \mid$ will 'belong' to the powerdomain, the consistency relation of $\mathcal{P}_H(\mathbf{A})$ is trivial. The entailment relation is also rather easy to discover since, given a set $\{X_1, \ldots, X_n\}$ of these tokens, they may all be referring to different elements of S and so do not interfere.

Definition
Let $\mathbf{A}$ be an information system. Then $\mathcal{P}_H(\mathbf{A})$ is defined to be $\langle B, Con, \vdash \rangle$, where

- $B = Con_A$,
- $Con = \wp(B)$, and
- $\mathcal{X} \vdash X \Leftrightarrow (X \subseteq^{\text{fin}} \overline{\emptyset}_A \vee \exists\, Y \in \mathcal{X}.\, Y \vdash_A X)$. ∎

To define the corresponding domain $(\mathcal{P}_H^{\emptyset}(\mathbf{A}))$ which includes the empty set it is only necessary to remove the first half of the disjunction on the right-hand side of the definition of $\vdash$. The following proposition is very easy to prove along the usual lines.

6.2.2 PROPOSITION
The domain constructors $\mathcal{P}_H(\cdot)$ and $\mathcal{P}_H^{\emptyset}(\cdot)$ are well-defined and continuous. ∎

We need to identify the elements of $\mid \mathcal{P}_H(\mathbf{A}) \mid$ with nonempty finitely generable downwards closed subsets of $\mid \mathbf{A} \mid$. This identification is in fact very straightforward to establish when we recall that the tokens of this new system are supposed to be telling us what sets of information are true of elements of our hypothetical subset S.

Definition
Suppose $z \in \mid \mathcal{P}_H(\mathbf{A}) \mid$ and S is a nonempty, finitely generable and downwards-closed subset of $\mid \mathbf{A} \mid$. Then we define

- $\mid z \mid = \{x \in \mid \mathbf{A} \mid\mid \forall\, X \subseteq^{\text{fin}} x.X \in z\}$, and
- $\theta(S) = \{X \mid \exists\, x \in S.X \subseteq^{\text{fin}} x\}$. ∎

The following result shows that these two maps are order isomorphisms.

6.2.3 THEOREM
The operations $|\cdot|$ and $\theta(\cdot)$ defined above for $\mathcal{P}_H(\mathbf{A})$ are well-defined, and furthermore

(a) $|\,\theta(S)\,| = S$ for all finitely generable nonempty downwards-closed $S \subseteq |\,\mathbf{A}\,|$,

(b) $\theta(|\,z\,|) = z$ for all $z \in |\,\mathcal{P}_H(\mathbf{A})\,|$ and

(c) $|\,\mathcal{P}_H(\mathbf{A})\,|$ is order isomorphic to the set of all finitely generable nonempty downwards-closed subsets of $|\,\mathbf{A}\,|$ ordered by $\sqsubseteq_H$.

PROOF
Since $\emptyset \vdash X$ in $\mathcal{P}_H(\mathbf{A})$ whenever $X \subseteq^{\text{fin}} \overline{\emptyset}_A$ it follows that $\overline{\emptyset}_A \in |\,z\,|$ for all $z \in |\,\mathcal{P}_H(\mathbf{A})\,|$. That $|\,z\,|$ is downwards-closed is trivial, so to prove $|\cdot|$ well-defined it will be enough to prove that $|\,z\,|$ is finitely generable. Suppose $x \in |\,\mathbf{A}\,|$ is such that for all $Z \subseteq^{\text{fin}} A$ there is $x' \in |\,z\,|$ with $x \cap Z = x' \cap Z$. Then in particular, whenever $X \subseteq^{\text{fin}} x$ there must be $x' \in |\,z\,|$ with $X \subseteq^{\text{fin}} x'$, which implies $X \in z$. This implies $x \in |\,z\,|$ as desired.

The fact that θ is well-defined is trivial.

Suppose S is as in part (a). It is easy to see that $S \subseteq |\,\theta(S)\,|$, so suppose $x \in |\,\theta(S)\,|$; we must prove $x \in S$. If not then there would, by the definition of finitely generable, be a $Z \subseteq^{\text{fin}} A$ such that $\forall\, x'.x \cap Z = x' \cap Z \Rightarrow x' \notin S$. Now since $x \in |\,\theta(S)\,|$ and $x \cap Z \subseteq^{\text{fin}} x$ there must be $y \in S$ such that $x \cap Z \subseteq^{\text{fin}} y$. But since S is downwards-closed it follows that $x' = \overline{x \cap Z} \in S$, and it is easy to see that $x' \cap Z = x \cap Z$ in contradiction to what was assumed above. We can thus conclude that $S = |\,\theta(S)\,|$.

Suppose z is as in part (b). If $X \in \theta(|\,z\,|)$ then there is $x \in |\,z\,|$ such that $X \subseteq^{\text{fin}} x$. But $x \in |\,z\,|$ and $X \subseteq^{\text{fin}} x$ implies $X \in z$. It follows that $\theta(|\,z\,|) \subseteq z$. For the reverse, suppose that $X \in z$. Then certainly $Y \in z$ whenever $X \vdash_A Y$, by definition of $\vdash$ (for $\mathcal{P}_H(\cdot)$) above. It follows that $\{\,Y \mid Y \subseteq^{\text{fin}} \overline{X}\} \subseteq z$ and hence $\overline{X} \in |\,z\,|$. This proves $X \in \theta(|\,z\,|)$, completing the proof of (b).

Parts (a) and (b) prove that θ and $|\cdot|$ are inverses of each other and are thus bijections. That they are both order-preserving is an elementary deduction from the fact that, if S and T are both downwards-closed, then $S \sqsubseteq_H T$ if and only if $S \subseteq T$. ∎

The above result shows that the domain constructor we have defined really does the job required of it.

Having constructed a powerdomain we must now discover how to use it, or in other words how the basic set operations are defined over it. Clearly we are restricted by the types of set that are allowed in the powerdomain: we must always create (nonempty) finitely generable and downwards closed sets. The most obvious

operations are singleton set (mapping $|\ \mathbf{A}\ |$ to $|\ \mathcal{P}_H(\mathbf{A})\ |$), intersection and union (both mapping $|\ \mathcal{P}_H(\mathbf{A})\ | \times |\ \mathcal{P}_H(\mathbf{A})\ |$ to $|\ \mathcal{P}_H(\mathbf{A})\ |$). In the Hoare powerdomain (and clearly in the Smyth one as well) we cannot hope to define a singleton set operation which is 'natural'. For the obvious definition below clearly identifies the 'singleton' $\{x\}_H$ with the set $\{y \mid y \subseteq x\}$. In this powerdomain the intersection and union operations both work well, though.

Definitions
Suppose $x \in |\ \mathbf{A}\ |$ and $z_1, z_2 \in |\ \mathcal{P}_H(\mathbf{A})\ |$. Then we define

- $\{x\}_H = \{Y \in Con_A \mid Y \subseteq x\}$;
- $z_1 \Cup_H z_2 = z_1 \cup z_2$, and
- $z_1 \Cap_H z_2 = z_1 \cap z_2$. ∎

The following result shows that, with the exception already noted above, these definitions all work well. Note that we have used variants of the usual set notations for these operations to denote the fact that these are powerdomain operations (working on elements of the powerdomain information system) as opposed to the operations they are modelling. Of course the H subscripts denote the particular powerdomain we are using.

6.2.4 LEMMA
The operations $\{\ \cdot\ \}_H$, $\Cup_H$ and $\Cap_H$ are all well-defined and continuous. Furthermore

(a) $|\ \{x\}_H\ | = \{y \in |\ \mathbf{A}\ || \ y \subseteq x\}$,

(b) $|\ z_1 \Cup_H z_2\ | = |\ z_1\ | \cup |\ z_2\ |$, and

(c) $|\ z_1 \Cap_H z_2\ | = |\ z_1\ | \cap |\ z_2\ |$.

PROOF
The well-definedness and continuity of these operations are all very easy to show, as is everything else perhaps with the exception of the inequality $|\ z_1 \Cup_H z_2\ | \subseteq |\ z_1\ | \cup |\ z_2\ |$. So suppose x belongs to the left-hand side but not to the right-hand side. Then there exist finite subsets X_1 and X_2 of x such that $X_1 \notin z_1$ and $X_2 \notin z_2$. It follows (by the definition of the entailment relation over $\mathcal{P}_H(\mathbf{A})$) that $X_1 \cup X_2 \notin z_1 \cup z_2$, contradicting the assumption that every finite subset of x is in $z_1 \cup z_2$.

It is perhaps worth noting that the union and intersection operations are, in this powerdomain, precisely the least upper bound and greatest lower bound operations. Note also that, even though we are restricting ourselves to nonempty

sets, the intersection operation is always defined because all sets contain the least element $\overline{\emptyset}_A$ of $|\ \mathbf{A}\ |$. ∎

Of course these simple operations are far from being the only ones one might wish to have for manipulating the 'sets' in powerdomains. Some make little sense; for example set difference $S - T$ is clearly impossible over the domain of downwards-closed sets, while others require some restrictions. One construction that is often used in constructing sets is that of applying a function to every element of another set: $\{f(x) \mid x \in S\}$ (often just written $f(S)$, producing a 'lifted' version of f).

Definition
Suppose f is a continuous function from $|\ \mathbf{A}\ |$ to $|\ \mathbf{B}\ |$; we can define the 'lifted' version of this: $\underline{f}_H :|\ \mathcal{P}_H(\mathbf{A})\ |\rightarrow|\ \mathcal{P}_H(\mathbf{B})\ |$ as follows: $\underline{f}_H(z) = \{\, Y \mid \exists\, x \in|\ z\ | \ .\, Y \subseteq^{\text{fin}} f(x)\}$. ∎

Given a general continuous $f :|\ \mathbf{A}\ |\rightarrow|\ \mathbf{B}\ |$, we clearly do not always have $f(S)$ downwards closed when S is. Thus we must expect that $|\ \underline{f}_H(z)\ |$ contains some extra elements on this count. Unfortunately it is sometimes not finitely generable either, so we need to make use of the fg-closure operator defined earlier. For an example, see Exercise 6.2.7 below. It is possible to put additional conditions on f to make sure that $f(S)$ is finitely generable – see the end of the next section.

6.2.5 LEMMA
If f is as in the above definition and $z \in|\ \mathcal{P}_H(\mathbf{A})\ |$ then $\underline{f}_H$ is well-defined, continuous, and furthermore

$$|\ \underline{f}_H(z)\ | = \overline{f(|\ z\ |)\downarrow}^{fg}\ .$$

PROOF
Since $\underline{f}_H(z)$ is obviously a subset of Con_B (the token set of $\mathcal{P}_H(\mathbf{B})$) and the consistency relation of $\mathcal{P}_H(\mathbf{B})$ is always trivial, for well-definedness it is only necessary to show deductive closure. This follows very simply from the facts that $|\ z\ |$ is nonempty and the deductive closure of the $f(x)$.

Since $z \subseteq z' \Rightarrow|\ z\ |\subseteq|\ z'\ |$, $\underline{f}_H$ is obviously monotone. The fact that it is continuous is an easy consequence of the continuity of f: any $Y \in \underline{f}_H(z)$ necessarily is a subset of some $f(\overline{X})$ with $X \in z$.

We know that $f(|\ z\ |) \subseteq|\ \underline{f}_H(z)\ |$ directly from the definition of $\underline{f}_H$, and the facts that $|\ \underline{f}_H(z)\ |$ is known to be downwards closed and finitely generable then give us that $\overline{f(|\ z\ |)\downarrow}^{fg} \subseteq|\ \underline{f}_H(z)\ |$. So suppose $y \in|\ \underline{f}_H(z)\ |$ and $Z \subseteq^{\text{fin}} B$. Then by definition of $\underline{f}_H$ there exists $x \in|\ z\ |$ such that $\overline{Z \cap y} \subseteq f(x)$. Clearly then, $\overline{Z \cap y} \in f(|\ z\ |)\downarrow$ and it is easy to prove that $\overline{Z \cap y} \cap Z = y \cap Z$. Since this holds

for all Z we know that $y \in \overline{f(|\ z\ |)\downarrow}^{fg}$. ∎

This discussion should have given the reader some idea of what can, and what cannot, be done in this powerdomain. The reader wishing to learn more should attempt the exercises below and also study the material in the rest of the chapter, where much of the more general material is common to all powerdomains, not just the ones under consideration.

Exercises

6.2.1 If X is any set, list the finitely generable subsets of the flat domain $|\ \mathbf{flat}(X)\ |$. Hence show that $|\ \mathcal{P}_H(\mathbf{flat}(X))\ |$ always gives a domain order-isomorphic to the full powerset $\mathcal{P}(X)$ of X, ordered by inclusion.

6.2.2 Prove the following order isomorphisms:

(i) $\mathcal{P}_H(\mathbf{A} \oplus \mathbf{B}) \cong (\mathcal{P}_H(\mathbf{A})) \times (\mathcal{P}_H(\mathbf{B}))$, and
(ii) $\mathcal{P}_H(\mathbf{A} + \mathbf{B}) \cong (\mathcal{P}^{\emptyset}_H(\mathbf{A})) \times (\mathcal{P}^{\emptyset}_H(\mathbf{B}))$.

6.2.3 Suppose $|\ \mathbf{A}\ |$ is linearly ordered. Prove that $|\ \mathcal{P}_H(\mathbf{A})\ |$ is also linearly ordered. Show that whenever $z \in |\ \mathcal{P}_H(\mathbf{A})\ |$ then $|\ z\ |$ contains its own supremum (least upper bound), and hence that $\mathcal{P}_H(\mathbf{A}) \cong \mathbf{A}$.

6.2.4 What is the structure of the least solution to the domain equation $\mathbf{A} = \mathcal{P}^{\emptyset}_H(\mathbf{A})$?

6.2.5 Because the elements of the Hoare powerdomain are downwards closed, the function $f_z :|\ \mathbf{A}\ | \to \{0, 1\}$ defined for each $z \in |\ \mathcal{P}_H(\mathbf{A})\ |$ by

$$f_z(x) = \begin{cases} 0 & \text{if } x \in |\ z\ | \\ 1 & \text{if } x \notin |\ z\ | \end{cases}$$

is always monotone. Is it continuous? Compare the constructors $\mathcal{P}^{\emptyset}_H(\cdot)$ and $\mathcal{P}_H(\cdot)$ with (respectively) the constructors mapping $\mathbf{A}$ to $\mathbf{A} \to \ \perp_{\uparrow}$ and $\mathbf{A} \multimap \ \perp_{\uparrow}$. (Recall from Exercise 4.3.3 that $\multimap$ is the *strict* continuous function space constructor.)

6.2.6 Suppose we try to extend the pairwise union operation $\cup_H$ into an operator $\bigcup_H :|\ \mathcal{P}_H(\mathcal{P}_H(\mathbf{A}))\ | \to |\ \mathcal{P}_H(\mathbf{A})\ |$ by defining

$$\bigcup_H \chi = \bigcup \chi\,.$$

Show that this operation is well defined and continuous. Satisfy yourself, by

looking at some (infinite) examples that the equality

$$|\bigsqcup_H \chi| = \bigcup\{|z| \mid z \in |\chi|\}$$

always holds. [We do not yet have the techniques required to prove this result. You might like to return and complete the proof after doing Exercise 6.3.3.] Why is it unreasonable even to try to define a corresponding intersection operator?

6.2.7 Find domains $\mathbf{A}$ and $\mathbf{B}$, a continuous $f : |\mathbf{A}| \to |\mathbf{B}|$ and a downwards closed finitely generable $S \subseteq |\mathbf{A}|$ such that $f(S)\downarrow$ is not finitely generable.
[Hint: let $\mathbf{B} = \mathbf{B}_\uparrow$ *and let* $\mathbf{A}$ *consist of infinitely many incomparable copies of* $\mathbf{B}$ *over a single bottom element.]*

6.3 The Smyth powerdomain

The tokens of the Smyth powerdomain should tell us things about the minimal elements of a set S, rather than the maximal ones. Also we want the largest elements of the powerdomain to correspond with the smallest, or most restricted, subsets. Thus the natural tokens for this domain will be ones that place a restriction on every element of S, constraining it (the element) to be sufficiently large. The simplest solution here seems to be that a token of $\mathcal{P}_S(\mathbf{A})$ is a finite subset Y of A, conveying the information that every element of S has at least one token from Y. If we are to exclude the empty set from the powerdomain it is obviously necessary that we disallow $Y = \emptyset$; but no further restriction is required. The consistency and entailment relations are defined as follows.

Definition
The Smyth powerdomain $\mathcal{P}_S(\mathbf{A})$ is defined to be $\langle B, Con, \vdash \rangle$, where

- $B = \wp(A) - \{\emptyset\}$;
- $\{Y_1, \ldots, Y_n\} \in Con \Leftrightarrow \exists\, a_1 \in Y_1 \ldots \exists\, a_n \in Y_n . \{a_1, \ldots, a_n\} \in Con_A$;
- $\{Y_1, \ldots, Y_n\} \vdash Y \Leftrightarrow \forall\, a_1 \in Y_1 \ldots \forall\, a_n \in Y_n . \{a_1, \ldots, a_n\} \in Con_A \Rightarrow \overline{\{a_1, \ldots, a_n\}} \cap Y \neq \emptyset$.

The Smyth powerdomain with empty set, $\mathcal{P}_S^{\emptyset}(\mathbf{A})$, has the same definition except that $\emptyset$ is included in B and the definition of Con becomes trivial. ∎

In both cases we will be identifying an element z of the powerdomain with the set of all elements of $|\mathbf{A}|$ meeting all the constraints implied by its elements: $\{x \in |\mathbf{A}| \mid \forall\, Y \in z . Y \cap x \neq \emptyset\}$. In $\mathcal{P}_S(\mathbf{A})$ one can thus expect a finite set $\mathcal{Y} = \{Y_1, \ldots, Y_n\}$ of tokens to be consistent if and only if the set $\{x \in |\mathbf{A}| \mid \forall\, Y \in$

$\mathcal{Y}.Y \cap x \neq \emptyset\}$ is nonempty. This is indeed what the above definition says, for on the one hand if $\{a_1, \ldots, a_n\} \in Con_A$ is as in the definition then $\overline{\{a_1, \ldots, a_n\}}$ is the required x, and on the other if $x \cap Y \neq \emptyset$ for all $Y \in \mathcal{Y}$, then choosing $a_i \in x \cap Y_i$ for all i necessarily gives $\{a_1, \ldots, a_n\} \in Con_A$.

Likewise we would expect $\{Y_1, \ldots, Y_n\} \vdash Y$ if and only if, for all $x \in |\mathbf{A}|$, if $Y_i \cap x \neq \emptyset$ for all i then also $Y \cap x \neq \emptyset$. This is exactly what the above definition says, once we observe that this property holds if and only if it holds of all the *smallest* x satisfying the antecedent – and these smallest x are precisely the $\overline{\{a_1, \ldots, a_n\}}$ of the definition.

Thus we have characterised the consistency and entailment relations of the powerdomain by reference to the statements its tokens make about elements of $|\mathbf{A}|$. We will find this useful later on.

6.3.1 PROPOSITION
The constructors $\mathcal{P}_S(\cdot)$ and $\mathcal{P}_S^{\emptyset}(\cdot)$ are well-defined and continuous.

PROOF
We will concentrate on $\mathcal{P}_S(\cdot)$. Axioms 1-4 of information systems are fairly easy to verify. Suppose for example that $\mathcal{Y} \vdash Y$ where $\mathcal{Y} = \{Y_1, \ldots, Y_n\}$. Then, since $\mathcal{Y} \in Con$, there are $a_i \in Y_i$ such that $\{a_1, \ldots, a_n\} \in Con_A$. Since $\mathcal{Y} \vdash Y$, there is some $a \in Y \cap \overline{\{a_1, \ldots, a_n\}}$. Since necessarily $\{a\} \cup \{a_1, \ldots, a_n\} \in Con_A$ it follows that $\mathcal{Y} \cup \{Y\} \in Con$.

Axiom 5 is a bit trickier to prove directly, but it follows reasonably easily from the characterisation demonstrated above. If $\mathcal{Y} \vdash \mathcal{Z}$ and $\mathcal{Z} \vdash Y$ then to prove $\mathcal{Y} \vdash Y$ it is sufficient to prove that, for all $x \in |\mathbf{A}|$, if $Y' \cap x \neq \emptyset$ for all $Y' \in \mathcal{Y}$, then $Y \cap x \neq \emptyset$. But this is easy to see, since we know from the same characterisation that under these circumstances $Z \cap x \neq \emptyset$ for all $Z \in \mathcal{Z}$, and that this implies what we want.

The argument that $\mathcal{P}_S(\cdot)$ is monotone is standard. Note, for example, that if $A \subseteq B$ then $(\wp(A) - \{\emptyset\}) \subseteq (\wp(B) - \{\emptyset\})$. Equally, continuity in token sets follows much as before, with again the fact that we are using only *finite* sets of $\mathbf{A}$'s tokens being crucial. ∎

As in the case of the Hoare powerdomain, we can formally define the relationship between $|\mathcal{P}_S(\mathbf{A})|$ and $\mathcal{P}(|\mathbf{A}|)$ by means of a pair of functions.

Definition
If $S \subseteq |\mathbf{A}|$ is nonempty, upwards closed and finitely generable, and $z \in |\mathcal{P}_S(\mathbf{A})|$ then we define

- $|z| = \{x \in |\mathbf{A}| \mid\mid \forall Y \in z.\, Y \cap x \neq \emptyset\}$, and

- $\theta(S) = \{\, Y \mid \forall\, x \in S.\, Y \cap x \neq \emptyset\}$. ∎

Note that we have used exactly the same notation for these operations as for the corresponding ones over $\mathcal{P}_H(\cdot)$. No confusion arises in practice because the identity of the map in question should always be obvious from the context.

As was the case in the last section, these maps will turn out to be inverses and order isomorphisms. Unfortunately the demonstration that this is so is no longer elementary – specifically it is necessary to appeal to some form of *compactness* argument, implicitly using some form of the axiom of choice. There are two main forms in which one can use the compactness principle which at first sight may seem unconnected, but are in fact really two sides of the same coin. One is *logical* compactness and the other is *topological* compactness. These two ideas are equally applicable here. To illustrate this we have chosen to analyse this powerdomain topologically and the one in the next section logically. We hope the reader will enjoy seeing how these two sorts of argument relate.

The starting point for any topological analysis must be to define a topology, namely a set of open sets closed under arbitrary union and finite intersection (or equivalently their complements, closed sets, that are closed under arbitrary intersection and finite union). Fortunately we are already equipped with a topology on $|\,\mathbf{A}\,|$, since it turns out that the finitely generable sets are the closed sets of a useful topology, which we shall call $\mathcal{T}_{FG}$.

6.3.2 LEMMA
The finitely generable subsets of a domain $|\,\mathbf{A}\,|$ are closed under finite union and arbitrary intersection. Furthermore, whenever $x \neq y$ are two elements of $|\,\mathbf{A}\,|$ then we can find clopen (closed and open) subsets U and V such that $x \in U$, $y \in V$, $U =| \mathbf{A} | - V$. (This last property is known in topology as being 'totally disconnected' and trivially implies that the topology is Hausdorff.)

PROOF
Since $\emptyset$ is trivially finitely generable, it is sufficient for the first part to prove that, if S and T are finitely generable than so is $S \cup T$. If $x \notin S \cup T$ then clearly $x \notin S$ and $x \notin T$ so, since S and T are finitely generable, there must exist finite sets Z_1 and Z_2 such that if $y \cap Z_1 = x \cap Z_1$ then $y \notin S$ and if $y \cap Z_2 = x \cap Z_2$ then $y \notin T$. Setting $Z = Z_1 \cup Z_2$, it is thus clear that if $y \cap Z = x \cap Z$ then $y \notin S$ and $y \notin T$, or in other words $y \notin S \cup T$. This proves that $S \cup T$ is finitely generable.

The whole space $|\,\mathbf{A}\,|$ is also trivially finitely generable, and so to prove closure under arbitrary intersection it will be sufficient to consider the case of $\bigcap W$, where W is a nonempty set of finitely generable sets. If $x \notin \bigcap W$ then there must be some $S \in W$ such that $x \notin S$, and hence some finite set Z such that $y \cap Z = x \cap Z$ implies $y \notin S$. But since $y \notin S$ implies $y \notin \bigcap W$, this means that $\bigcap W$ is indeed

finitely generable.

Suppose x and y are two distinct elements of $|\ \mathbf{A}\ |$, then without loss of generality we may assume that there is some token $a \in x - y$. Set $U = \{z \in|\ \mathbf{A}\ ||\ a \in z\}$ and $V = \{z \in|\ \mathbf{A}\ ||\ a \notin z\}$. It is easy to prove that these satisfy all that is required of them. ∎

A topology is said to be *compact* if every *open cover* (set of open sets whose union is the whole space) has a finite subcover (a finite subset that is also a cover). The equivalent formulation in terms of closed sets is that any set $\mathcal{C}$ of closed sets with the *finite intersection property* ($\bigcap \mathcal{F} \neq \emptyset$ for all $\mathcal{F} \subseteq^{\text{fin}} \mathcal{C}$) has nonempty intersection (i.e., $\bigcap \mathcal{C} \neq \emptyset$). That the two are equivalent is obvious when we note that, if $\mathcal{U}$ is an open cover for a space X with no finite subcover, then $\{X - U \mid U \in \mathcal{U}\}$ has both the finite intersection property and empty intersection. It turns out that the topology $\mathcal{T}_{FG}$ is compact, which is the result we will need to justify our definitions of θ and $|\cdot|$. The next three paragraphs sketch a proof of this compactness result, so any reader who wishes to take this result on trust should skip over them.

A *basis* for the closed sets of a topology is a set $\mathcal{B}$ of closed sets such that every closed set can be written $\bigcap \mathcal{A}$ for some $\mathcal{A} \subseteq \mathcal{B}$. It should by now be easy to see that a basis for $\mathcal{T}_{FG}$ is given by

$$\mathcal{B}_1 = \{\{x \in|\ \mathbf{A}\ ||\ x \cap Z \neq Z'\} \mid Z' \subseteq Z \subseteq^{\text{fin}} A\}\,,$$

where the element determined by Z and Z' is denoted $C_1(Z, Z')$.

Given any set A (in our case the token set of $\mathbf{A}$), one can form a compact topology $\mathcal{T}$ on the full powerset $\mathcal{P}(A)$ by means of the basis

$$\mathcal{B}_2 = \{\{x \subseteq A \mid x \cap Z \neq Z'\} \mid Z' \subseteq Z \subseteq^{\text{fin}} A\}\,.$$

where the element determined by Z and Z' is denoted $C_2(Z, Z')$. The fact that this is a compact topology is deduced from Tychonoff's theorem[1] and the fact that this topology is homeomorphic to the product of A copies of the discrete topology on $\{0, 1\}$. Since the corresponding elements of the bases $\mathcal{B}_1$ and $\mathcal{B}_2$ are related by the simple relationship $C_1(Z, Z') = C_2(Z, Z') \cap |\ \mathbf{A}\ |$ this means that the topology $\mathcal{T}_{FG}$ is actually the subspace topology induced by $(\mathcal{P}(A), \mathcal{T})$ on $|\ \mathbf{A}\ |$.

In fact, $|\ \mathbf{A}\ |$ is closed in $(\mathcal{P}(A), \mathcal{T})$, since it clearly consists precisely of the intersection of the basis elements

$$\{C_2(X, X) \mid X \notin Con_A\} \cup \{C_2(X, X - \{a\}) \mid X \vdash_A a\}\,,$$

which respectively enforce consistency and deductive closure. But it is well known (and very easy to prove) that the subspace topology induced by a compact space

[1] Tychonoff's theorem states that the topological product of any indexed family of compact spaces is compact. It is an equivalent of the axiom of choice.

on one of its closed subsets is itself compact. This is precisely what we wanted to demonstrate.

We are now in a position to prove the following theorem.

6.3.3 THEOREM
The operations $|\cdot|$ and $\theta(\cdot)$ defined above for $\mathcal{P}_S(\cdot)$ are well-defined, and furthermore

(a) $|\theta(S)| = S$ for all finitely generable nonempty and upwards closed $S \subseteq |\mathbf{A}|$,

(b) $\theta(|z|) = z$ for all $z \in |\mathcal{P}_S(\mathbf{A})|$ and

(c) $|\mathcal{P}_S(\mathbf{A})|$ is order isomorphic to the set of all finitely generable nonempty and upwards closed subsets of $|\mathbf{A}|$ ordered by $\sqsubseteq_S$.

PROOF
To prove the well-definedness of $|\cdot|$ we must prove that $|z|$ is always upwards closed, finitely generable and nonempty. The first is trivial: if $x \subseteq y$ and $x \cap Y \neq \emptyset$ for all $Y \in z$ then certainly the same holds of y. The second comes more or less by definition: suppose that $x \notin |z|$. Then, by definition of $|\cdot|$ there must exist $Y \in z$ such that $x \cap Y = \emptyset$. But clearly then, if x' is any other element of $|\mathbf{A}|$ and $x' \cap Y = x \cap Y$ then $x' \notin |z|$.

The proof that $|z|$ is nonempty relies on compactness. For we know that every finite subset $\mathcal{X}$ of z belongs to Con and so, by the argument earlier that justified the definition of Con, there is some $x \in \{x \in |\mathbf{A}| \mid \forall Y \in \mathcal{X}. Y \cap x \neq \emptyset\}$, (a set which clearly equals $|\overline{\mathcal{X}}|$). We know from the last paragraph that all such $|\overline{\mathcal{X}}|$ are finitely generable (and hence closed), and since clearly $|\overline{\mathcal{X} \cup \mathcal{Y}}| = |\overline{\mathcal{X}}| \cap |\overline{\mathcal{Y}}|$ for all $\mathcal{X}, \mathcal{Y}$, we have that the family $\{|\overline{\mathcal{X}}| \mid\mid \mathcal{X} \subseteq^{\text{fin}} z\}$ has the finite intersection property. Compactness tells us that the intersection of this set is nonempty. But since every element of z belongs to some such $\mathcal{X}$ we have that $\bigcap\{|\overline{\mathcal{X}}| \mid\mid \mathcal{X} \subseteq^{\text{fin}} z\} = |z|$, as desired. This completes the proof that $|\cdot|$ is well-defined.

To prove θ well-defined we must prove that $\theta(S)$ is always consistent and deductively closed. Consistency follows directly from the fact that S is nonempty: we know by definition that when $x \in S$ then $x \cap Y \neq \emptyset$ for all $Y \in \theta(S)$. This, and the characterisation of Con derived earlier, give us what we want. Now suppose $\mathcal{X} \subseteq^{\text{fin}} \theta(S)$ and $\mathcal{X} \vdash Y$. If $Y \notin \theta(S)$ then there would be some $x \in S$ such that $x \cap Y = \emptyset$. But we know that, since $\mathcal{X} \vdash Y$, whenever $Y \cap x = \emptyset$ then there is some $Y' \in \mathcal{X}$ such that $Y' \cap x = \emptyset$. It follows that there must be $Y' \in \theta(S)$ and $x \in S$ such that $Y' \cap x = \emptyset$, in direct contradiction to the definition of θ. This completes the proof that θ is well-defined.

We can thus turn to the proof of (a). It follows directly from the definitions that, for all S, $S \subseteq |\theta(S)|$. So suppose $x \notin S$. Since S is upwards closed we

know that $y \in S \Rightarrow y \not\subseteq x$. It follows that for every $y \in S$ we can choose a token (element of A) $a_y \in y - x$. Now, for each element w of S, we define $S_w = \{y \in S \mid a_w \notin y\}$. Each S_w is closed (it is easily seen to be the intersection of two finitely generable sets), and the intersection $\bigcap\{S_w \mid w \in S\}$ is plainly empty since $w \notin S_w$. Compactness thus tells us that these closed sets do *not* have the finite intersection property, or in other words that there is a finite $F \subseteq S$ such that $\emptyset = \bigcap\{S_w \mid w \in F\} = \{y \in S \mid \{a_w \mid w \in F\} \cap y = \emptyset\}$. Let $Y_x = \{a_w \mid w \in F\}$; by construction we know that $x \cap Y_x = \emptyset$ and that $y \cap Y_x \neq \emptyset$ for all $y \in S$. Thus $Y_x \in \theta(S)$ and so $x \notin \mid \theta(S) \mid$, which proves (a).

Part (b) is proved if we can show that the operation $\mid \cdot \mid$ is injective. This is because we know that $\mid z \mid = \mid \theta(\mid z \mid) \mid$ by part (a), which then implies $z = \theta(\mid z \mid)$. So suppose for contradiction that z_1 and z_2 were two different elements of $\mid \mathcal{P}_S(\mathbf{A}) \mid$ such that $\mid z_1 \mid = \mid z_2 \mid$. Without loss of generality we may assume that $Y \in z_1 - z_2$. If $\mathcal{X} \subseteq^{\text{fin}} z_2$ then we know that $\mathcal{X} \not\Vdash Y$ and hence that the set

$$C_{\mathcal{X}} = \{x \in \mid \mathbf{A} \mid\mid x \cap Y = \emptyset \wedge \forall\, Y' \in \mathcal{X}.\, Y' \cap x \neq \emptyset\}$$

is nonempty. But it is easy to see that $C_{\mathcal{X}} \cap C_{\mathcal{Y}} = C_{\mathcal{X} \cup \mathcal{Y}}$ which means that the family of closed sets formed as $\mathcal{X}$ varies have the finite intersection property. Hence their intersection, which is

$$\{x \in \mid \mathbf{A} \mid\mid x \cap Y = \emptyset \wedge \forall\, Y' \in z_2.\, Y' \cap x \neq \emptyset\}$$

is nonempty. But this shows that there is some x which is in $\mid z_2 \mid$ and yet which satisfies $x \cap Y = \emptyset$, which is a contradiction to $x \in \mid z_1 \mid$. This completes the proof of (b).

Parts (a) and (b) show that $\mid \cdot \mid$ and θ are both bijections and are each other's inverses. We will show that they are order isomorphisms. It is clear from their definitions and the fact that, for upwards closed sets, $S \sqsubseteq_S T \Leftrightarrow S \supseteq T$, that $z \subseteq z' \Rightarrow \mid z \mid \sqsubseteq_S \mid z' \mid$ and that $S \sqsubseteq_S S' \Rightarrow \theta(S) \subseteq \theta(S')$. Suppose $\mid z \mid \sqsubseteq_S \mid z' \mid$; then we have

$$z = \theta(\mid z \mid) \subseteq \theta(\mid z' \mid) = z'$$

which completes the proof that $\mid \cdot \mid$ is an order isomorphism. The same fact about θ can be deduced either in the same way or from the fact that it is the inverse of an order isomorphism. ∎

As in the last section, now that we have defined the powerdomain we should discover what we can about how to use it. We will again begin by defining the basic operations: singleton, union and intersection. However we cannot in general (unless $\mid \mathbf{A} \mid$ has a greatest element) expect intersection to be a total function on

$|\ \mathcal{P}_S(\mathbf{A})\ |^2$, simply because the intersection of two nonempty upwards-closed sets can be empty.

Definition
Suppose $x \in |\ \mathbf{A}\ |$ and $z_1, z_2 \in |\ \mathcal{P}_S(\mathbf{A})\ |$. Then we define

- $\{x\}_S = \{\, Y \in \wp(A) - \{\emptyset\} \mid Y \cap x \neq \emptyset\}$;
- $z_1 \underline{\cup}_S z_2 = z_1 \cap z_2$, and
- $z_1 \underline{\cap}_S z_2 = \overline{z_1 \cup z_2}$ if this is defined. ∎

Except for the restriction noted in the last section that we cannot expect to define an 'accurate' singleton set operation over upwards closed sets, these operations all work well. The proof of the following result, which is very similar to that of the corresponding result in the last section, is omitted.

6.3.4 LEMMA
The operations $\{\ \cdot\ \}_S$, $\underline{\cup}_S$ and $\underline{\cap}_S$ are all well-defined and continuous ($\underline{\cap}_S$ in its domain of definition). Furthermore

(a) $|\ \{x\}_S\ | = \{y \in |\ \mathbf{A}\ ||\ y \supseteq x\}$,

(b) $|\ z_1 \underline{\cup}_S z_2\ | = |\ z_1\ | \cup |\ z_2\ |$, and

(c) $|\ z_1 \underline{\cap}_S z_2\ | = |\ z_1\ | \cap |\ z_2\ |$. ∎

Given the topological analysis earlier in this section, we should not be too surprised that it is possible to define intersection and union operators successfully. For we discovered there that the finitely generable sets are closed under the operations of (finite) union and (arbitrary) intersection.

Topology sheds light on the feasibility of some of the other set operations one might want. Once again set difference makes no sense simply on the grounds that the difference of two upwards-closed sets need not be upwards closed; but we should also note that the difference of two finitely generable (i.e., closed) sets need not be closed either – so that even if we used a 'patch' for the upwards closure problem, we would still have to use the fg-closure operator.

Remember that in the other construction discussed earlier, $f(S) = \{f(x) \mid x \in S\}$ (with $f :|\ \mathbf{A}\ | \to |\ \mathbf{B}\ |$ continuous), we had to use the fg-closure operation. To a topologist this at first seems surprising, for it is a well-known theorem of topology that the continuous image in a Hausdorff space of a closed subset of a compact Hausdorff space is closed. But this is not really a paradox because there are two different definitions of 'continuous' at work here: f is assumed to be continuous in the partial order sense defined in Chapter 2, while the notion of continuity needed

for the theorem is topological continuity. A function f from one topological space X to another Y is said to be *continuous* if, whenever U is open in Y, then $f^{-1}(U)$ is open in X. (This is equivalent to saying that whenever C is closed in Y, then $f^{-1}(C)$ is closed in X.) Recall that in Section 2.4 we identified a topology (the Scott topology) which characterised partial order continuity, and that was certainly not the topology currently under examination.

The subject of topologies on partial orders is a large one and we do not have space to discuss them in detail here. Their properties are, however, described in great detail in [Compendium], where the topology of finitely generable sets is called the *Lawson* topology (see Exercise 6.3.5 below). We can, however, state two criteria for functions to be continuous with respect to $\mathcal{T}_{FG}$. The proof is left as an exercise.

6.3.5 LEMMA

(a) A function $f :| \mathbf{A} |\!\rightarrow\!| \mathbf{B} |$ is continuous with respect to $\mathcal{T}_{FG}$ if and only if, for all $x \in| \mathbf{A} |$ and $b \in B$ there is $Z \subseteq^{\text{fin}} A$ such that

$$x \cap Z = x' \cap Z \Rightarrow (b \in f(x) \Leftrightarrow b \in f(x')) \,.$$

(b) The distributive[2] function $f :| \mathbf{A} |\!\rightarrow\!| \mathbf{B} |$ is continuous with respect to $\mathcal{T}_{FG}$ if and only if it is partial order continuous and *infinitely distributive* (i.e., $f(\bigcap S) = \bigcap\{f(x) \mid x \in S\}$ for all nonempty $S \subseteq| \mathbf{A} |$). ∎

We now know that any function which satisfies either of the above can reliably be used to define $\underline{f}_H$ without the need for fg-closure. (It is trivial to show that if S is finitely generable then so are $S{\downarrow}$, and indeed $S{\uparrow}$ and $ccl(S)$.) Curiously, this analysis is *not* required for the Smyth powerdomain: if $f :| \mathbf{A} |\!\rightarrow\!| \mathbf{B} |$ is (partial order) continuous and $z \in| \mathcal{P}_S(\mathbf{A}) |$ then $f(| z |)$ need not be finitely generable, but $f(| z |){\uparrow}$ always is.

6.3.6 LEMMA

Suppose $f :| \mathbf{A} |\!\rightarrow\!| \mathbf{B} |$ is (partial order) continuous and $S \subseteq| \mathbf{A} |$ is finitely generable. Then $f(S){\uparrow}$ is finitely generable.

PROOF

If not then there would be $y \notin f(S){\uparrow}$ such that for all $Z \subseteq^{\text{fin}} B$ there is $x \in S$ with $f(x) \cap Z \subseteq y \cap Z$. Hence $S_Z = \{x \in S \mid f(x) \cap Z \subseteq y\}$ is always nonempty. It is also closed, since it is the intersection of S and the set $\{x \in| \mathbf{A} || f(x) \cap Z \subseteq y\}$

[2]Recall that the function $f :| \mathbf{A} |\!\rightarrow\!| \mathbf{B} |$ is distributive if and only if $f(x \cap y) = f(x) \cap f(y)$ for all x, y. In other words it maps the greatest lower bounds of two elements of an information system to the greatest lower bound of their images.

which is closed since, if x is not a member then, as f is continuous, there is a finite subset X of x such that $f(\overline{X}) \cap Z \not\subseteq y$ (and hence $x' \cap X = x \cap X \Rightarrow f(x') \cap Z \not\subseteq y$). Using compactness and the fact that $S_{Z \cup Z'} = S_Z \cap S_{Z'}$ we can deduce that

$$\emptyset \neq \bigcap\{S_Z \mid Z \subseteq^{\text{fin}} B\} = \{x \in S \mid f(x) \subseteq y\}$$

which contradicts the assumption that $y \notin f(S)\uparrow$. ∎

This means we can now make the following definition with some confidence.

Definition
If $f :| \mathbf{A} |\to| \mathbf{B} |$ is a continuous function then we define $\underline{f}_S :| \mathcal{P}_S(\mathbf{A}) |\to| \mathcal{P}_S(\mathbf{B}) |$ by

$$\underline{f}_S(z) = \{\, Y \in \wp(B) - \{\emptyset\} \mid \forall\, x \in| z |\, .\, Y \cap f(x) \neq \emptyset\}.$$ ∎

Given Lemma 6.3.6, the following result is quite straightforward, except perhaps for the continuity part, which requires another compactness argument very similar to that in the proof of Lemma 6.3.6.

6.3.7 LEMMA
If $f :| \mathbf{A} |\to| \mathbf{B} |$ is continuous, then $\underline{f}_S$ is well-defined, continuous, and furthermore $|\, \underline{f}_S(z) \,| = f(|\, z \,|)\uparrow$. ∎

Exercises

6.3.1 Compute the structure of the domain $\mathcal{P}_S(\mathbf{flat}(X))$. (Recall your answer to Exercise 6.2.1.)

6.3.2 (a) If $|\, \mathbf{A} \,|$ ordered by $\subseteq$ is order isomorphic to the same set ordered by $\supseteq$ (note that $|\, \mathbf{A} \,|$ necessarily has a greatest element), need $\mathcal{P}_S(\mathbf{A}) \cong \mathcal{P}_H(\mathbf{A})$ (i) in the case where $|\, \mathbf{A} \,|$ is finite, and (ii) in general?
[Hint: What would happen to your answers to 6.2.1 and 6.3.1 if you were to add a top element to the domain?]

*(b) Explain the answer to (a).

6.3.3 Explain why $|\, \mathcal{P}_S(\mathbf{A}) \,|$ is a complete partial order even though $\mathcal{Q}_S(|\, \mathbf{A} \,|)$ need not be (see Exercise 6.1.3).

6.3.4 Recall the domain constructor *over* from Exercise 4.1.4. Show that $\mathcal{P}_S(\mathbf{A} \mathit{over} \mathbf{B}) \cong \mathcal{P}_S(\mathbf{A})$ *over* $\mathcal{P}_S(\mathbf{B})$.

*6.3.5 Show that, if $\chi \in| \mathcal{P}_S(\mathcal{P}_S(\mathbf{A})) |$ then $\bigcup\{|\, z \,||\, z \in| \chi \,|\}$ is finitely generable. (Notice that this says more than is deducible from the fact that the

finitely generable sets form a topology.) Hence define an operator $\underline{\bigcup}_S$:| $\mathcal{P}_S(\mathcal{P}_S(\mathbf{A}))$ |→| $\mathcal{P}_S(\mathbf{A})$ | such that

$$|\,\underline{\bigcup}_S(\chi)\,| = \bigcup\{|\,z\,|\,|\,z \in |\,\chi\,|\}\,.$$

Is it continuous?
[Hint: if Λ is a finitely generable subset of | $\mathcal{P}_S(\mathbf{A})$ |, show that whenever $x \in |\,\mathbf{A}\,|$ and $Z \subseteq^{\text{fin}} A$ then

$$\Lambda_Z = \{z \in \Lambda \mid \exists\, x' \in |\,z\,|\,.x' \cap Z = x \cap Z\}$$

is closed, and apply compactness.]

*6.3.6 (a) Recall that a set C is closed in the Scott topology if and only if, whenever $\Delta \subseteq C$ is directed, then $\{x \mid x \subseteq \bigcup \Delta\} \subseteq C$. Prove that every Scott-closed set is finitely generable.

(b) The *lower* topology on a partial order P is defined to be the smallest one where all sets of the form $\{x\}\uparrow$ (usually abbreviated $x\uparrow$) are closed. (A closed-set basis is given by $\{P\} \cup \{x_1\uparrow \cup \ldots \cup x_n\uparrow \mid n \in \mathbb{N} \wedge \forall\, i.x_i \in P\}$.) Prove that every set closed in the lower topology is finitely generable.

(c) The *Lawson* topology is the smallest topology which contains both the Scott topology and the lower topology. (Namely, its closed sets are those derivable from those of the other two via finite unions and arbitrary intersections.) Show that, if $S \subseteq |\,\mathbf{A}\,|$ is finitely generable and $x \notin S$ then there are a Scott-closed set C_1 and a lower-closed set C_2 such that $S \subseteq C_1 \cup C_2$ and $x \notin C_1 \cup C_2$. Deduce that the topology of finitely generable sets is precisely the Lawson topology.

6.4 The reverse inclusion powerdomain

It is possible to define a powerdomain constructor $\mathcal{P}_{fg}(\cdot)$ whose effect is to produce the domain of all finitely generable subsets of a given domain, ordered by reverse inclusion ($\supseteq$). As with the Smyth and Hoare powerdomains, one can do this either in a way which allows the empty set or specifically excludes it. As discussed earlier it is often useful to have the latter. The nonempty-sets-only case is a little harder and more interesting to analyse, so we will concentrate on it. We will reserve the notation $\mathcal{P}_{fg}(\mathbf{A})$ for the version without $\emptyset$, and use $\mathcal{P}^{\emptyset}_{fg}(\mathbf{A})$ for the version that includes $\emptyset$.

Let us examine the definition of finitely generable sets more closely. It says that $x \in |\,\mathbf{A}\,|$ belongs to the set S if we cannot tell finitely that this is impossible. A finite piece of information that x is not in S looks like the pair $\langle Z, Z \cap x\rangle$ for some $Z \subseteq^{\text{fin}} A$. A pair $\langle Z, Z \cap x\rangle$ such that $x \notin S$ if $Z \cap x' = Z \cap x \Rightarrow x' \notin S$ can be

thought of a 'certificate' that $x \notin S$ – and the definition of finitely generable says that some certificate must exist for every $x \notin S$. It therefore seems promising to try to represent the set S by the set of certificates which witness non-membership of S. We will base the token set of $\mathcal{P}_{fg}(\mathbf{A})$ on this idea.

The token set of $\mathcal{P}_{fg}(\mathbf{A})$ will consist of pairs $\langle X, Y\rangle$, where $X,\ Y \subseteq^{\text{fin}} A$. The pair $\langle X, Y\rangle$ will mean that all elements x of our hypothetical set S *either* have $X \not\subseteq x$, *or* $Y \cap x \neq \emptyset$. This asserts that, for no $x \in S$ do we have $(X \cup Y) \cap x = X$, and thus says that $\langle X \cup Y, X\rangle$ is a 'certificate' for non-membership of S of the type discussed in the last paragraph. The reason for this slightly different presentation is to ease later definitions.

There are many pairs $\langle X, Y\rangle$ which are better left out of the token set. Certainly there is no point in including tokens which will be true of every set S. We need two restrictions to remove these: first we only allow $\langle X, Y\rangle$ where $X \in Con_A$ (any other set cannot be a subset of x for $x \in\mid \mathbf{A} \mid$, which is necessary if x is to fail the restriction imposed by $\langle X, Y\rangle$). Second, we require $\overline{X} \cap Y = \emptyset$ – for otherwise there can be no x such that $X \subseteq x \ \wedge\ x \cap Y = \emptyset$. Note, however, that if $\langle X, Y\rangle$ does satisfy both these restrictions then there is an x, namely $\overline{X}$, which is excluded by the token. If we choose to remove the empty set from our powerdomain then we can allow no token $\langle X, Y\rangle$ which is naturally only true of it – because $\langle X, Y\rangle$ excludes *all* x from a set. If $Y \neq \emptyset$ this cannot occur – for whenever $a \in Y$ the element $\overline{\{a\}}$ passes the test represented by $\langle X, Y\rangle$. Similarly if $X \not\subseteq \overline{\emptyset}_A$ then $\overline{\emptyset}_A$ passes the test. However if neither of these apply, namely $X \subseteq \overline{\emptyset}_A$ and $Y = \emptyset$ then necessarily $X \subseteq x \wedge x \cap Y = \emptyset$ for all $x \in\mid \mathbf{A} \mid$, and the token $\langle X, Y\rangle$ excludes all nonempty sets. Thus, to satisfy Axiom 2 of information systems, we cannot allow tokens of this form in the powerdomain of nonempty sets.

Definition
The token set B of $\mathcal{P}_{fg}(\mathbf{A})$ is defined

$$B = \{\langle X, Y\rangle \mid X \in Con_A \wedge Y \subseteq^{\text{fin}} A - \overline{X} \wedge (X \not\subseteq \overline{\emptyset}_A \vee Y \neq \emptyset)\}\,. \qquad \blacksquare$$

The corresponding definition for $\mathcal{P}^{\emptyset}_{fg}(\mathbf{A})$ is the same except that the last clause $X \not\subseteq \overline{\emptyset}_A \vee Y \neq \emptyset$ is omitted.

Definition
The token set $B^{\emptyset}$ of $\mathcal{P}^{\emptyset}_{fg}(\mathbf{A})$ is defined

$$B = \{\langle X, Y\rangle \mid X \in Con_A \wedge Y \subseteq^{\text{fin}} A - \overline{X}\}\,. \qquad \blacksquare$$

Note that the pair $\langle \emptyset, \emptyset\rangle$ is always token of $\mathcal{P}^{\emptyset}_{fg}(\mathbf{A})$ but never of $\mathcal{P}_{fg}(\mathbf{A})$.

If we wanted to allow the empty set into the powerdomain then the consistency relation would be trivial, for every token would be true of the empty

set. (All of $\emptyset$'s members satisfy the constraints imposed by every possible token!) The situation is not so simple when we exclude it, for then a finite set $\mathcal{X} = \{\langle X_i, Y_i\rangle \mid 1 \leq i \leq n\}$ of tokens will be consistent only if there is some $x \in |\ \mathbf{A}\ |$ that none of them excludes. For such an x, we know that for all i we *either* have $x \cap Y_i \neq \emptyset$ *or* $X_i \not\subseteq x$. Let Λ be the set of all $i \in \{1 \ldots n\}$ where the first of these apply. For each $i \in \Lambda$ choose $a_i \in Y_i \cap x$. The set $X = \{a_i \mid i \in \Lambda\}$ is consistent (for it is a subset of x) and $\overline{X} \subseteq x$. Therefore whenever $i \notin \Lambda$ we have $\overline{X} \not\supseteq X_i$. It follows that $\mathcal{X}$ is consistent in the sense defined below.

Definition
The consistency relation Con of $\mathcal{P}_{fg}(\mathbf{A})$ is defined as follows: $\mathcal{X} = \{\langle X_i, Y_i\rangle \mid 1 \leq i \leq n\} \in Con$ if and only if there is a subset Λ of $\{1 \ldots n\}$ and $a_i \in Y_i$ for $i \in \Lambda$ such that $X = \{a_i \mid i \in \Lambda\} \in Con_A$ and $X_j \not\subseteq \overline{X}$ when $j \notin \Lambda$. ∎

If $\mathcal{X} \in Con$ then there is an element of $|\ \mathbf{A}\ |$ not excluded, namely $\overline{X}$ where $X = \{a_i \mid i \in \Lambda\}$ as in the Definition above. Thus the consistency relation Con coincides precisely with our expectation that a $\mathcal{X}$ should be consistent if any only if there is some element of $|\ \mathbf{A}\ |$ that it does not exclude.

Given an element $\mathcal{X} = \{\langle X_i, Y_i\rangle \mid 1 \leq i \leq n\}$ of Con and $\langle X, Y\rangle \in B$, we would expect that $\mathcal{X} \vdash \langle X, Y\rangle$ if and only if there is no $x \in |\ \mathbf{A}\ |$ excluded by $\langle X, Y\rangle$ that is not excluded by some element of $\mathcal{X}$. In other words, all information conveyed by $\langle X, Y\rangle$ about a set S is already contained in $\mathcal{X}$. If such an x existed then we would have $X \subseteq x$, $x \cap Y = \emptyset$, and as in the discussion above there would be a subset Λ of $\{1 \ldots n\}$ such that $x \cap Y_i \neq \emptyset$ for $i \in \Lambda$ and $X_i \not\subseteq x$ for $i \notin \Lambda$. Again pick $a_i \in x \cap Y_i$ for each $i \in \Lambda$. Defining $X^* = X \cup \{a_i \mid i \in \Lambda\}$ we clearly have

(a) $\overline{X^*} \subseteq x$ and hence $X_i \not\subseteq \overline{X^*}$ for $i \notin \Lambda$;

(b) $\overline{X^*} \cap Y_i \neq \emptyset$ for $i \in \Lambda$;

(c) $X \subseteq \overline{X^*}$ and $\overline{X^*} \cap Y = \emptyset$.

It follows that $\mathcal{X} \nvdash \langle X, Y\rangle$ when $\vdash$ is defined as below.

Definition
The entailment relation of $\mathcal{P}_{fg}(\mathbf{A})$ is defined as follows. If $\mathcal{X} = \{\langle X_i, Y_i\rangle \mid 1 \leq i \leq n\} \in Con$ and $\langle X, Y\rangle \in B$, then $\mathcal{X} \vdash \langle X, Y\rangle$ if and only if *whenever* $\Lambda \subseteq \{1 \ldots n\}$ and $a_i \in Y_i$ for $i \in \Lambda$ are such that $X^* = X \cup \{a_i \mid i \in \Lambda\} \in Con_A$ then *either* $\overline{X^*} \cap Y \neq \emptyset$ *or* there is some $j \notin \Lambda$ such that $X_j \subseteq \overline{X^*}$. ∎

If $\mathcal{X}$ is as above and $\mathcal{X} \nvdash \langle X, Y\rangle$, then the definition above generates a Λ and X^* which violate the constraints. Clearly $\overline{X^*}$ is an element of $|\ \mathbf{A}\ |$ which is not excluded by any element of $\mathcal{X}$ but is excluded by $\langle X, Y\rangle$. Thus we would not

naturally expect to have $\mathcal{X}$ entail $\langle X, Y\rangle$ since $\mathcal{X}$ would be 'true' of the set $\{\overline{X^*}\}$ while $\langle X, Y\rangle$ is not. These remarks show that our definition of $\vdash$ means exactly what was intended.

The entailment relation of $\mathcal{P}^{\emptyset}_{fg}(\mathbf{A})$ has exactly the same definition (with the larger sets Con and B. For note that a set $\mathcal{X}$ that excludes every x necessarily entails every pair $\langle X, Y\rangle$.

Having made these definitions we have an obligation to show that they are reasonable: namely that the constructors $\mathcal{P}_{fg}(\cdot)$ and $\mathcal{P}^{\emptyset}_{fg}(\cdot)$ are well-defined, continuous and have the expected sets of elements.

The analysis of this powerdomain is at least as complex as that of the Smyth powerdomain earlier. We illustrated there how one could use topology to prove the trickiest parts of necessary arguments. As was stated earlier, in this section we let logic play the same role, and in fact we can give it a slightly wider one because of the idea that tokens are propositions about elements, so an information system is itself logically based. The tokens of powerdomains are often assertions about the elements of their hypothetical sets S, and it is helpful to have a framework that allows us to integrate these assertions with logic.

Suppose $\mathbf{A} = \langle A, Con_A, \vdash_A\rangle$ is an information system. We can create a propositional language whose statement letters are in correspondence with the tokens A: $\{p_a \mid a \in A\}$ and whose connectives are the usual ones of the propositional calculus: $\{\vee, \wedge, \neg, \Rightarrow\}$. Intuitively we will think of formulae in this language as making statments about a subset x of A – p_a being true will correspond to $a \in x$, while p_a being false will mean $a \notin x$. The consistency and entailment relations of $\mathbf{A}$ can themselves be turned into a set of formulae, which will allow us to restrict x to being an element of $|\ \mathbf{A}\ |$.

Definition
If $\mathbf{A} = \langle A, Con_A, \vdash_A\rangle$ is as above, then we define $\Gamma_{\mathrm{A}} = \Gamma_1 \cup \Gamma_2$, where

- $\Gamma_1 = \{\neg(p_{a_1} \wedge p_{a_2} \wedge \ldots \wedge p_{a_k}) \mid \{a_1, a_2, \ldots, a_k\} \notin Con_A\}$ and
- $\Gamma_2 = \{(p_{a_1} \wedge p_{a_2} \wedge \ldots \wedge p_{a_k}) \Rightarrow p_b \mid \{a_1, a_2, \ldots, a_k\} \vdash_A b\}$. ∎

The set Γ_{A} simply says that $x \in |\ \mathbf{A}\ |$. For any x satisfying it must be consistent – for any inconsistent finite subsets would violate an element of Γ_1 – and deductively closed – for if $X \subseteq^{\mathrm{fin}} x$ and $X \vdash_A b$ then $b \in x$ by the appropriate member of Γ_2.

If $\langle X, Y\rangle \in B$ (the token set of $\mathcal{P}_{fg}(\mathbf{A})$) where $X = \{a_1, \ldots, a_n\}$ and $Y = \{b_1, \ldots, b_m\}$ then we can easily translate it into the logical proposition it represents, namely

$$\phi_{\langle X,Y\rangle} = \neg p_{a_1} \vee \ldots \vee \neg p_{a_k} \vee p_{b_1} \vee \ldots \vee p_{b_k}\ .$$

For an element x is only excluded by $\langle X, Y\rangle$ if every p_{a_i} is true (i.e., $X \subseteq x$) and every p_{b_j} is false (i.e., $Y \cap x = \emptyset$). The set of x allowed by some subset z of B thus becomes identified with (the truth assignments satisfying) $\Sigma_z = \Gamma_{\mathrm{A}} \cup \{\phi_{\langle X,Y\rangle} \mid \langle X, Y\rangle \in z\}$. For the subsets x of A satisfying Σ_z are all elements of $\mid \mathbf{A} \mid$ which violate none of the constraints imposed by z.

The definitions of the consistency and entailment relations above were accompanied by rather detailed discussions. The first of these showed that $\mathcal{X} \in Con$ if and only if there is some element of $\mid \mathbf{A} \mid$ simultaneously satisfying all the constraints of $\mathcal{X}$. But this is precisely equivalent to the statement that the set $\Sigma_{\mathcal{X}}$ is satisfiable, which is equivalent to saying that it is logically consistent. In the notation of logic we can re-write this as $\Sigma_{\mathcal{X}} \not\models False$.[3]

The second discussion showed that $\mathcal{X} \vdash \langle X, Y\rangle$ if and only if there is no element of $\mid \mathbf{A} \mid$ that satisfies the constraints of $\mathcal{X}$ but is excluded by $\langle X, Y\rangle$. This is precisely equivalent to the logical statement $\Sigma_{\mathcal{X}} \models \phi_{\langle X,Y\rangle}$.

We have thus characterised both the consistency and entailment relations of the system $\mathcal{P}_{fg}(\mathbf{A})$ in terms of the corresponding notions from propositional logic. We will find this invaluable, and the first application is the following.

6.4.1 PROPOSITION
The domain constructor $\mathcal{P}_{fg}(\cdot)$ is well-defined and continuous.

PROOF
It is well-defined if $\mathcal{P}_{fg}(\mathbf{A})$ is an information system when $\mathbf{A}$ is. The token set B was carefully constructed to satisfy Axiom 2, since all tokens that would have excluded all x were discarded. All of the others follow very straightforwardly from the characterisation above. Consider Axiom 5, for example. Suppose that $\mathcal{X} \vdash \mathcal{Y}$ and $\mathcal{Y} \vdash \langle X, Y\rangle$. Then in terms of the logical representation this means that

$$\Sigma_{\mathcal{X}} \models \phi_{\langle X',Y'\rangle} \qquad \text{for all } \langle X', Y'\rangle \in \mathcal{Y}.$$

and that $\Sigma_{\mathcal{Y}} \models \phi_{\langle X,Y\rangle}$. But it follows easily from the first of these that $\Sigma_{\mathcal{X}} \models \Sigma_{\mathcal{Y}}$, and hence that $\Sigma_{\mathcal{X}} \models \phi_{\langle X,Y\rangle}$ (by properties of $\models$). This is exactly what is required.

The proof of continuity (via monotonicity and continuity in token sets) is a fairly standard calculation. There are two more interesting parts; one is the observation that, if $\mathbf{A} \trianglelefteq \mathbf{A}'$, then $B \subseteq B'$ because exactly the same pairs $\langle X, Y\rangle$ with $X, Y \subseteq^{\mathrm{fin}} A$ appear in both. The other is the fact that continuity in token sets depends vitally on the fact that the tokens $\langle X, Y\rangle$ are composed of *finite* X

[3] If $\Gamma \cup \{\phi\}$ is a set of propositional formulae then the notation $\Gamma \models \phi$ means that there is no assignment to the variables of the language which satisfies all of Γ but not ϕ. Since $False$ is satisfied by no assignment, the statement $\Gamma \models False$ is equivalent to Γ being unsatisfiable.

and Y: it is this and the usual directed set argument that ensures each token of $\mathcal{P}_{fg}(\bigsqcup \Delta)$ (Δ directed) appears in some $\mathcal{P}_{fg}(\mathbf{A})$ for $\mathbf{A} \in \Delta$. ∎

The corresponding result holds for $\mathcal{P}^{\emptyset}_{fg}(\cdot)$ with a similar proof. We should note that, in $\mathcal{P}^{\emptyset}_{fg}(\mathbf{A})$, the consistency relation is trivial and therefore no longer depends on the consistency of $\Sigma_{\mathcal{X}}$. However the entailment relation is still modelled by $\models$ in precisely the same way as in $\mathcal{P}_{fg}(\mathbf{A})$.

The next step in this development must be to verify that $\mid \mathcal{P}^{\emptyset}_{fg}(\mathbf{A}) \mid$ and $\mid \mathcal{P}_{fg}(\mathbf{A}) \mid$ correspond to the finitely generable, and nonempty finitely generable, subsets of $\mid \mathbf{A} \mid$. We already have a good idea of what the correspondences will look like.

Definition
Suppose $z \in \mid \mathcal{P}_{fg}(\mathbf{A}) \mid$ and S is a nonempty finitely generable subset of $\mid \mathbf{A} \mid$. Then, if B is the token set of $\mathcal{P}_{fg}(\mathbf{A})$ as above, we define

- $\mid z \mid = \{x \in \mid \mathbf{A} \mid\mid \forall \langle X, Y\rangle \in z. x \cap (X \cup Y) \neq X\}$ and
- $\theta(S) = \{\langle X, Y\rangle \in B \mid \forall x \in S. x \cap (X \cup Y) \neq X\}$.

Exactly the same clauses apply to $\mathcal{P}^{\emptyset}_{fg}(\mathbf{A})$, dropping the restriction that S is nonempty and replacing B by $B^{\emptyset}$. ∎

The following result proves (for $\mathcal{P}_{fg}(\cdot)$) that this correspondence has exactly the properties we desire. Of course an analogous result holds for $\mathcal{P}^{\emptyset}_{fg}(\cdot)$.

6.4.2 THEOREM
The operations $\mid \cdot \mid$ and $\theta(\cdot)$ defined above for $\mathcal{P}_{fg}(\cdot)$ are well-defined, and furthermore

(a) $\mid \theta(S) \mid = S$ for all finitely generable nonempty $S \subseteq \mid \mathbf{A} \mid$,

(b) $\theta(\mid z \mid) = z$ for all $z \in \mid \mathcal{P}_{fg}(\mathbf{A}) \mid$ and

(c) $\mid \mathcal{P}_{fg}(\mathbf{A}) \mid$ is order isomorphic to the set of all finitely generable nonempty subsets of $\mid \mathbf{A} \mid$ ordered by reverse inclusion ($\supseteq$).

PROOF
The well-definedness of $\mid \cdot \mid$ requires two things demonstrated. Firstly we must show that $\mid z \mid$ is finitely generable, and secondly we must prove that it is nonempty. The first of these comes more or less by definition: suppose that $x \notin \mid z \mid$. Then, by definition of $\mid \cdot \mid$ there must exist $\langle X, Y\rangle \in z$ such that $x \cap (X \cup Y) = X$. But clearly then, if x' is any other element of $\mid \mathbf{A} \mid$ and $x' \cap (X \cup Y) = x \cap (X \cup Y)$ then $x' \notin \mid z \mid$. It follows immediately that $\mid z \mid$ is well-defined.

The proof that $\mid z \mid$ is nonempty relies on our logical reformulation of the system above. For we know that every finite subset $\mathcal{X}$ of z belongs to Con. It follows that $\Sigma_{\mathcal{X}}$ is satisfiable for all such $\mathcal{X}$. It follows that the set Σ_z is finitely satisfiable (i.e., every finite subset is satisfiable); but by the compactness theorem for the propositional calculus[4] this implies that Σ_z is satisfiable. It is easy to see that the set of all a such that p_a is true in the satisfying assignment belongs to $\mid z \mid$. This proves that $\mid z \mid$ is indeed nonempty.

To prove θ well-defined we must prove that $\theta(S)$ is always consistent and deductively closed. Consistency follows directly from the fact that S is nonempty: we know by definition that when $x \in S$ then $x \cap (X \cup Y) \neq X$ for all $\langle X, Y\rangle \in \theta(S)$. It follows that $\Sigma_{\theta(S)}$ is satisfied by the truth assignment corresponding to x. Thus, in particular, every $\Sigma_{\mathcal{X}}$ for $\mathcal{X} \subseteq^{\text{fin}} \theta(X)$ is satisfiable, so $\mathcal{X} \in Con$. For deductive closure, suppose $\mathcal{X} \subseteq^{\text{fin}} \theta(S)$ and $\mathcal{X} \vdash \langle X, Y\rangle$. If $\langle X, Y\rangle \notin \theta(S)$ then there would be some $x \in S$ such that $x \cap (X \cup Y) = X$. But we know that, since $\mathcal{X} \vdash \langle X, Y\rangle$, there is no element of $\mid \mathbf{A} \mid$ excluded by x that is not excluded by some element of $\mathcal{X}$. It follows that there is some $\langle X', Y'\rangle \in \mathcal{X}$ such that $x \cap (X' \cup Y') = X'$, in contradiction to our assumption that $\mathcal{X} \subseteq^{\text{fin}} \theta(S)$.

We can thus turn to the proof of (a). It follows directly from the definitions that, for all S, $S \subseteq \mid \theta(S) \mid$. So suppose $x \notin S$. Since S is assumed to be finitely generable there must be a set $Z \subseteq^{\text{fin}} A$ such that $x \cap Z = x' \cap Z \Rightarrow x' \notin S$. Thus the token $\langle x \cap Z, Z - x\rangle$ belongs to $\theta(S)$. But, by construction, x does not satisfy $\langle x \cap Z, Z - x\rangle$, which means $x \notin \mid \theta(S) \mid$. This proves $S \supseteq \mid \theta(S) \mid$, which completes the proof of (a).

Part (b) is proved if we can show that the operation $\mid \cdot \mid$ is injective. This is because we know that $\mid z \mid = \mid \theta(\mid z \mid) \mid$ by part (a), which then implies $z = \theta(\mid z \mid)$. So suppose for contradiction that z_1 and z_2 were two different elements of $\mid \mathcal{P}_{fg}(\mathbf{A}) \mid$ such that $\mid z_1 \mid = \mid z_2 \mid$. Since, in general, the elements of $\mid z \mid$ are in direct correspondence with the set of truth assignments satisfying Σ_z, it follows that the sets of truth assignments satisfying Σ_{z_1} and Σ_{z_2} are the same. In other words $\Sigma_{z_1} \models \Sigma_{z_2}$ and $\Sigma_{z_2} \models \Sigma_{z_1}$. Without loss of generality we can assume there is some $\langle X, Y\rangle \in z_1 - z_2$. Now we know that $\Sigma_{z_2} \models \phi_{\langle X, Y\rangle}$, and it follows by the compactness theorem that there is some finite subset Γ of Σ_{z_2} such that $\Gamma \models \langle X, Y\rangle$.[5] But if we define $\mathcal{X} = \{\langle X', Y'\rangle \mid \phi_{X', Y'} \in \Gamma\}$, we then have $\mathcal{X} \subseteq^{\text{fin}} z_2$ and $\mathcal{X} \vdash \langle X, Y\rangle$. This contradicts our assumption that $\langle X, Y\rangle \notin z_2$. Hence $z_1 = z_2$, completing the proof of part (b).

Parts (a) and (b) show that $\mid \cdot \mid$ and θ are both bijections and are each

[4]The compactness theorem for the propositional calculus states that any set of propositional formulae which is finitely satisfiable is satisfiable. This consequence of Zorn's Lemma is proved in most textbooks on mathematical logic.

[5]If $\Sigma \models \psi$ then $\Sigma \cup \{\neg\psi\}$ is unsatisfiable. Therefore some finite $\Gamma \subseteq \Sigma$ has $\Gamma \cup \{\neg\psi\}$ unsatisfiable, which means $\Gamma \models \psi$.

other's inverses. We will show that they are order isomorphisms. It is clear from their definitions that $z \subseteq z' \Rightarrow | z | \sqsupseteq | z' |$ and that $S \supseteq S' \Rightarrow \theta(S) \subseteq \theta(S')$. Suppose $| z | \sqsupseteq | z' |$; then we have

$$z = \theta(| z |) \subseteq \theta(| z' |) = z'$$

which completes the proof that $| \cdot |$ is an order isomorphism. The same fact about θ can be deduced either in the same way or from the fact that it is the inverse of an order isomorphism. ∎

Perhaps the most curious feature of this powerdomain is that its partial order is not derived in any way from the order on the underlying domain. We will discuss this fact and its consequences in some detail below.

The basic set operations are generally better behaved in this powerdomain than in the other ones: singleton set, intersection and pairwise union all mean exactly what one would expect. (For the set of finitely generable subsets of $| \mathbf{A} |$ contains all singleton sets and is closed under finite union and arbitrary intersection.) Naturally in $\mathcal{P}_{fg}(\mathbf{A})$ (as opposed to $\mathcal{P}^{\emptyset}_{fg}(\mathbf{A})$), intersection is a partial function, since the intersection of two nonempty sets can be empty. Of course the intersection and union operations are just the least upper bound and greatest lower bound operations on the domain.

Definition
Suppose that $x \in | \mathbf{A} |$ and $z_1, z_2 \in | \mathcal{P}_{fg}(\mathbf{A}) |$. Then we define:

- $\{x\}_{fg} = \theta(\{x\})$;
- $z_1 \underline{\cup}_{fg} z_2 = z_1 \cap z_2$ and
- $z_1 \underline{\cap}_{fg} z_2 = \overline{z_1 \cup z_2}$ (provided $| z_1 | \cap | z_2 | \neq \emptyset$).

6.4.3 LEMMA
The operators $\underline{\cup}_{fg}$ and $\underline{\cap}_{fg}$ are well-defined and continuous (the latter on its domain of definition). Furthermore, if $z_1, z_2 \in | \mathcal{P}_{fg}(\mathbf{A}) |$ then

(a) $| z_1 \underline{\cup}_{fg} z_2 | = | z_1 | \cup | z_2 |$, and

(b) $| z_1 \underline{\cap}_{fg} z_2 | = | z_1 | \cap | z_2 |$. ∎

The most obvious *advantage* of this powerdomain is the fact that it includes all finitely generable sets. Unlike in the classical powerdomains, there is no upwards-closed, downwards-closed or convex-closed condition to worry about. This will be a positive advantage if there is no good reason for expecting any of them for other

reasons. Also the partial order used, reverse inclusion, is both naturally occurring and has the natural interpretation that stronger means more deterministic.

The singleton sets $\{x\}_{fg}$ and $\{y\}_{fg}$ are always incomparable (in fact, maximal in $\mid \mathcal{P}_{fg}(\mathbf{A}) \mid$) even if $x \subseteq y$. This means that the singleton set operation is not one which can be used in a context where a monotone or continuous operation is required in order to obtain a fixed point. This is perhaps the most obvious *disadvantage* of this powerdomain, since in contexts where powerdomains are used reflexively (i.e., in domain equations) it is often natural to use the singleton set operation in exactly such contexts. The best that can be done is to use an approximation to the singleton set operator. The most obvious one is the one used in the Smyth powerdomain, namely $\{y \mid x \subseteq y\}$, though in contexts where we know the image of the continuous function f is contained in some finitely generable set S smaller than $\mid \mathbf{A} \mid$ it may be preferable to use the approximation $\{y \in S \mid f(x) \subseteq y\}$ to $\{f(x)\}$.

The result of using these approximations in one's semantics will be that one gets semantic values that are approximations to the real answers (in some sense of the word 'real'). For example, if the final answer is in the powerdomain, it could be a larger set. However it seems clear that one would always obtain a result which was at least as good (and very possibly better) than would be obtained by using the Smyth powerdomain itself.

Quite independently of the above, it could be viewed as a disadvantage that the partial order on $\mid \mathcal{P}_{fg}(\mathbf{A}) \mid$ owes nothing to that of $\mid \mathbf{A} \mid$. (Though equally, if the order on $\mid \mathbf{A} \mid$ is not of great importance it may be better to use the determinism order provided by $\mid \mathcal{P}_{fg}(\mathbf{A}) \mid$.) Note that the same is true of the left-hand side of the function space operator – like $\mathcal{P}_{fg}(\cdot)$ (see the appendix at the end of this chapter) the *membership* of $\mid \mathbf{A} \to \mathbf{B} \mid$ depends on the order on $\mid \mathbf{A} \mid$, but the *order* does not.

The direct image $f(S)$ of a set under a function works well over this powerdomain. It is better to consider this aspect of our analysis using the compact topology $\mathcal{T}_{FG}$ discussed in the last section. Provided f is continuous with respect to the topology then $f(S)$ is bound to be finitely generable if S is. And since the order is now just reverse inclusion it is clear that $S \supseteq S' \Rightarrow f(S) \supseteq f(S')$, or in other words f is monotone on sets. It is also continuous, for using the fact that finitely generable sets are closed under arbitrary intersection tells us that if Δ is a directed set of finitely generable sets then $\bigcap \Delta$ is also finitely generable, and clearly

$$f(\bigcap \Delta) \subseteq \bigcap \{f(S) \mid S \in \Delta\} .$$

And if y were an element of the right-hand side above then the closed sets $\{x \in S \mid f(x) = y\}$ would have the finite intersection property as S varies, proving that y is in the left-hand side.

As before we can define this lifted version of f as an operator on powerdomains.

Definition
Suppose $f :| \mathbf{A} |\to| \mathbf{B} |$ is continuous with respect to the topology $\mathcal{T}_{FG}$, then we define $\underline{f}_{fg} :| \mathcal{P}_{fg}(\mathbf{A}) |\to| \mathcal{P}_{fg}(\mathbf{B}) |$ as follows

$$\underline{f}_{fg}(z) = \{\langle X, Y\rangle \mid \forall\, x \in| z \mid .f(x) \cap (X \cup Y) \neq X\}\,. \qquad \blacksquare$$

6.4.4 LEMMA
If $f :| \mathbf{A} |\to| \mathbf{B} |$ is continuous with respect to $\mathcal{T}_{FG}$ then $\underline{f}_{fg}$ is well-defined and continuous, and furthermore $|\, \underline{f}_{fg}(z) \,|= f(|\, z \,|)$ for all $z \in| \mathcal{P}_{fg}(\mathbf{A}) |$. $\blacksquare$

Exercises

6.4.1 Calculate the structures of $| \mathcal{P}_{fg}(\mathbf{flat}(\mathbb{N})) |$ and $| \mathcal{P}_{fg}(\mathbf{A}) |$, where $\mathbf{A} = \mathbf{A}_{\uparrow}$. Comment on your answer.

6.4.2 Suppose $f :| \mathbf{A} |\to| \mathbf{A} |$ is monotonic and continuous with respect to $\mathcal{T}_{FG}$.

(i) Show that f is partial order continuous.
(ii) Show that if $\underline{f}_{fg}$ has only one fixed point then so does f.
(iii) If $| \mathbf{A} |$ has a greatest element, show that if f has only one fixed point then so does $\underline{f}_{fg}$.
(iv) Does the result of (iii) still hold if we omit the assumption that $| \mathbf{A} |$ has a greatest element?

[The results of this exercise can be interpreted as being of significance to *fixed point induction* – see Chapter 2.]

6.4.3 Union ($\underline{\bigcup}_{fg}$) and intersection ($\underline{\bigcap}_{fg}$) operators can both be defined from $| \mathcal{P}_{fg}(\mathcal{P}_{fg}(\mathbf{A})) |$ to $| \mathcal{P}_{fg}(\mathbf{A}) |$ in such a way that

$$|\, \underline{\bigcup}_{fg}\chi \,|= \bigcup\{|\, z \,|| \, z \in|\, \chi \,|\} \quad \text{and} \quad |\, \underline{\bigcap}_{fg}\chi \,|= \bigcap\{|\, z \,|| \, z \in|\, \chi \,|\}$$

(the second of these when the intersection is nonempty). Give suitable definitions and show that the subset $\{\chi \in| \mathcal{P}_{fg}(\mathcal{P}_{fg}(\mathbf{A})) \,||\, \underline{\bigcap}_{fg}\chi \text{ is defined}\}$ is itself finitely generable.

6.4.4 The topology $\mathcal{T}_{FG}$ on $| \mathcal{P}_{fg}(\mathbf{A}) |$ consists of the finitely generable subsets of $| \mathcal{P}_{fg}(\mathbf{A}) |$, each of which can be identified with a set of closed subsets of $| \mathbf{A} |$. Thus it actually defines a topology whose points are the closed sets in another one. (This is usually called a *hyperspace* topology.) In fact this particular hyperspace topology can be characterised quite nicely, purely in terms of the topology on $| \mathbf{A} |$, as the following show.

(i) Given any topological space $(X, \mathcal{T})$, the *upper* hyperspace topology on the closed sets of $\mathcal{T}$ is defined to be the smallest one where all sets of the form $\{C \mid C \subseteq U\}$, with U open in $\mathcal{T}$, are open. In the case where $X =\mid \mathbf{A} \mid$ and $\mathcal{T} = \mathcal{T}_{FG}$, show that the lower topology is precisely the Scott topology (see Section 2.4 and Exercise 6.3.6) on the set of $\mathcal{T}_{FG}$-closed sets ordered by reverse inclusion.

(ii) Given any topological space $(X, \mathcal{T})$, the *lower* hyperspace topology on the closed sets of $\mathcal{T}$ is defined to be the smallest one where all sets of the form $\{C \mid C \subseteq C'\}$, with C' closed in $\mathcal{T}$, are closed. Show that, for $(\mid \mathbf{A} \mid, \mathcal{T}_{FG})$ this coincides precisely with the lower topology already defined in Exercise 6.3.6 for the closed subsets of $\mid \mathbf{A} \mid$ ordered by reverse inclusion.

(iii) The *Vietoris* topology on the closed sets of a topology $(X, \mathcal{T})$ is defined to be the smallest one containing both the upper and the lower hyperspace topologies. Parts (i) and (ii), plus the result of Exercise 6.3.6 show that this is exactly the hyperspace topology induced by $\mathcal{T}_{FG}$ on $\mid \mathcal{P}_{fg}(\mathbf{A}) \mid$. What are the corresponding results for $\mid \mathcal{P}_S(\mathbf{A}) \mid$ and $\mid \mathcal{P}_H(\mathbf{A}) \mid$?

6.5 The Plotkin powerdomain

Readers who have attempted Exercise 6.1.8 will already know that it is impossible to define a domain constructor whose result is always the Plotkin powerdomain of its argument, because this domain constructor does not preserve consistent completeness. Given a cpo X, the Plotkin powerdomain will consist of the convex-closed nonempty finitely generable sets, ordered by the Egli-Milner order. Actually, we have not defined the notion of 'finitely generable' over an arbitrary cpo – readers interested in the definition should consult the Appendix below.

To get this constructor defined we need a theory of domains that admits partial orders that are not consistently complete. The best-known theory of this form is that of SFP-objects (SFP stands for 'sequence of finite partial orders'), which is described briefly in Chapter ??. The result, of course, is a domain with more sets in it than the Hoare or Smyth powerdomains. It has rather less sets than the powerdomain $\mathcal{P}_{fg}(\cdot)$ seen in the last section, but a partial order that will usually be more satisfactory.

As in $\mathcal{P}_{fg}(\mathbf{A})$, the singleton set operator is properly defined over the Plotkin powerdomain (since a singleton set is always convex). But it is now a continuous operator. Intersection is still a partial function because of the omission of the empty set; the definition of union requires an application of the convex closure operator, since for example if $x < y < z$ then we would expect $\{x\}_{EM} \cup_{EM} \{z\}_{EM} = \{x, y, z\}_{EM}$.

The fact that it is impossible to define this constructor over information systems is a little surprising when we observe that in a sense this powerdomain can be seen as a combination of the Hoare and Smyth domains, both of which are achievable without too much difficulty. Indeed one might think that a token set consisting of Hoare-tokens and Smyth-tokens (the first telling us about the greatest elements of a set, the second about the least elements) would be sufficient to describe elements of the Plotkin order – after all, every convex-closed set S has a canonical expression $S{\uparrow} \cap S{\downarrow}$ as the intersection of upwards- and downwards-closed sets.

What this tells us is that if, given a system $\mathbf{A}$, we take a token set $Con_A + (\wp(A) - \{\emptyset\})$, then the tokens

$$\{\iota_{left}(X) \mid \exists\, x \in S.X \subseteq^{\text{fin}} x\} \cup \{\iota_{right}(Y) \mid \forall\, x \in S.Y \cap x \neq \emptyset\}$$

describe every nonempty finitely generable convex-closed $S \subseteq| \mathbf{A} |$ completely. This set would of course be the map $\theta(\cdot)$ for identifying sets with elements of a hypothetical powerdomain. It is not at all hard to show that the subset order on these sets corresponds precisely to the Egli-Milner order on the sets S – so this is not what goes wrong (as something must)!

The difficulty comes when we try to invent an information system based on these tokens whose elements are exactly these $\theta(S)$. Any information system containing all of these elements must necessarily contain more, in general.

We argued much earlier in this book that information systems were a general model of computation because each process becomes identified with the set of all finite propositions true of it or finite observations true of it. It is interesting to look a little harder at this assertion in the context of the Plotkin powerdomain constructor. One of the fundamental assumptions that one makes in information systems is that every consistent, deductively closed set represents an element of the space of possible program values. The underlying philosophy here is that any process that has shown us such a set of tokens can legitimately decide to do no more (generally by looping). The problem in the Plotkin powerdomain is essentially that, given a set of observations which are consistent and from which no more can be deduced, it is not always reasonable to regard this set as complete because, even though one cannot deduce any more tokens, we can at least deduce that some more will eventually be observable.

This is perhaps made a little clearer by an example. Suppose we are trying to construct the Plotkin powerdomain using disjoint copies of the Hoare and Smyth tokens as described above. Let $\mathbf{A} = \mathbf{flat}(\{a, b\})$ and consider the Smyth token $\{a, b\}$. This says that every element of some subset S of $| \mathbf{A} | = \{\emptyset, \{a\}, \{b\}\}$ intersects $\{a, b\}$. If, as in the Hoare domain, every finite subset of each $x \in S$ is present as a (Hoare) token, any set with $\{a, b\}$ as a Smyth token must have either $\{a\}$ or $\{b\}$ as a Hoare token – but we do not know which. This problem

is surmountable (by picking different tokens) for $\mathcal{P}_{EM}(\mathbf{flat}(\{a, b\}))$ – since this is consistently complete. But by the time one gets to the slightly richer domain (actually the Cartesian product of $\mathbf{flat}(\{a, b\})$ with itself) seen in Exercise 6.1.8, *any* representation is bound to lead to a similar difficulty.

The question of whether it is operationally reasonable to ask for domains with this sort of 'I know there is something else to see but I don't know what' property is an interesting question. Of course the answer is subjective and may well depend on the context. If the answer were 'no' then one could not define the Plotkin powerdomain without adding additional elements (to make it consistently complete) in a similar way to the extra bottom added in the separated sum constructor.

Exercises

6.5.1 Compute the structure of the Plotkin powerdomains of $|\ \mathbf{flat}(\mathbb{N})\ |$ and $|\ \mathbf{A}\ |$, where $\mathbf{A} = \mathbf{A}_{\uparrow}$.

6.5.2 Let C be the set of all tokens of $\mathcal{P}_{fg}(\mathbf{A})$ with the form $\langle X, \emptyset\rangle$ or $\langle \emptyset, Y\rangle$. Show that $|\ \mathcal{P}_{fg}(\mathbf{A})\big|\!\setminus C\ |$ can be thought of as consisting of the convex, finitely generable nonempty subsets of $|\ \mathbf{A}\ |$, ordered by reverse inclusion.

6.6 Appendix: on finitely generable sets

The definition we have given of what it means to be finitely generable was purely in terms of the ideas of information systems, namely token sets. When powerdomains are presented in other formalisations of domain theory some other definition has to be given, usually in terms of partial orders alone. There are two good reasons for us to check that the simple definition we have used is equivalent to more usual ones. The obvious reason is that we should check that the definition used in this book is the same as that used elsewhere. The other reason is that it is by no means immediately obvious that the definition of finitely generable given earlier depends only on the partial order structure of $|\ \mathbf{A}\ |$ – one might think it possible that different subsets of some partial order would be deemed finitely generable depending on which representation of the order was chosen. In fact several of the results and exercises in this chapter prove that the definition is representation-independent, but it is a sufficiently important fact to warrant a direct proof – see Theorem 6.6.1 below.

The usual definition of what is meant by finitely generable (subsets of a countably (ω-)algebraic cpo P) is given in terms of finitely branching trees T. Each node n of the tree is labelled with an element x_n of P in such a way that all the children m of a node n are labelled so that $x_m \geq x_n$. A subset S of P is said to be

finitely generable if there is some such tree T where the set

$$\{\bigsqcup\{x_{n_i} \mid i \in \mathbb{N}\} \mid \langle n_i \mid i \in \mathbb{N}\rangle \text{ is an infinite path though } T\}$$

equals S. Now not all information systems are *countably* algebraic, so this tree definition is for a world that is neither smaller (for it does not assume consistent completeness) nor larger than ours. Thus the most we can hope to prove is that, in the intersection, the two definitions are the same. This will not be quite enough to meet the second objective above, for it will not prove anything about information systems that are not countably algebraic. Therefore our first result gives an alternative characterisation of finitely generable in terms of the partial order structure of an arbitrary information system.

6.6.1 THEOREM
A subset S of $|\,\mathbf{A}\,|$ is finitely generable if and only if, when $E = \{\overline{X} \mid X \in Con_A\}$ is the set of $|\,\mathbf{A}\,|$'s po-finite elements, S satisfies: whenever $x \notin S$, then there is a finite subset F of E such that

$$\forall\, x'.\{e \in F \mid e \subseteq x\} = \{e \in F \mid e \subseteq x'\} \Rightarrow x' \notin S\,.$$

PROOF
If S is finitely generable then it certainly meets the above condition, since if $x \notin S$ then there are Z, X such that if $x \cap Z = X$ and if, for any x', $x' \cap Z = X$ then $x' \notin S$. Thus the set $F = \{\overline{X}\} \cup \{\overline{\{a\}} \mid a \in Z - X\}$ will suffice to meet the condition in the statement of the theorem.

So suppose S satisfies the above condition; we must show that it is finitely generable. If $x \notin S$, let

$$F = \{\overline{X_i} \mid 1 \leq i \leq n\} \cup \{\overline{Y_i} \mid 1 \leq i \leq m\}$$

be the finite set given by the condition for x, where $\overline{X_i} \subseteq x$ for all i and $\overline{Y_i} \not\subseteq x$ for all i. Plainly an element x' of $|\,\mathbf{A}\,|$ satisfies

$$\{e \in F \mid e \subseteq x\} = \{e \in F \mid e \subseteq x'\}$$

if and only if $X_i \subseteq x'$ for all i and $Y_i \not\subseteq x'$ for all i. (This uses the fact that, for any information system $\mathbf{A} = \langle A, Con_A, \vdash_A\rangle$, if $X \in Con$ and $x \in |\,\mathbf{A}\,|$ then $X \subseteq x$ if and only if $\overline{X} \subseteq x$.) Setting $Z = \bigcup\{X_i \mid 1 \leq i \leq n\} \cup \bigcup\{Y_i \mid 1 \leq i \leq m\}$ we can thus state that $x \cap Z = x' \cap Z$ implies that $x' \notin S$. This is what was required to prove S finitely generable. ∎

6.6.2 THEOREM
If $|\ \mathbf{A}\ |$ is ω-algebraic then $S \subseteq |\ \mathbf{A}\ |$ is finitely generable if and only if there is a finite branching tree T of the type described above that generates it.

PROOF
Let $E = \{e_1, e_2, e_3, \ldots\}$ be the po-finite elements of $|\ \mathbf{A}\ |$.

First suppose that there is a tree T generating S and (for contradiction) S is not finitely generable. Then there is $x \notin S$ such that for all j there is $x_j \in S$ such that

$$\{e_k \mid 1 \leq k \leq j \wedge e_k \subseteq x_j\} = \{e_k \mid 1 \leq k \leq j \wedge e_k \subseteq x\}\,.$$

Converging to each of these x_j there must be an infinite path $\langle x_{j,i} \mid i \in \mathbb{N}\rangle$ of labels in T. It is easy to see that there is some i so that $i \leq i'$ implies

$$\{e_k \mid 1 \leq k \leq j \wedge e_k \subseteq x_{j,i'}\} = \{e_k \mid 1 \leq k \leq j \wedge e_k \subseteq x\}\,.$$

Let $i_j = max\{j, i\}$ and choose the node n_j to be the one labelled x_{j,i_j}.

Because we have ensured that n_j occurs at depth at least j in T, there must be infinitely many different n_j. Since T is, by assumption, finitely branching there is, by König's Lemma, an infinite path through T containing infinitely many different n_j. It is easy to see that the limit of the labels on this path is x, in contradiction to our assumption. This proves that all sets generated by trees are finitely generable.

To prove the reverse, suppose S is finitely generable; we must create a tree. For each j, create a node $\langle j, F\rangle$ for every set F which is of the form

$$\{e_i \mid 1 \leq i \leq j \wedge e_i \subseteq x\} \qquad \text{for } x \text{ in } S.$$

These will be the nodes of our tree, where the level j nodes are the $\langle j, F\rangle$ and where $\langle j+1, F'\rangle$ is a descendant of $\langle j, F\rangle$ if and only if $F \subseteq F'$. The node $\langle j, F\rangle$ is labelled by $\bigcup F$ (the definition of F ensures it is po-consistent). It is easy to check that this tree T generates precisely S. (Note that T has at most binary branching at each node and has all its labels po-finite.) ∎

Chapter 7

The lambda calculus

The λ-calculus is an abstract functional language that provided one of the main spurs to the early work on domain theory. It was invented by Church in the 1930s, but existed only as a syntactic formalism before Scott developed his first model for it in the late 1960s.

The λ-calculus provides the foundation – in terms of syntax, semantics and implementation – for all functional programming languages. Aside from this there are several other good reasons for including it in this book. It exhibits many of the difficulties and apparent paradoxes that we began to see in the introduction. Furthermore it does this in a very pure form, in that its syntax is simple and its semantics uncluttered by the relatively straightforward details that inevitably creep in with a real programming language. This means that we are able to analyse its semantics rather quickly, since we can concentrate on the essentials. From the time of Strachey on the λ-calculus, and notation derived from it, have been widely used as tools for the semantic description of more complex languages so an understanding of its semantics is almost an essential prerequisite for those interested in the semantics of real languages. Finally, for all the reasons described above, the λ-calculus has become the traditional test bed for semantic techniques, so it is useful to start here for comparative purposes.

The first part of this chapter gives an introduction to the λ-calculus as a formal, syntactic theory. It is only a summary of those parts of the theory that will later be of direct interest when we are constructing the semantics. Another description on the same lines can be found in [Stoy], but note that there a slightly different theory (including η-conversion, a conversion rule we do not adopt) is described. Anyone who is particularly enthusiastic about the λ-calculus as a formal system should have a look at *The Lambda Calculus, its Syntax and Semantics*, by Barendregt [Bar], where a great deal more of its theory is described. We end the

first section by some examples that illustrate the expressive power of the λ-calculus.

The second part describes its semantics in a reflexive domain. Because of its combination of depth and simplicity, we use the λ-calculus as the main example of this book to illustrate how a denotational semantics can be analysed in depth and related to operational behaviour. This study is begun in Section 7.2 and concluded in Chapter 11, where we are able to obtain a number of deep results arising from an analysis of operational semantics.

7.1 The syntax of the lambda calculus

The λ-calculus is a notation for the description and manipulation of functions. It can be regarded as the purest functional programming language: it consists only of those parts of such a language (abstraction and application) that are directly related to functions. Its syntax is therefore very simple.

SYNTACTIC DOMAINS

x	$\in$	Ide	identifiers	
E	$\in$	Exp	expressions	
$(b$	$\in$	Bas	basic values or atoms)	*Sometimes absent: see below.*

$$E ::= x \mid E_1\, E_2 \mid \lambda\, x.E\, (\mid b)$$

$E_1\, E_2$ represents the 'application' of E_1 to E_2, and $\lambda\, x.E$ represents the function that takes an argument and produces the result that comes from evaluating E with the argument substituted for x. ($\lambda\, x.E$ is an abbreviation for 'f, where $f(x) = E$'.) For example, we would expect the following λ-expressions to have the same meaning:

$$(\lambda\, x.(xy))(\lambda\, z.z) \qquad (\lambda\, z.z)y \qquad y.$$

(The second is obtained from the first by carrying out the substitution of $\lambda\, z.z$ implied by the outer application; the third is then obtained by carrying out the substitution of y for z in $\lambda\, z.z$.)

If Bas is present it represents any basic functional or non-functional objects that are assumed to be present, such as natural numbers or '+'. If it is present we say the language is a λ-calculus *with atoms*, or, if not, the λ-calculus *without atoms* or the *pure* λ-calculus. The rest of this chapter and the next are concerned with the λ-calculus without atoms, but almost everything can be adapted to the other case. Indeed, since realistic functional languages are, to a large extent, just ways of

dressing up a λ-calculus with atoms, we will be doing much of this adaptation in the next chapter.

In addition to these constructs we will allow bracketing in the concrete syntax of λ-expressions, to enable us to deduce their abstract syntax. We will respect the parsing convention, mentioned in Chapter 1, that functional application associates to the left. Thus $E_1\ E_2\ E_3\ E_4$ will be read as $((E_1\ E_2)\ E_3)\ E_4$. We will read abstractions $(\lambda\, x.)$ as extending over the largest expression available (i.e., extending to the right until the end of the expression or an unmatched right bracket is reached). Thus, $\lambda\, x.xy$ will be the same as $\lambda\, x.(xy)$, and $\lambda\, x.\, \lambda\, y.xy$ the same as $\lambda\, x.(\lambda\, y.(xy))$.

Functional programmers may already be familiar with many of these ideas.

We study this notation first by formalising syntactic manipulations such as the above. This will give us the insight necessary to formulate and analyse its semantics later in this chapter.

Definition
An occurrence of an identifier in a λ-expression can be said to be *free* or *bound*, much as in the predicate calculus; the binding being done by λ rather than $\forall$ or $\exists$.

(a) x appears free in y if and only if $x = y$.

(b) x appears free in $E_1\ E_2$ if and only if it appears free in E_1 or in E_2.

(c) x does not appear free in $\lambda\, x.E$.

(d) If $x \neq y$ then x appears free in $\lambda\, y.E$ if and only if it appears free in E.

(a) x does not appear bound in y.

(b) x appears bound in $E_1\ E_2$ if and only if it appears bound in E_1 or in E_2.

(c) x appears bound in $\lambda\, x.E$.

(d) If $x \neq y$ then x appears bound in $\lambda\, y.E$ if and only if it appears bound in E. ∎

A λ-expression without any free identifiers is said to be *closed* and is sometimes termed a *combinator*.

In order to be able to 'evaluate' a λ-expression of the form $(\lambda\, x.E_1)\ E_2$ syntactically we need to be able to substitute E_2 for every free occurrence of x in E_1. What does this mean? Apparently

$$(\lambda\, x.\, \lambda\, y.x)\ y\ =\ \lambda\, y.y \qquad \text{(the identity function)}$$

whereas

$$(\lambda\, x.\, \lambda\, z.x)\ y\ =\ \lambda\, z.y \qquad \text{(a constant function).}$$

This is counter-intuitive because one would not expect the name of a bound identifier to matter: $\lambda\, z.x$ and $\lambda\, y.x$ should both have the same meaning and one would expect that substituting E for x in two expressions with the same meaning would produce equivalent results. Of course this problem arises because a free identifier in the substituted expression may become bound when it is simply copied into the unaltered function body. Exactly the same problem arises in predicate calculus and we adopt the same solution as is traditional there: during a substitution we carry out systematic renaming of bound identifiers to avoid binding any free identifiers. To do this we assume *Ide* contains a denumerably infinite subset $\{z_1, z_2, z_3, \ldots\}$ from which we can select a new identifier when one is needed. This set is always sufficient because every λ-expression is finite and so contains only finitely many different identifiers.

Definition
If $x \in Ide$ and $E, E' \in Exp$, we define $[E'/x]E$ (the substitution of E' for x in E) as follows.

(a) If $E = x$ then $[E'/x]E = E'$.

(b) If $E = y$ and $x \neq y$ then $[E'/x]E = y$.

(c) If $E = E_1\, E_2$ then $[E'/x]E = ([E'/x]E_1)\,([E'/x]E_2)$.

(d) If $E = \lambda\, x.E_1$ then $[E'/x]E = E$.

(e) If $E = \lambda\, y.E_1$ and $y \neq x$ and either x does not appear free in E_1 or y does not appear free in E' then $[E'/x]E = \lambda\, y.[E'/x]E_1$.

(f) If $E = \lambda\, y.E_1$ and $y \neq x$ and x appears free in E_1 and y free in E' then $[E'/x]E = \lambda\, z_i.[E'/x]([z_i/y]E_1)$, where z_i is the first of our list of identifiers not to occur free in E_1 or E'. ∎

This definition needs to be justified by induction on the syntactic complexity of E (depth of its syntax tree) rather than by straightforward structural induction, because of the double substitution in clause (f). The first substitution $[z_i/y]E_1$ simply replaces one identifier by another and so (provably) leaves the syntactic complexity unaffected[1].

We should now regard $(\lambda\, x.E_1)\, E_2$ as having the same meaning as $[E_2/x]E_1$ and $\lambda\, x.E$ as having the same meaning as $\lambda\, y.[y/x]E$ provided y is not free in E. These ideas can be expressed formally as *conversion rules*.

[1]This would not have been the case if the inner substitution replaced the identifier with any other type of expression.

Definition
The binary relations $\underline{\text{cnv}}_\alpha$ and $\underline{\text{cnv}}_\beta$ on λ-expressions are defined as follows.

$$\lambda\, x.E \quad \underline{\text{cnv}}_\alpha \quad \lambda\, y.[y/x]E \quad \text{provided } y \text{ is not free in } E$$

$$(\lambda\, x.E_1)\, E_2 \quad \underline{\text{cnv}}_\beta \quad [E_2/x]E_1$$

(Right-to-left conversions are allowed as well as left-to-right.) In general we say E_1 $\underline{\text{cnv}}$ E_2 if there is a finite sequence of λ-expressions, the first of which is E_1 and the last E_2, each of which is obtained from its predecessor by some conversion (in either direction) of the whole expression or some subexpression. ∎

We will regard expressions that can be interconverted as equivalent.

Generally, α-conversion will be used to remove confusion of the type encountered above and β-conversion, in the left-to-right direction, to 'evaluate' an expression – to simplify it by eliminating λ-abstractions. We will term these two forms of conversion *reductions*. If E_1 $\underline{\text{cnv}}$ E_2 using only α- and left-to-right β-conversion, we write E_1 $\underline{\text{red}}$ E_2.

A *redex* is an expression that can be β-reduced (i.e., one of the form $(\lambda\, x.E_1)\, E_2$). If an expression contains no redexes then it cannot be significantly reduced any further, for α-conversion only changes the names of bound variables, never the overall shape of an expression. In particular, α-conversion never introduces a redex. Such an expression is said to be in *normal form*.

Some expressions contain several redexes, for example

$$(\lambda\, x.xx)((\lambda\, y.\, \lambda\, z.y)(\lambda\, t.t))$$

and therefore have several possible reduction routes. There is no immediate reason why these should always lead to the same normal form. Indeed it is quite possible to have two routes, one of which terminates and the other does not. Consider, for example, the expression

$$(\lambda\, x.\, \lambda\, y.y)((\lambda\, z.zz)(\lambda\, z.zz))$$

which does not alter if the right-hand expression is reduced but, if the outer redex is reduced, is immediately in normal form.

It can however be proved that if two reduction sequences both terminate then they both lead to the same result (modulo α-conversion). This is a direct consequence of the following theorem (which we do not prove here, but is proved in Barendregt [Bar], for example).

7.1.1 THEOREM (THE CHURCH-ROSSER THEOREM)
If E and E' are λ-expressions and E $\underline{\text{cnv}}$ E', then there is a λ-expression E^* such that E $\underline{\text{red}}$ E^* and E' $\underline{\text{red}}$ E^*. ∎

7.1.2 COROLLARY
If E is a λ-expression and E $\underline{\text{cnv}}$ E_1 and E $\underline{\text{cnv}}$ E_2 where E_1 and E_2 are in normal form, then E_1 may be converted to E_2 using only α-conversions.

PROOF
We know, by the theorem, that there exists E^* that both E_1 and E_2 reduce to. But the only reduction that can be carried out on normal form expressions is α-conversion, so all steps in the paths from E_1 and E_2 to E^* must be α-conversions. Thus E_1 may be converted to E_2 by following the path to E^* and then reversing the steps made in getting from E_2 to E^*. ∎

Some λ-expressions reduce to normal form whatever reduction strategy is employed and some need to be reduced with care to avoid an infinite path. There are also some that have no normal form at all, for example

$$(\lambda\, x.xx)(\lambda\, x.xx)$$

where the only reduction leaves it unaltered and

$$(\lambda\, x.xxx)(\lambda\, x.xxx)$$

where each successive reduction only serves to increase the expression's length. One very interesting combinator with no normal form is

$$Y \;=\; \lambda f.(\lambda\, x.f(xx))(\lambda\, x.f(xx)),$$

because for any expression E it is easy to see that

$$Y\ E\ \underline{\text{cnv}}\ E(Y\ E)$$

(if necessary Y is α-converted to avoid any clashes with E). Thus Y is a 'function' that apparently returns a fixed point of any function to which it is applied. For this reason it is sometimes termed the *paradoxical* combinator. (After all, there are many functions in mathematics that have no fixed point.) Y is the simplest of infinitely many λ-expressions (known as fixed-point combinators) that have the same effect. Some more appear in the exercises in this chapter.

There are a number of strategies that can be employed in an attempt to reduce λ-expressions to normal form. Perhaps the most obvious one is *applicative order* reduction, which can be summarised as follows.

(a) An identifier x is already in normal form.

(b) To reduce $\lambda\, x.E$, reduce E.

(c) To reduce $E_1\,E_2$, first reduce E_1 and E_2 to E_1' and E_2'. If E_1' is not of the form $\lambda\,x.E$, the expression is already in normal form. Otherwise carry out the β-conversion and reduce the resultant expression.

This means that, when evaluating the application of a function to its argument, both the function and its argument are evaluated as much as possible before 'applying' the function. The fact that the argument is evaluated before substitution will be efficient when the function uses it more than once within its body. However, as in the example above, it can lead to problems if the function ignores its argument and the reduction of the argument expression does not terminate. There is a reduction strategy which is always guaranteed to terminate if termination is possible. It is called *normal order* reduction, and can be described in one line.

- Always reduce the *leftmost* redex. That is, the one with the leftmost λ.

This strategy does essentially the reverse of the earlier one: it always substitutes a function's argument for the corresponding identifier before reducing the argument at all. This means that if the argument is in fact ignored, it is never reduced. On the other hand if it is used several times it has to be reduced more than once. While this is inefficient it cannot lead to nontermination: if evaluating the same expression several times fails to terminate, so does evaluating it once. The subject of reduction strategies is very important in connection with real functional programming languages, and we will return to it when we discuss their semantics later.

We end this section with a number of examples to illustrate the use of the λ-calculus and also to demonstrate its considerable expressive power.

EXAMPLE: REPLACING ABSTRACTION BY COMBINATORS

Recall that a combinator is a λ-expression with no free identifiers. It is an interesting and useful fact that it is possible to eliminate all use of λ-abstraction expressions in favour of a small number of combinators (i.e., where the only uses of abstraction appear in these combinators). Just how small a set of combinators is necessary will be seen later, but for now we will observe that there is an 'adequate' set of three. These are defined:

$$\begin{aligned} I &= \lambda\,x.x \\ K &= \lambda\,x.\,\lambda\,y.x \\ S &= \lambda\,x.\,\lambda\,y.\,\lambda\,z.x\,z\,(y\,z) \end{aligned}$$

The first of these is the identity function, the second takes two arguments and returns the first. The third combinator S is of a form specially designed to play its role of the inductive step of the Lemma below.

Our main result is the following theorem:

7.1.3 THEOREM
If E is a λ-expression all of whose free identifiers are contained in the set V, then there is an expression E' such that E $\underline{\text{cnv}}$ E' and E' involves only the identifiers of V, the combinators S, K, I, and application.

PROOF
The proof of this theorem essentially consists of the demonstration that we can produce a 'compiler' for converting λ-expressions into the special form. Most of the work is done by the following Lemma.

LEMMA
Suppose E is an expression involving only the identifiers of V, S, K, I and application. Then there exists E' of the same form (with $V \setminus \{x\}$ replacing V) such that $\lambda\, x.E$ $\underline{\text{cnv}}$ E'.

PROOF OF LEMMA
This is proved by structural induction on E. If $E = x$ then we set $E' = I$, and if either $E = y$ $(y \neq x)$ or E is S, K or I then we set $E' = K\, E$. The only case remaining is where $E = E_1 E_2$, and we then know by induction that there exist E'_1 and E'_2 of the correct form such that E'_i $\underline{\text{cnv}}$ $\lambda\, x.E_i$. All we then have to observe is that $S\, E'_1\, E'_2$ $\underline{\text{cnv}}$ $\lambda\, z.(\lambda\, x.E_1)\, z\, ((\lambda\, x.E_2)\, z)$ which after two β-reductions and an α-conversion becomes $\lambda\, x.E_1\, E_2$ as desired. This completes the proof. ∎

Observe that the second clause (the one using K) can also be applied in the case where E is any expression with no free x. Judicious use of this observation can lead to much shorter combinator expressions in the output of our compiler.

Given this Lemma, the proof of the Theorem is now an easy structural induction. If E is a identifier then it does not need to be changed. If $E = E_1 E_2$ then we can use induction to give us E'_1 and E'_2 of the correct form such that E_i $\underline{\text{cnv}}$ E'_i. We then set $E' = E'_1 E'_2$. If $E = \lambda\, x.E_1$ we use induction to give us E'_1 of the correct form (using the identifiers in V plus x) and then appeal to the Lemma to observe that there is an expression of the correct form equivalent to $\lambda\, x.E'_1$. ∎

Unfortunately the expressions produced by this mechanical process are generally very long and unwieldy. The reader might like to verify this by applying the process to the λ-expression S, namely $\lambda\, x.\, \lambda\, y.\, \lambda\, z.x\, z\, (y\, z)$. There are various 'optimisations' which can be applied but even so the process described above rarely produces the most elegant expression.

It is often observed that the three combinators can be reduced to two, since $S\, K\, K$ $\underline{\text{cnv}}$ I and so all instances of I in the output of our compiler may be replaced

by SKK. In fact the set of combinators can be reduced in size still further. If we define

$$W = \lambda\, e.e\, S\, (\lambda\, x.\, \lambda\, y.\, \lambda\, z.x)$$

then observe that, abbreviating $\lambda\, x.\, \lambda\, y.\, \lambda\, z.x$ by E,

$$\begin{array}{lll} WW & \underline{\text{cnv}}_\beta & W\, S\, E \\ & \underline{\text{cnv}}_\beta & S\, S\, E\, E \\ & \underline{\text{cnv}}_\beta & S\, E\, (E\, E) \\ & \underline{\text{cnv}}_\beta & \lambda\, z.E\, z\, (E\, E\, z) \\ & & \quad (=\ \lambda\, z.(\lambda\, x.\, \lambda\, y.\, \lambda\, t.x)\, z\, (E\, E\, z)\ ) \\ & \underline{\text{cnv}}_\beta & \lambda\, z.\, \lambda\, t.z \\ & \underline{\text{cnv}}_\alpha & K \end{array}$$

and that $W(W\, W)$ $\underline{\text{cnv}}$ $W\, K$ $\underline{\text{cnv}}$ $K\, S\, E$ $\underline{\text{cnv}}$ S. Thus there is an expression involving only W for both S and K, so W is adequate by itself! Other examples of one-element bases (adequate sets) can be found in Exercise 7.1.6 and [Bar].

It has sometimes been argued that it is better to present the λ-calculus using combinators (such as S, K and I) rather than λ abstraction. To do this one gives a number of conversion rules such as $I\, E \equiv E$, $K\, E\, F \equiv E$ and $S\, E\, F\, G = E\, G\, (F\, G)$. One argument for this is that there is no need any more to name bound identifiers – the presence of which in the standard λ-calculus leads to a certain loss of abstractness. See Barendregt for more details.

Combinators have been much used recently in the implementation of functional languages. Programs are reduced, as above, to expressions involving a limited range of combinators (in some cases S, K and I) and implemented on a machine which uses the combinators as machine code. Thus the proof of Theorem 7.1.3 above does in a genuine sense generate a compiler. The existence of the W combinator means that it would be possible to construct a machine with only one instruction!

There are some exercises on combinators at the end of this section.

EXAMPLE: BOOLEANS

Although the λ-calculus only directly contains the ideas of functional application and abstraction, it can be used to model various other concepts. One of these is the boolean values *true* and *false*. It is convenient to model these by the combinators $K = \lambda\, x.\, \lambda\, y.x$ and $K\, I = \lambda\, x.\, \lambda\, y.y$ both of which take two arguments and return one of them. They are not interconvertible for they are both in normal form.

All the usual propositional connectives may be defined. For example $not = \lambda\, b.b\, (KI)\, K$, $and = \lambda\, p.\, \lambda\, q.p\, q\, (K\, I)$ and $or = \lambda\, p.\, \lambda\, q.p\, K\, q$. The reader should experiment to see why these definitions work and try to give definitions of some other connectives.

The connectives are specified only to the extent that they must give one particular answer when applied to an appropriate number of arguments of the form K and KI. Thus there are alternative definitions for the connectives, including the ones we have given here, representing different functions when applied to other arguments. An example is $not' = \lambda\, b.\, \lambda\, x.\, \lambda\, y.byx$.

EXAMPLE: TUPLES

A *tuple* is an ordered pair, triple, etc. of objects. An n-tuple is usually written $\langle a_1, a_2, \ldots, a_n\rangle$. Tuples should have the properties than for any $a_1,\ a_2,\ \ldots, a_n$ the n-tuple $\langle a_1, a_2, \ldots, a_n\rangle$ exists, and that there are projection functions π_i^n such that $\pi_i^n\langle a_1, a_2, \ldots, a_n\rangle = a_i$. It is easy to construct tupling and projection functions in the λ-calculus. We will first treat ordered pairs.

The combinator *pair* is defined

$$pair = \lambda\, x.\, \lambda\, y.\, \lambda\, z.z\ x\ y$$

so that the pair $\langle E_1, E_2\rangle$ is identified with the expression $\lambda\, z.z\ E_1\ E_2$. The components of a pair can be recovered by the two combinators

$$fst = \lambda\, p.p(\lambda\, x.\, \lambda\, y.x) \quad \text{and} \quad snd = \lambda\, p.p(\lambda\, x.\, \lambda\, y.y)\ .$$

Note that the first and second components of a pair are recovered by applying the pair to K and KI respectively (which were identified with 'true' and 'false' in the last example.)

One could define longer tuples by iterated pairing, since one can identify for example the triple $\langle a_1, a_2, a_3\rangle$ with the pair $\langle a_1, \langle a_2, a_3\rangle\rangle$. But it possible to extend the ideas used to define pairing directly, so we can define the n-tupling combinator as follows:

$$T^n = \lambda\, x_1. \ldots \lambda\, x_n.\, \lambda\, z.z\ x_1 \ldots x_n$$

so that $\langle E_1, E_2, , \ldots, E_n\rangle$ is identified with $\lambda\, z.z\ E_1\ \ldots E_n$. The projection function π_i^n is then $\lambda\, t.t\ C_i^n$ where C_i^n is the combinator which selects the ith of n arguments:

$$C_i^n = \lambda\, x_1. \ldots \lambda\, x_n.x_i\ .$$

EXAMPLE: THE CHURCH NUMERALS

One of the most interesting and theoretically useful classes of λ-expressions are the so-called *Church numerals*, which were used by Church to show that the λ-calculus could be said to contain arithmetic. He sought to identify the natural number n with the expression $\underline{n} = \lambda f.\, \lambda\, x.f(f \ldots (f\ x) \ldots)$ (f applied n times to x). Thus

$\underline{0} = \lambda f.\,\lambda x.x$ and $\underline{1} = \lambda f.\,\lambda x.f\,x$. It is possible to define a successor function

$$succ = \lambda n.\,\lambda f.\,\lambda x.f(n\,f\,x)$$

and to build up from it a variety of the usual arithmetic operations. There are a number of ways of defining these, for example *add* may be defined to be either of the combinators

$$\lambda n.\,\lambda m.m\;succ\;n \qquad \text{or} \qquad \lambda n.\,\lambda m.\,\lambda f.\,\lambda x.n\,f\,(m\,f\,x)$$

amongst others. (These two expressions are truly different as λ-expressions even though they have the same effect when applied to two Church numerals – they cannot be interconverted as they are both in normal form, and see also Theorem 7.2.5 below. This is the same sort of ambiguity we saw with logical combinators.)

Indeed, a little work establishes that *all* computable functions of the natural numbers may be encoded as λ-expressions using the Church numerals. There are numerous characterisations of what a 'computable' function is (see Cutland [], for example) and a number of these can be used to prove this claim. The principle that underlies these several characterisations of computability is known as *Church's thesis*, which states that all reasonable 'complete' models of computation have the same expressive power. But perhaps the most natural way of showing it in this case is to build up all the elements of a more-or-less standard functional programming language over the numbers and booleans such as the one seen in Section 1.2. The λ-calculus already provides us with the framework of application and abstraction; it gives us recursion via Y and tupling via the trick seen above. A non-recursive $\underline{define}\ldots$ construct can easily be modelled using abstraction and application:

$$\underline{define}\;\; x = E_1 \mathbf{in}\; E_2 \quad \text{becomes} \quad (\lambda x.E_2)E_1$$

with a similar trick for functional definitions. The booleans can be modelled using $K = \lambda x.\,\lambda y.x$ and $KI = \lambda x.\,\lambda y.y$ as above and so the conditional construct

$$\underline{if}\;\; E_1\;\; \underline{then}\;\; E_2\;\; \underline{else}\;\; E_3 \quad \text{becomes} \quad E_1E_2E_3\,.$$

For clarity it is better to write the right hand side above as

$$cond\,E_1\,E_2\,E_3 \quad \text{where} \quad cond = \lambda x.\,\lambda y.\,\lambda z.x\;y\;z$$

to distinguish it from the more standard use of application. (The fact that conditional composition becomes application is an accident arising from our choice of representation of the booleans. The use of the *cond* connective makes this decision less crucial by modularising it into this one place: a different representation of booleans could be matched by a different *cond*.)

Thus the only things left to define are the various 'standard' functions of a programming language such as $+$, $-$, $\times$, $/$, $=$, $\leq$ and *and*. The boolean combinators have already been discussed above, as have the successor and addition functions. We define two more here before leaving the rest as an exercise to the reader.

Rather than define full equality immediately it is useful to define first a test for equality with $\underline{0}$. This is quite simply done as follows:

$$iszero = \lambda\, n.n(\lambda\, x.KI)K\ .$$

When applied to a Church numeral, this function returns K or KI depending on whether or not the numeral is $\underline{0}$. (The definition of the full equality over numerals is dealt with in Exercise 7.1.2.)

The *predecessor* function is much trickier to define than the successor function. (The predecessor function maps $\underline{n+1}$ to $\underline{n}$ and $\underline{0}$ to $\underline{0}$.) Consider the combinator P defined to be

$$\begin{aligned}\lambda\, p.cond(iszero\,(fst\,p))(pair\,(succ\,(fst\,p))\,(snd\,p))\\ (pair\,(fst\,p)\,(succ\,(snd\,p)))\end{aligned}$$

This is a function which, when given a pair of numerals, tests the first for equality with $\underline{0}$. If it is not zero it returns the same pair with the successor function applied to the second component. If it is zero, it returns the same pair with the successor function applied to the first component. This allows us to define

$$pred = \lambda\, n.snd(n\ P\ (pair\,\underline{0}\,\underline{0}))$$

since the result of applying P $n+1$ times to $pair\,\underline{0}\,\underline{0}$ is $pair\underline{1}\,\underline{n}$.

See the exercises for ways in which these functions can be used to build up the rest of the standard functions.

Exercises

7.1.1 Reduce each of the following λ-expressions as far as possible or reasonable. In which of the expressions does the order of reduction make any difference? Which of the expressions has a normal form?

(i) $(\lambda\, x.\, \lambda\, y.yx)(\lambda\, x.yx)$
(ii) $(\lambda\, x.\, \lambda\, y.xyx)(\lambda\, x.\, \lambda\, y.xyx)$
(iii) $(\lambda\, y.\, \lambda\, x.xyx)z(\lambda\, y.\, \lambda\, x.xyx)$
(iv) $(\lambda\, y.\, \lambda\, x.x(xy))(\lambda\, y.yz)(\lambda\, x.xx)$
(v) $(\lambda\, x.xx)(\lambda\, x.yxxx)$
(vi) $(\lambda\, x.xx)(\lambda\, x.y(xxx))$

(vii) $(\lambda\, z.\, \lambda\, t.tz)((\lambda\, y.\, \lambda\, x.xyx)z(\lambda\, y.\, \lambda\, x.xyx))(\lambda\, z.\, \lambda\, t.t)$

7.1.2 Recall our definitions of successor and addition over the Church numerals. Find λ-expressions for multiplication *(times)* and exponentiation *(exp)* so that

$$\begin{array}{rcl} times\, \underline{n}\, \underline{m} & \underline{\text{cnv}} & \underline{n \times m} \\ exp\, \underline{n}\, \underline{m} & \underline{\text{cnv}} & \underline{n^m} \end{array}$$

Now use the *pred* function to define a subtraction function *minus* so that $minus\, \underline{n}\, \underline{m} = \underline{n - m}$ or $\underline{0}$ depending on whether $m \leq n$ or not. By using *pred* or otherwise, define a function that tests for equality between arbitrary Church numerals.

7.1.3 Find expressions using only S, K, I which convert to the following

(i) $\lambda\, x.\, \lambda\, y.\, \lambda\, z.y$
(ii) $(\lambda\, x.xx)(\lambda\, x.xx)$.
(iii) $\lambda\, x.\, \lambda\, y.x(xy)$

7.1.4 Let

$$Y_0 = \lambda f.(\lambda\, x.f(xx))(\lambda\, x.f(xx)), \text{ and}$$
$$G = \lambda\, y.\, \lambda f.f(yf)\ .$$

Inductively define $Y_{n+1} = Y_n\ G\ \ (n \geq 0)$.

7.1.5 Verify that, for each λ-expression E, the λ-expression

$$Y^E\ =\ \lambda f.(\lambda\, y.\, \lambda\, x.f(xyx))E(\lambda\, y.\, \lambda\, x.f(xyx))$$

is a fixed point operator (i.e. $Y^E f\ \underline{\text{cnv}}\ f(Y^E f)$).

7.1.6 Show that the combinator $\lambda\, e.e\ C_3^4 S$ is adequate in the same sense as W, namely that every λ-expression can be re-written using only it, free identifiers and application.

7.1.7 Suggest a way of replacing the representation of natural numbers (via Church numerals) with a representation of all integers. Define some arithmetic and comparison operators over this new class.

7.2 The semantics of the lambda calculusc

It would appear that every object in the pure λ-calculus is a function (after all, we are at liberty to apply any expression to any other). If we take this philosophy as the basis of our semantics it leads to the domain equation $\mathbf{D} = \mathbf{D} \to \mathbf{D}$. We have

already seen, in Section 5.2, that this equation has a trivial natural solution, and that one needs to go to special lengths to get non-trivial solutions. Of course, the reason this happens is that there is really no good reason for wanting to tell two objects apart in a language where all we are told is that every program is a function from programs to programs, because it is perfectly consistent to believe that there is only one program value: a representation x of a function which, when given x as argument, returns x. Any non-trivial model must result from imposing some distinctions between λ-expressions, and then using these to discriminate between the others. (These distinctions must necessarily have their origin outside the basic philosophy that the value of a λ-expression is a function from the values of λ-expressions to themselves.)

We will adopt a slightly different philosophy of what λ-expressions can be. We do not admit that *every* expression in the pure λ-calculus can properly be called a function. Consider for example the expression

$$(\lambda\, x.xx)(\lambda\, x.xx)\ .$$

When we try to apply this to something else we just go straight down an infinite loop: the argument never even gets looked at. If we were in a λ-calculus with atoms, we would go down exactly the same loop if we tried to evaluate this expression as an integer; it would not produce the 'error' value that one would expect if one tried to evaluate a function as an integer. We take the view that an infinite loop should no more be regarded as a function than as an integer: a λ-expression should not be considered to be a function unless it can demonstrate that it is one (by conversion to the form $\lambda\, x.E$ so that it can actually use an argument). Note also that in a λ-calculus with atoms, even though we would probably not choose to regard 3 as a function, there is nothing to prevent us writing an expression like $3(\lambda\, x.x)$: thus it is sometimes reasonable not to regard an object as a function, even though it can be applied like one. We expect that looping expression above will be identified with the minimal element of the domain, because it can prove nothing about itself at all.

This gives us a natural reason for wanting to distinguish between two objects. (Is an object a function, or not?) This philosophy leads us to the domain equation

$$\mathbf{D}\ =\ (\mathbf{D} \to \mathbf{D})_{\uparrow}$$

Every object is either $\emptyset$ (because it cannot be demonstrated to be a function) or is a function from the domain to itself. We have seen that the element set of the only solution to this equation is infinite. Thus a pleasant side-effect of our philosophy is that we naturally get an interesting mathematical model. We will see later that the natural semantics for the λ-calculus in this domain ties in very well with this philosophy.

Perhaps the best reason to use this domain is that it is much closer to what is required for modelling real functional languages, since as in the case of a λ-calculus with atoms, the domain will have to be extended to handle non-functional types. The evaluation/reduction strategies of such languages (especially ones with non-lazy semantics) tie in much better with the philosophy of $\mathbf{D} = (\mathbf{D} \to \mathbf{D})_\uparrow$ than that of $\mathbf{D} = (\mathbf{D} \to \mathbf{D}$.

It is this choice of semantic model which forces us to abandon η-conversion, for we wish to distinguish

$$E_1 = (\lambda\, x.xx)(\lambda\, x.xx)$$

(which is not a function) from

$$E_2 = \lambda\, y.(\lambda\, x.xx)(\lambda\, x.xx)y$$

(which is).

Now that we have decided on this philosophy for the semantics of the λ-calculus, the definitions of the semantic domains and functions are straightforward.

Semantic Domains

$$\begin{array}{lll} \epsilon, \delta \in \mid \mathbf{D} \mid & & \text{where } \mathbf{D} = (\mathbf{D} \to \mathbf{D})_\uparrow \\ \rho \in U & = Ide \to \mid \mathbf{D} \mid & \text{(environments)} \end{array}$$

We will, in defining and analysing the semantics, frequently make use of the fact that $\mathbf{D}$ is the solution to this equation. Define $\langle A, Con, \vdash \rangle = \mathbf{D}$ and $\langle A', Con', \vdash' \rangle = \mathbf{D} \to \mathbf{D}$, so that we have notation for the token sets etc. of these systems, for use in the rest of this section.

Semantic Function

$$\mathcal{E} : Exp \to U \to \mid \mathbf{D} \mid$$

Semantics

$$\begin{array}{rcl} \mathcal{E}[\![x]\!]\rho & = & \rho[\![x]\!] \\ \mathcal{E}[\![E_1\, E_2]\!]\rho & = & apply(\mathcal{E}[\![E_1]\!]\rho)(\mathcal{E}[\![E_2]\!]\rho) \\ & & \text{where } apply\, \epsilon\, \delta = \{a \in A \mid \exists\, X \subseteq^{\text{fin}} \delta.\, \langle X, a\rangle \sqsubseteq \epsilon\} \\ \mathcal{E}[\![\lambda\, x.E]\!]\rho & = & \iota_\uparrow(\{\langle X, a\rangle \mid X \in Con \wedge a \in \mathcal{E}[\![E]\!]\rho[\overline{X}/x]\}) \end{array}$$

The semantics of application and abstraction are exactly what we might have expected from the construction of our domain. *apply* is a function which takes two elements of $\mid \mathbf{D} \mid$ and 'applies' the first to the second (ϵ to δ, say). To do this it

sees what tokens it can deduce under the approximable mapping ϵ from the finite subsets of δ. Note that because $\epsilon \in |\ (\mathbf{D} \to \mathbf{D})_{\uparrow}\ |$ rather than $|\ \mathbf{D} \to \mathbf{D}\ |$ the tokens of ϵ need to be 'projected' before they can be applied: $\langle X, a\rangle \sqsubseteq \epsilon$ is equivalent to $\langle X, a\rangle \in \pi_{\uparrow}(\epsilon)$. Notice that in the case where $\epsilon = \emptyset$ (the minimal element of $|\ \mathbf{D}\ |$, the only element that, since it is not in the lifted part of the domain, fails to be a 'function'), we have $\mathit{apply}\,\epsilon\,\delta = \emptyset$. Since we are to identify expressions which loop endlessly in an effort to prove they are functions with $\emptyset$, this is just what we would expect: if the evaluation of E_1 loops before it has taken its argument there is no way in which the evaluation of $E_1\ E_2$ can fail to loop.

Notice that any abstraction is automatically given semantics in the lifted part of the domain: this is because such expressions are demonstrably functions. To compute the value as an approximable mapping of an abstraction, we need to see what we can deduce from each finite consistent set X of tokens. This is done by evaluating the body of the abstraction with the closure of X substituted for the identifier of the abstraction.

7.2.1 THEOREM
For every $E \in \mathit{Exp}$, $\mathcal{E}[\![E]\!]$ is a well-defined continuous function from U to $|\ \mathbf{D}\ |$.

PROOF
We use structural induction on $E \in \mathit{Exp}$.

$\mathcal{E}[\![x]\!]$ is well-defined since, for each $\rho \in U$, $\mathcal{E}[\![x]\!]\rho = \rho[\![x]\!]$ which is in $|\ \mathbf{D}\ |$ by definition of U. If $\Delta \subseteq U$ is directed then, if $\rho^* = \bigsqcup \Delta$, we have

$$\rho^*[\![y]\!] = \bigcup_{\rho\in\Delta} \rho[\![y]\!]$$

for each $y \in \mathit{Ide}$. Thus

$$\bigcup_{\rho\in\Delta} \mathcal{E}[\![x]\!]\rho = \bigcup_{\rho\in\Delta} \rho[\![x]\!] = \rho^*[\![x]\!] = \mathcal{E}[\![x]\!]\rho^*\ .$$

So $\mathcal{E}[\![x]\!]$ is continuous.

As the main step in proving the result for the $E_1\ E_2$ clause we will prove that *apply* is well-defined and continuous. So suppose that $\epsilon, \delta \in |\ \mathbf{D}\ |$. If $\{a_1, a_2, \ldots, a_n\} \subseteq^{\text{fin}} \mathit{apply}\,\epsilon\,\delta$ then we can find $X_1, X_2, \ldots, X_n \subseteq^{\text{fin}} \delta$ such that $\langle X_i, a_i\rangle \sqsubseteq \epsilon$ for each i. But then $X^* = \bigcup_{i=1}^{n} X_i \in \mathit{Con}$ (as δ is consistent), so the consistency relation of the function space constructor tells us $\{a_1, a_2, \ldots, a_n\} \in \mathit{Con}$ (consistent information in. consistent information out).

For deductive closure, suppose $X = \{a_1, a_2, ..., a_n\} \subseteq^{\text{fin}} \mathit{apply}\,\epsilon\,\delta$ and $X \vdash b$. Then we can find X_i and X^* exactly as above. In the information system $\mathbf{D} \to \mathbf{D}$ we then have

$$\{\langle X_1, a_1\rangle, \ldots, \langle X_n, a_n\rangle\} \vdash' \langle X^*, b\rangle, \ \text{ since } \{a_i \mid X^* \vdash X_i\} \vdash b.$$

Hence $\langle X^*, b\rangle \sqsubseteq \epsilon$ as ϵ is a (deductively closed) element of $|\ \mathbf{D}\ | = |\ (\mathbf{D} \to \mathbf{D})_\uparrow\ |$. Since by construction $X^* \subseteq^{\text{fin}} \delta$, we thus have $b \in \mathit{apply}\,\epsilon\,\delta$ as required. This completes our proof that $\mathit{apply}\,\epsilon\,\delta$ is well-defined.

We will now show that *apply* is continuous (it is easy to see that it is monotonic). By virtue of what we know about continuous functions on cpos it will be enough to show that it is continuous in each argument separately, viz. if $\mathcal{D}$ is any directed subset of $|\ \mathbf{D}\ |$ and $\epsilon \in |\ \mathbf{D}\ |$, then

$$\mathit{apply}\,\epsilon\,(\bigcup \mathcal{D}) = \bigcup_{\delta\in\mathcal{D}} \mathit{apply}\,\epsilon\,\delta \quad \text{and}$$
$$\mathit{apply}\,(\bigcup \mathcal{D})\,\epsilon = \bigcup_{\delta\in\mathcal{D}} \mathit{apply}\,\delta\,\epsilon\ .$$

In the first of these equations we know, by monotonicity, that the right hand side is a subset of the left hand side. So suppose b is an element of the left hand side. Then there is some X such that $\langle X, b\rangle \sqsubseteq \epsilon$ and $X \subseteq^{\text{fin}} \bigcup \mathcal{D}$. Hence by the usual directed set argument there is some $\delta^* \in \mathcal{D}$ with $X \subseteq \delta^*$. But then $b \in \mathit{apply}\,\epsilon\,\delta^*$, so that b is in the right hand side as required. The proof of continuity in the first argument is even easier and is omitted.

The well-definedness of $\mathcal{E}[\![E_1\ E_2]\!]$ now follows trivially by induction. Its continuity follows since, if $\Delta \subseteq U$ is directed and $\rho^* = \bigsqcup \Delta$, we have

$$\begin{aligned}
\mathcal{E}[\![E_1\ E_2]\!]\rho^* &= \mathit{apply}(\mathcal{E}[\![E_1]\!]\rho^*)(\mathcal{E}[\![E_2]\!]\rho^*) \\
&= \mathit{apply}(\bigcup_{\rho\in\Delta} \mathcal{E}[\![E_1]\!]\rho)(\bigcup_{\rho\in\Delta} \mathcal{E}[\![E_2]\!]\rho) \quad \text{by induction} \\
&= \bigcup_{\rho_1,\rho_2\in\Delta} \mathit{apply}(\mathcal{E}[\![E_1]\!]\rho_1)(\mathcal{E}[\![E_2]\!]\rho_2) \text{ continuity of } \mathit{apply} \\
&= \bigcup_{\rho\in\Delta} \mathit{apply}(\mathcal{E}[\![E_1]\!]\rho)(\mathcal{E}[\![E_2]\!]\rho) \\
&\qquad \text{by monotonicity and directedness, for if } \rho_1, \rho_2 \in \Delta, \text{ there is } \rho \in \Delta \text{ with } \mathcal{E}[\![E_1]\!]\rho_j \subseteq \mathcal{E}[\![E_1]\!]\rho, \quad j \in \{1,2\} \\
&= \bigcup_{\rho\in\Delta} \mathcal{E}[\![E_1\ E_2]\!]\rho\ .
\end{aligned}$$

We now turn to the case of $\lambda\, x.E$. Recall that $\mathcal{E}[\![\,\lambda\, x.E]\!]\rho = \iota_\uparrow(\phi(\rho))$, where

$$\phi(\rho)\ =\ \{\langle X, a\rangle \mid\ X \in \mathit{Con}\ \wedge\ a \in \mathcal{E}[\![E]\!]\rho[\overline{X}/x]\}\ .$$

In view of the well-definedness and continuity of the injection $\iota_\uparrow$ it will be enough for this clause to prove that ϕ is a well-defined and continuous map from U to $|\ \mathbf{D} \to \mathbf{D}\ |$.

Suppose that $\rho \in U$. To show that $\phi(\rho)$ is consistent it will be enough to show that, whenever $\{\langle X_1, a_1\rangle, \ldots, \langle X_n, a_n\rangle\} \subseteq \phi(\rho)$ and $X^* = \bigcup_{i=1}^n X_i \in \mathit{Con}$, we have $\{a_1, \ldots, a_n\} \in \mathit{Con}$ also. By induction we know that $\mathcal{E}[\![E]\!]$ is monotonic. Hence, since by assumption $a_i \in \mathcal{E}[\![E]\!]\rho[\overline{X_i}/x]$ for each i, we have $a_i \in \mathcal{E}[\![E]\!]\rho[\overline{X}^*/x]$ for all i. Thus $\{a_1, \ldots, a_n\} \in \mathit{Con}$ by the well-definedness (by induction) of $\mathcal{E}[\![E]\!]$.

If $\{\langle X_1, a_1\rangle, \ldots, \langle X_n, a_n\rangle\} \subseteq \phi(\rho)$ and $\{\langle X_1, a_1\rangle, \ldots, \langle X_n, a_n\rangle\} \vdash' \langle Y, b\rangle$ then (by definition of $\vdash'$) $\{a_i \mid Y \vdash X_i\} \vdash b$. But $Y \vdash X_i$ implies that $\overline{Y} \supseteq \overline{X_i}$, so as in the last paragraph we have $a_i \in \mathcal{E}[\![E]\!]\rho[\overline{Y}/x]$ for each such i by monotonicity of $\mathcal{E}[\![E]\!]$. Hence $b \in \mathcal{E}[\![E]\!]\rho[\overline{Y}/x]$ by the deductive closure (by induction) of $\mathcal{E}[\![E]\!]\rho[\overline{Y}/x]$. Thus $\langle Y, b\rangle \in \phi(\rho)$, proving that this set is deductively closed, completing the proof that ϕ is well-defined.

It is easy to show by induction that ϕ is monotonic. Now suppose that $\Delta \subseteq U$ is directed, and that $\rho^* = \bigsqcup \Delta$.

$$
\begin{aligned}
\phi(\rho^*) &= \{\langle X, a\rangle \mid X \in \mathit{Con} \wedge a \in \mathcal{E}[\![E]\!]\rho^*[\overline{X}/x]\} \\
&\qquad\qquad \text{by definition of } \phi \\
&= \{\langle X, a\rangle \mid X \in \mathit{Con} \wedge a \in \bigcup_{\rho\in\Delta} \mathcal{E}[\![E]\!]\rho[\overline{X}/x]\} \\
&\qquad\qquad \text{as } \{\rho[\overline{X}/x] \mid \rho \in \Delta\} \text{ is directed with limit} \\
&\qquad\qquad \rho^*[\overline{X}/x], \text{ and } \mathcal{E}[\![E]\!] \text{ is continuous} \\
&= \bigcup_{\rho\in\Delta} \{\langle X, a\rangle \mid X \in \mathit{Con} \wedge a \in \mathcal{E}[\![E]\!]\rho[\overline{X}/x]\} \\
&= \bigcup_{\rho\in\Delta} \phi(\rho)
\end{aligned}
$$

Hence ϕ is continuous.

This completes the proof of the $\lambda\, x.E$ case of Theorem 7.2.1. This completes the proof of the Theorem. ∎

Now that we have given the λ-calculus a semantics we must check that it has all the properties that we expect: for example we would like α- or β-conversion to preserve the semantics of a λ-expression. The following pair of lemmas show that the semantics is well-behaved in respect of free identifiers and substitution.

7.2.2 LEMMA
If $E \in \mathit{Exp}$ and $x \in \mathit{Ide}$ does not appear free in E, then

$$\mathcal{E}[\![E]\!]\rho = \mathcal{E}[\![E]\!]\rho[\delta/x]$$

for all $\rho \in U$ and $\delta \in |\, \mathbf{D}\, |$.

PROOF
This is a straightforward structural induction and is left as an exercise. ∎

7.2.3 LEMMA (THE SUBSTITUTION LEMMA)
If $E_1, E_2 \in Exp$, $x \in Ide$ and $\rho \in U$, then

$$\mathcal{E}[\![[E_1/x]E_2]\!]\rho \;=\; \mathcal{E}[\![E_2]\!]\rho[\mathcal{E}[\![E_1]\!]\rho/x] \;.$$

PROOF
This result is proved by induction on the depth of the syntax tree of E_2, for *all* E_1, x and ρ simultaneously. Specifically we will prove that the result holds for particular E_1, E_2, x and ρ given that it holds for all E_2' simpler than E_2 for *every* combination of E_1', x' and ρ'.

We use induction on the depth of syntax trees rather than structural induction for exactly the same reason as was alluded to earlier when discussing the well definedness of substitution: we sometimes have to change the names of bound identifiers.

We abbreviate $\rho[\mathcal{E}[\![E_1]\!]\rho/x]$ by ρ^*.

- If $E_2 \;=\; x$, then $[E_1/x]E_2 \;=\; E_1$, so that

$$\mathcal{E}[\![[E_1/x]E_2]\!]\rho \;=\; \mathcal{E}[\![E_1]\!]\rho \;=\; \rho^*[\![x]\!] \;=\; \mathcal{E}[\![E_2]\!]\rho^*$$

as desired.

- If $E_2 \;=\; y$ (where $y \neq x$), then $[E_1/x]E_2 \;=\; y$, so that

$$\mathcal{E}[\![[E_1/x]E_2]\!]\rho \;=\; \mathcal{E}[\![y]\!]\rho \;=\; \rho[\![y]\!] \;=\; \rho^*[\![y]\!] \;=\; \mathcal{E}[\![E_2]\!]\rho^* \;.$$

- If $E_2 \;=\; E_3\, E_4$, then $[E_1/x]E_2 \;=\; ([E_1/x]E_3)([E_1/x]E_4)$, so that

$$\begin{aligned}
\mathcal{E}[\![[E_1/x]E_2]\!]\rho \;&=\; \mathcal{E}[\![([E_1/x]E_3)([E_1/x]E_4)]\!]\rho \\
&=\; \mathit{apply}(\mathcal{E}[\![[E_1/x]E_3]\!]\rho)(\mathcal{E}[\![[E_1/x]E_4]\!]\rho) \\
&=\; \mathit{apply}(\mathcal{E}[\![E_3]\!]\rho^*)(\mathcal{E}[\![E_4]\!]\rho^*) \qquad \text{by induction} \\
&=\; \mathcal{E}[\![E_3\, E_4]\!]\rho^* \qquad \text{as required.}
\end{aligned}$$

- If $E_2 = \lambda\, x.E$ then $[E_1/x]E_2 = E_2$. The result then follows immediately from Lemma 7.2.2: $\mathcal{E}[\![E_2]\!]\rho^* = \mathcal{E}[\![E_2]\!]\rho$ because x is not free in E_2.
- If $E_2 = \lambda\, y.E$, where $x \neq y$, there are two cases to consider. The first is when $[E_1/x]E_2 = \lambda\, y.[E_1/x]E$. In this case *either* y is not free in E_1 *or* x is not free in E.

In the first of these subcases, for every $X \in Con$ we have

$$\begin{aligned}
\mathcal{E}[\![[E_1/x]E]\!]\rho[\overline{X}/y] &= \mathcal{E}[\![E]\!]\rho[\overline{X}/y][\mathcal{E}[\![E_1]\!]\rho[\overline{X}/y]/x] \quad \text{by induction} \\
&= \mathcal{E}[\![E]\!]\rho[\overline{X}/y][\mathcal{E}[\![E_1]\!]\rho/x] \quad \text{as } y \text{ is not free in } E_1 \\
&= \mathcal{E}[\![E]\!]\rho^*[\overline{X}/y] \ .
\end{aligned}$$

In the second subcase

$$\begin{aligned}
\mathcal{E}[\![[E_1/x]E]\!]\rho[\overline{X}/y] &= \mathcal{E}[\![E]\!]\rho[\overline{X}/y][\mathcal{E}[\![E_1]\!]\rho[\overline{X}/y]/x] \quad \text{by induction} \\
&= \mathcal{E}[\![E]\!]\rho^*[\overline{X}/y] \quad \begin{array}{l}\text{for, since } x \text{ is not free in } E, \text{ we} \\ \text{may alter its value in } \rho \text{ as we} \\ \text{please.}\end{array}
\end{aligned}$$

Hence in either subcase

$$\begin{aligned}
\mathcal{E}[\![[E_1/x]E_2]\!]\rho &= \mathcal{E}[\![\,\lambda\, y.[E_1/x]E]\!]\rho \\
&= \iota_\uparrow(\{\langle X, a\rangle \mid X \in Con \ \wedge \ a \in \mathcal{E}[\![[E_1/x]E]\!]\rho[\overline{X}/y]\}) \\
&= \iota_\uparrow(\{\langle X, a\rangle \mid X \in Con \ \wedge \ a \in \mathcal{E}[\![E]\!]\rho^*[\overline{X}/y]\}) \\
&= \mathcal{E}[\![\,\lambda\, y.E]\!]\rho^* \quad \text{as desired.}
\end{aligned}$$

The other case is when $[E_1/x]E_2 \ = \ \lambda\, z.[E_1/x][z/y]E$, where z is an identifier chosen so that it appears neither in $\lambda\, x.E$ nor E_1. Whenever $X \in Con$, we then have

$$\begin{aligned}
&\mathcal{E}[\![[E_1/x][z/y]E]\!]\rho[\overline{X}/z] \\
&\quad = \mathcal{E}[\![[z/y]E]\!]\rho[\overline{X}/z][(\mathcal{E}[\![E_1]\!]\rho[\overline{X}/z])/x] && \text{by induction} \\
&\quad = \mathcal{E}[\![[z/y]E]\!]\rho[\overline{X}/z][\mathcal{E}[\![E_1]\!]\rho/x] && z \text{ not free in } E_1 \\
&\quad = \mathcal{E}[\![[z/y]E]\!]\rho^*[\overline{X}/z] && \text{as } z \neq x \\
&\quad = \mathcal{E}[\![E]\!]\rho^*[\overline{X}/z][(\mathcal{E}[\![z]\!]\rho^*[\overline{X}/z])/y] && \text{by induction} \\
&\quad = \mathcal{E}[\![E]\!]\rho^*[\overline{X}/z][\overline{X}/y] \\
&\quad = \mathcal{E}[\![E]\!]\rho^*[\overline{X}/y] && z \text{ not free in } E.
\end{aligned}$$

Hence

$$\begin{aligned}\mathcal{E}[\![[E_1/x]E_2]\!]\rho &= \mathcal{E}[\![\,\lambda\, z.[E_1/x][z/y]E]\!]\rho \\ &= \iota_\uparrow(\{\langle X, a\rangle \mid X \in Con \wedge a \in \mathcal{E}[\![[E_1/x][z/y]E]\!]\rho[\overline{X}/z]\}) \\ &= \iota_\uparrow(\{\langle X, a\rangle \mid X \in Con \wedge a \in \mathcal{E}[\![E]\!]\rho^*[\overline{X}/y]\}) \\ &= \mathcal{E}[\![\,\lambda\, y.E]\!]\rho^* \quad \text{as desired.}\end{aligned}$$

This completes the proof of Lemma 7.2.3. ∎

Lemma 7.2.3 is very important, for it shows that the effect of syntactic substitution (as practiced in the conversion rules) coincides exactly with the technique of substituting semantic values into the environment which was used in defining the semantics of lambda-abstraction. It can be regarded as a justification for our rules for syntactic substitution.

The following Theorem, which shows that conversion preserves semantics, establishes that our semantics does provide a respectable model for our version of the λ-calculus. Formally it shows that the conversion rules are *sound* with respect to our semantics.

7.2.4 THEOREM
If $E_1, E_2 \in Exp$ are such that E_1 $\underline{\text{cnv}}$ E_2, then $\mathcal{E}[\![E_1]\!] = \mathcal{E}[\![E_2]\!]$.

PROOF
Our semantics is denotational, in that the semantic value of a compound expression is determined from the values of its parts. Hence if E is a compound expression, one of whose syntactic subcomponents is E', we will not change the semantic value of E if we replace E' by an expression E'' such that $\mathcal{E}[\![E']\!] = \mathcal{E}[\![E'']\!]$.

This observation means that in order to prove Theorem 7.2.4 it will be enough to prove the validity of α- and β-conversion separately, for recall that E_1 $\underline{\text{cnv}}$ E_2 if and only if there exist $E_1^* = E_1, E_2^*, \ldots, E_n^* = E_2$ such that each E_{r+1}^* $(1 \leq r \leq n-1)$ is obtained from E_r^* by α- or β-conversion either at the top level or of some subcomponent.

α-conversion We must show that, if y is not free in E, then for all $\rho \in U$

$$\mathcal{E}[\![\,\lambda\, x.E]\!]\rho \;=\; \mathcal{E}[\![\,\lambda\, y.[y/x]E]\!]\rho\,.$$

Whenever $X \in Con$ we have

$$\begin{aligned}\mathcal{E}[\![[y/x]E]\!]\rho[\overline{X}/y] &= \mathcal{E}[\![E]\!]\rho[\overline{X}/y][(\mathcal{E}[\![y]\!]\rho[\overline{X}/y])/x] \quad \text{by Lemma 7.2.3}\\ &= \mathcal{E}[\![E]\!]\rho[\overline{X}/y][\overline{X}/x]\\ &= \mathcal{E}[\![E]\!]\rho[\overline{X}/x] \quad \text{as } y \text{ is not free in } E.\end{aligned}$$

Hence

$$\begin{aligned}\mathcal{E}[\![\,\lambda\, y.[y/x]E]\!]\rho &= \iota_{\uparrow}(\{\langle X, a\rangle \mid X \in Con \wedge a \in \mathcal{E}[\![[y/x]E]\!]\rho[\overline{X}/y]\})\\ &= \iota_{\uparrow}(\{\langle X, a\rangle \mid X \in Con \wedge a \in \mathcal{E}[\![E]\!]\rho[\overline{X}/x]\})\\ &= \mathcal{E}[\![\,\lambda\, x.E]\!]\rho \quad \text{as desired.}\end{aligned}$$

β-conversion We must show that, for all $E, E_1 \in Exp$, $x \in Ide$ and $\rho \in U$,

$$\mathcal{E}[\![(\lambda\, x.E)E_1]\!]\rho \;=\; \mathcal{E}[\![[E_1/x]E]\!]\rho\,.$$

This is done as follows:

$$\begin{aligned}\mathcal{E}[\![(\lambda\, x.E)E_1]\!]\rho &= \{a \mid \exists\, X \subseteq^{\text{fin}} \mathcal{E}[\![E_1]\!]\rho.\ a \in \mathcal{E}[\![E]\!]\rho[\overline{X}/x]\}\\ &= \bigcup\{\mathcal{E}[\![E]\!]\rho[\overline{X}/x] \mid X \subseteq^{\text{fin}} \mathcal{E}[\![E_1]\!]\rho\}\\ &= \mathcal{E}[\![E]\!]\rho[\mathcal{E}[\![E_1]\!]\rho/x] \quad \text{as } \{\rho[\overline{X}/x] \mid X \subseteq^{\text{fin}} \mathcal{E}[\![E_1]\!]\rho\} \text{ is directed with limit } \rho[\mathcal{E}[\![E_1]\!]\rho/x], \text{ and } \mathcal{E}[\![E]\!] \text{ is continuous by Theorem 7.2.1}\\ &= \mathcal{E}[\![[E_1/x]E]\!]\rho \quad \text{by Lemma 7.2.3.}\end{aligned}$$

This completes the proof of Theorem 7.2.4. ∎

It is perhaps worth noting that not only does that this Theorem allow us to identify many pairs of λ-expressions which are identified by $\mathcal{E}$, but it also helps to show that many pairs are distinct. The validity of β-conversion means that a λ-abstraction really does 'mean' the function which it seeks to represent.

7.2.5 THEOREM
If E_1 and E_2 are two λ-expressions of the form

$$\lambda\, x_1.\, \lambda\, x_2. \ldots .\, \lambda\, x_n.E$$

where E is a λ-expression with no abstractions, then E_1 and E_2 are semantically equivalent if and only if they are α-convertible.

PROOF
We first prove this result for the case when both E_1 and E_2 are Church numerals (expressions of the form $\lambda f.\,\lambda x.f(f(\ldots f(fx)\ldots))$). Clearly any pair of non-α-convertible numerals can assumed to be of the form

$$E_1 \quad = \quad \lambda f.\,\lambda x.f(f(\ldots f(fx)\ldots)) \qquad (n\ f\text{'s}), \text{ and}$$

$$E_2 \quad = \quad \lambda f.\,\lambda x.f(f(\ldots f(fx)\ldots)) \qquad (m\ f\text{'s})$$

where $n < m$.

Assume for a contradiction that $\mathcal{E}[\![E_1]\!] = \mathcal{E}[\![E_2]\!]$. We then know that $\mathcal{E}[\![E_1\,E\,E^*]\!]\rho = \mathcal{E}[\![E_2\,E\,E^*]\!]\rho$ for all λ-expressions E, E^* and environments ρ. Now we know that the domain $\mid \mathbf{D} \mid$ contains the object $\emptyset$ which is distinct from any function. Let ρ be any environment such that $\rho[\![y]\!] = \emptyset$, and let

$$E \quad = \quad \lambda x.xy \quad \text{and}$$

$$E^* \quad = \quad \lambda z_1.\,\lambda z_2.\ldots.\lambda z_m.z_m\ .$$

It is not very hard (using repeated β-reduction) to show that

$$E_1\,E\,E^* \ \underline{\text{cnv}}\ \lambda z_{n+1}.\ldots.\lambda z_m.z_m$$

and $E_2 E\,E^*\ cnv\ y$.

But $\mathcal{E}[\![y]\!]\rho \neq \mathcal{E}[\![\lambda z_{n+1}.\ldots.\lambda z_m.z_m]\!]\rho$ (the former is $\emptyset$, the latter is a function), contradicting our assumption that $\mathcal{E}[\![E_1]\!] = \mathcal{E}[\![E_2]\!]$.

Next we give the proof for expressions with no λ-abstractions – in other words just combinations of identifiers. Given two such combinations we must find an environment which distinguishes them. This is not as easy as it looks, particularly as a single identifier can appear arbitrarily often in one expression. What we will do is take advantage of the result proved above by associating each distinct expression with a different Church numeral. The technique of associating expressions with numbers is known as *Gödel numbering*, after its use in the proof of Gödel's famous incompleteness theorem. In our case one can do it by associating a different prime $g(x)$ (other than 2 and 3) with each $x \in Ide$. (If for some reason there are more than countably many identifiers one cannot do this, but to tell two expressions E_1 and E_2 apart – which is all we are trying to do – it is only necessary to associate a different prime with each of the finitely many identifiers in them.) The function g is then extended to the set of all λ-free expressions by setting

$$g(E_1 E_2) = 2^{g(E_1)} \times 3^{g(E_2)}\ .$$

Because of the unique factorisation of natural numbers, the exact syntax of any such λ-expression can be recovered from its Gödel number. It will thus be enough to prove this case of the theorem if we can find an expression E and an environment ρ such that the semantic value of $E\,E'$ is, for any λ-free E', the same as the Church numeral corresponding to E's Gödel number. For since the Gödel numbers of all expressions are different, it follows from the fact that all Church numerals have different values that all E' have different values.

To do this we will find a series of combinators $E(n)$, and a specific one X such that $X\,E(n)\,\underline{\text{cnv}}\,\underline{n}$ (the nth Church numeral) for all n and $E(n)\,E(m)\,\underline{\text{cnv}}\,E(2^n3^m)$ for all n, m. For then we can choose ρ to be the environment where each x is sent to the semantic value of $E(g(x))$, and it is clear that the value of E' in ρ will be that of the $E(g(E'))$. Then the value of $E\,E'$ will be the same as the $g(E')$th Church numeral, as required.

The precise function $f(n,m) = 2^n3^m$ is largely irrelevant here; we know from the discussion of Church numerals in the last section that there is a λ expression F such that

$$F\,\underline{n}\,\underline{m} \;=\; \underline{2^n3^m}$$

for all n, m. What in fact we will do is construct expressions E and $E^F(n)$ such that, if f is any binary function on $\mathbb{N}$ represented by the function F on Church numerals, then $E\,E^F(n)\,\underline{\text{cnv}}\,\underline{n}$ and

$$E^F(n)\,E^F(m)\,\underline{\text{cnv}}\,E^F(f\;n\;m)\,.$$

One pair of definitions that works is

$$E^F(n) = \lambda\,y.y\;C^F\,\underline{n} \qquad \text{and} \qquad X = \lambda\,x.x(\lambda\,y.\,\lambda\,z.z)$$

where

$$C^F = \lambda\,x.\,\lambda\,y.\,\lambda\,z.\,\lambda\,t.t\;x\;(F\;z\;y)\,.$$

To check this is just routine calculation. (Actually there is nothing in the definition of the $E^F(n)$ which relies on $\underline{n}$ being a Church numeral or F being a function of them. In fact these combinators show that it is possible to 'calculate' an arbitrary binary function of λ-expressions by application.)

Thus we have proved our result for λ-free expressions. It is easy now to complete the proof. First we observe that no two expressions with *different* numbers of leading λs can ever be be equivalent, for if $n < m$ and E_1, E_2 are λ-free then for any identifier x

$$\mathcal{E}[\![(\lambda\,x_1.\ldots.\lambda\,x_n.E_1)\;x\;\ldots x]\!]\rho_\perp = \emptyset$$

while

$$\mathcal{E}[\![(\lambda\, y_1.\ldots.\lambda\, y_m.E_2)\, x\ \ldots\ x]\!]\rho_\perp \neq \emptyset$$

where in each case there are n xs and $\rho_\perp$ is the environment mapping all identifiers to $\emptyset$.

Secondly, if $\lambda\, x_1.\ldots.\lambda\, x_n.E_1$ and $\lambda\, y_1.\ldots.\lambda\, y_n.E_2$ are two expressions with the same number of λs, and $z_1, \ldots, z_n$ are distinct identifiers not appearing in either, then it is clear that the two expressions are α-convertible if and only if the natural β-reductions of

$$(\lambda\, x_1.\ldots.\lambda\, x_n.E_1)\, z_1\ \ldots\ z_n \quad \text{and} \quad (\lambda\, y_1.\ldots.\lambda\, y_n.E_2)\, z_1\ \ldots\ z_n.$$

are the same. But we know by the result about λ-free expressions that if these expressions are different (i.e., the original expressions are not α-convertible) then they, and hence the original expressions, have different values. ∎

Unfortunately Theorem 7.2.5 cannot be strengthened to show that *every* pair of distinct normal forms are semantically distinct (see Exercises). This can be regarded as a disadvantage of our non-standard semantics of the λ-calculus, for a full-strength theorem can be proved in the classical models with η-conversion.

Remember that there are λ-expressions Y^* with the property that, if E is any λ-expression, then Y^*E $\underline{\text{cnv}}$ $E(Y^*E)$. Such expressions thus appear to construct fixed points of the functions represented by arbitrary λ-expressions. Theorem 7.2.4 tells us that this really is the case, in that

$$\mathcal{E}[\![\, Y^*E]\!]\rho \;=\; \mathit{apply}(\mathcal{E}[\![E]\!]\rho)(\mathcal{E}[\![\, Y^*E]\!]\rho)$$

for all such Y^* and any $E \in \mathit{Exp}$ and $\rho \in \mathit{Env}$. Thus $\mathcal{E}[\![\, Y^*E]\!]\rho$ is always a fixed point of the function $\mathit{apply}(\mathcal{E}[\![E]\!]\rho)$. It might initially be surprising to realise that the semantic value of every λ-expression has some fixed point, until we remember that approximable mappings represent continuous functions, and that continuous functions on cpos such as $|\,\mathbf{D}\,|$ always have fixed points. Nevertheless, fixed point operators are one of the most intriguing features of the λ-calculus.

We will consider one fixed point combinator, the syntactically simplest one, in detail. Remember that we defined

$$Y \;=\; \lambda f.(\lambda\, x.f(xx))(\lambda\, x.f(xx))\ .$$

It is elementary to show that Yf $\underline{\text{cnv}}$ $f(Yf)$, so that Y is indeed a fixed point combinator. It is interesting to ask just which fixed point of a given λ-expression Y picks out, since often a function will have many fixed points (for example $\lambda\, x.x$ will fix every point). An answer to this question will be important when we use a

Y-like construct to imitate recursion. (Remember that we did this in our Lisp-like language, when we showed that recursive *define* was not necessary when defining objects recursively.)

If $\phi :| \mathbf{D} |\rightarrow| \mathbf{D} |$ is a continuous function, the fixed point $fix\phi$ we already know about is given by

$$fix\phi \;=\; \bigcup_{n=0}^{\infty} \phi^n(\emptyset)\ .$$

Remember that $fix\phi \subseteq z$ for all fixed points z of ϕ; it is the *least* fixed point. If we think of $fix\phi$ for a moment as a set of tokens, we see that it is the set of all tokens that can be finitely demonstrated to be in any fixed point z of ϕ: clearly $\emptyset \subseteq z$, and if $a \in \phi^n(\emptyset)$ then by continuity there is some finite subset X of $\phi^n(\emptyset)$ such that $a \in \phi(\overline{X})$. *fix* is thus a fixed point operator that ties in particularly well with the idea that a token is something which is finitely observable, for it is possible to 'observe' each of the tokens in $z = fix\phi$ simply by postulating that z is a fixed point of ϕ.

In fact it will turn out that Y produces exactly the same fixed point as *fix*. This is a very fortunate result, for it shows that the natural semantic method of finding fixed points coincides with the most natural syntactic method. Before we can proceed to the proof of this interesting result it is necessary to analyse our semantic domain in a little more detail than we have done hitherto.

Definition
Recall that our domain $\mathbf{D} = \langle A, Con, \vdash \rangle$ is the least upper bound (under $\trianglelefteq$) of the chain of information systems $\mathbf{D}_n = \langle A_n, Con_n, \vdash_n \rangle$, where

$$\begin{aligned} \mathbf{D}_0 &= \bot, \quad \text{and} \\ \mathbf{D}_{n+1} &= (\mathbf{D}_n \rightarrow \mathbf{D}_n)_{\uparrow} \end{aligned}$$

Whenever $\epsilon \in| \mathbf{D} |$ and $n \in \mathbb{N}$, define

$$\epsilon{\downarrow_n} = \overline{\epsilon \cap A_n}$$

where deductive closure is taken in $\mathbf{D}$. $\epsilon{\downarrow_n}$ is just the canonical projection of ϵ into the natural embedding of the domain $| \mathbf{D}_n |$ within $| \mathbf{D} |$; it is just the set of all information deducible about ϵ from its level-n tokens. Observe that $\langle \epsilon{\downarrow_n} \mid \ n \in \mathbb{N} \rangle$ is a chain in $| \mathbf{D} |$ with union ϵ and that $\epsilon{\downarrow_0} = \emptyset$. ∎

Each level-$n+1$ token (other than the minimal *lifted* token, which conveys no functional information) has the form $\langle old, \langle X, a \rangle \rangle$, where X is a consistent set of level-n tokens and a is a level-n token. It means 'from the information X about the argument, deduce a about the result'. Thus, level-$n+1$ tokens tell us how level-n tokens relate under a function, but convey no information about how the others

relate: in particular they tell us nothing about the relationships of those level-$n{+}1$ tokens that are not also level-n tokens. This is the intuition behind the following result.

7.2.6 LEMMA
If $\epsilon, \delta \in |\, \mathbf{D} \,|$ and $n \in \mathbb{N}$, then $apply\,\epsilon{\downarrow_{n+1}}\ \delta\ =\ (apply\,\epsilon{\downarrow_{n+1}}\ \delta{\downarrow_n}){\downarrow_n}$.

PROOF
Clearly the right hand side above is a subset of the left hand side. Suppose, then, that a is an element of the left hand side. This means that there is some $X \subseteq^{\text{fin}} \delta$ such that $\langle X, a\rangle \sqsubseteq \epsilon{\downarrow_{n+1}}$.

By definition of $\epsilon{\downarrow_{n+1}}$ it is possible to find $\langle X_1, b_1\rangle, \ldots, \langle X_k, b_k\rangle \sqsubseteq \epsilon \cap A_{n+1}$ such that $\{\langle X_1, b_1\rangle, \ldots, \langle X_k, b_k\rangle\} \vdash' \langle X, a\rangle$, or in other words, such that $\{b_i \mid X \vdash X_i\} \vdash a$.

Suppose i is such that $X \vdash X_i$. Then $X_i \subseteq \delta$ as $X \subseteq \delta$ and δ is deductively closed. Since $\langle X_i, b_i\rangle \in A_{n+1}$ we know that $(X_i \cup \{b_i\}) \subseteq A_n$. Hence $X_i \subseteq \delta{\downarrow_n}$, and so $b_i \in (apply\,\epsilon{\downarrow_{n+1}}\ \delta{\downarrow_n}) \cap A_n$.

But $\{b_i \mid X \vdash X_i\} \vdash a$, so we now have $a \in (apply\,\epsilon{\downarrow_{n+1}}\ \delta{\downarrow_n}){\downarrow_n}$ by definition of $\downarrow_n$, as required. Hence the left hand side in the statement of this lemma is a subset of the right hand side, completing its proof. ∎

We are now in a position to prove our result about Y. The above Lemma, while it is crucial in this proof, is actually of much wider interest.

7.2.7 THEOREM
Suppose that $\delta \in |\, \mathbf{D} \,|$ represents the function $\phi :|\, \mathbf{D} \,| \to |\, \mathbf{D} \,|$ (i.e. $\phi = apply\,\delta$). Then, for each $\rho \in U$,

$$\mathcal{E}[\![\, Yf]\!]\rho[\delta/f]\ =\ \bigcup_{n=0}^{\infty} \phi^n(\emptyset)\quad (=\ fix\phi)\ .$$

PROOF
For brevity denote $\mathcal{E}[\![\, Yf]\!]\rho[\delta/f]$ by x, $fix\phi$ by y, and $\rho[\delta/f]$ by ρ'.

It is a straightforward consequence of the validity of β-conversion that

$$x\ =\ apply\delta x\ =\ \phi(x)\ ,$$

so x is *a* fixed point of ϕ. Hence $y \subseteq x$, for y is the *least* fixed point of ϕ. It will therefore suffice to prove that $x \subseteq y$.

Now $x = apply\,\epsilon\,\epsilon$, where $\epsilon = \mathcal{E}[\![\, \lambda x.f(xx)]\!]\rho'$. Because *apply* is continuous

we know that

$$apply\,\epsilon\,\epsilon \;=\; \bigcup_{n=0}^{\infty} apply\,\epsilon{\downarrow_n}\;\epsilon\;.$$

Claim that, for each n, $apply\,\epsilon{\downarrow_n}\;\epsilon \subseteq \phi^n(\emptyset)$. This is trivially true when $n = 0$, for $apply\,\emptyset\,\epsilon = \emptyset$. Assume that it holds for n. Then

$$\begin{array}{lll}
apply\,\epsilon{\downarrow_{n+1}}\;\epsilon = (apply\,\epsilon{\downarrow_{n+1}}\;\epsilon{\downarrow_n})\,{\downarrow_n} & & \text{by Lemma 7.2.6} \\
\subseteq apply\,\epsilon\,\epsilon{\downarrow_n} & & \text{monotonicity of } apply \\
\subseteq apply(\mathcal{E}[\![\,\lambda\,x.f(xx)]\!]\rho')\,\epsilon{\downarrow_n} & & \text{by definition of } \epsilon \\
\subseteq \mathcal{E}[\![f(xx)]\!]\rho'[\epsilon{\downarrow_n}\;/x] & & \text{as } \beta\text{-conversion is valid} \\
\subseteq apply\,\delta\,(apply\,\epsilon{\downarrow_n}\;\epsilon{\downarrow_n}) & & \\
\subseteq \phi(apply\,\epsilon{\downarrow_n}\;\epsilon) & & \text{by monotonicity and } \phi = apply\,\delta \\
\subseteq \phi(\phi^n(\emptyset)) \;=\; \phi^{n+1}(\emptyset) & & \text{by induction, as required.}
\end{array}$$

The result follows immediately. ∎

An immediate corollary to this result is that $E = (\lambda\,x.xx)(\lambda\,x.xx)$, the simplest 'looping' λ-expression, is mapped by $\mathcal{E}$ to $\emptyset$ (in any environment), for $Y(\lambda\,x.x)$ $\underline{\text{cnv}}$ E, and clearly the least fixed point of the identity function is $\emptyset$. This is a pleasing result, for it is precisely the value we wanted for a λ-expression that could never show that it was a function. In fact *all* λ-expressions that cannot show themselves to be functions (or potential functions in a simple way) map to $\emptyset$, but this is a much harder result and is delayed to the next section (Theorem 7.3.?).

We motivated the model $\mathbf{D} = (\mathbf{D} \to \mathbf{D})_{\uparrow}$ by the idea that a λ-expression should not be considered a function unless it could show itself to be one. The clearest definition of λ-expression E 'showing itself to be a function' is for it to be possible to convert E into $\lambda\,x.E'$ for some x and E' – for in this form E demonstrably takes an argument. We already know, by the semantic validity of the conversion rules, that in this case $\mathcal{E}[\![E]\!]\rho = \mathcal{E}[\![\,\lambda\,x.E']\!]\rho$ which is in the lifted part of $|\,\mathbf{D}\,|$.

Thus, expressions convertible to abstractions are *never* mapped to $\emptyset$. There is another class which is *sometimes* (i.e., for some environments ρ) not mapped to $\emptyset$. These are ones which are convertible to something of the form

$$x\,E_1 E_2 \ldots E_n$$

(i.e., a free identifier applied to any list of expressions). For in any environment

where x is mapped to the value of an expression of the form

$$\lambda\, y_1 . \ldots . \lambda\, y_{n+1} . E' \ ,$$

the original expression is equivalent to an abstraction (since x is not supplied with enough arguments to use up all its λs).

These two forms into which we might convert an expression are conveniently grouped together under the following definition.

DEFINITION
Say that a λ-expression is in *weak head normal form* (whnf) if it has either of the forms

$$\lambda\, x . E$$

$$x\, E_1 E_2 \ldots E_n$$

or in other words, if it is a λ-abstraction or the application of a free identifier to a number of λ-expressions. ∎

No λ-expression that is convertible to whnf is semantically equivalent to the looping expression $(\lambda\, x . xx)(\lambda\, x . xx)$. On the other hand, we might reasonably hope that a λ-expression that *cannot* be converted to one in whnf *is* equivalent to $(\lambda\, x . xx)(\lambda\, x . xx)$, since it neither shows that it can take an argument directly nor gives us the opportunity to make a substitution for a free identifier as above. This converse is in fact true, providing perhaps the clearest demonstration that we have lived up to earlier aspirations.

The following result, which establishes this, requires a rather difficult proof. The best way of presenting it is as a corollary to a general congruence between our denotational semantics and an operational semantics. We will do this later, in Chapter ???11. Among other corollaries will be the result that not only Y, but *all* fixed point combinators, are semantically equivalent to *fix*.

7.2.8 THEOREM
If $E \in Exp$, then $\mathcal{E}[\![E]\!]\rho = \emptyset$ for all $\rho \in U$ if and only if E cannot be reduced to whnf. ∎

Exercises

7.2.1 Prove Lemma 3.2: that if x is not free in the λ-expression E then $\mathcal{E}[\![E]\!]\rho = \mathcal{E}[\![E]\!]\rho[\delta/x]$ for all $\rho \in U$ and $\delta \in |\, \mathbf{D}\, |$.

*7.2.2 Find a pair of λ-expressions E_1 and E_2 which have distinct (i.e. it is impossible to get from one to the other by α-conversion) normal forms, but such that $\mathcal{E}[\![E_1]\!] = \mathcal{E}[\![E_2]\!]$.
[Hint: such a pair of normal forms are likely to be η-convertible.]

*7.2.3 Prove that $\mathcal{E}[\![Y^E]\!] = \mathcal{E}[\![Y]\!]$, where Y^E is as defined in Exercise 7.1.5.
[Hint: adapt the proof of Theorem 7.3.7 to prove a corresponding result about Y^E.]

7.2.4 Recall the combinator $G = \lambda y.\, \lambda f.f(y\,f)$ defined in Exercise 7.1.4. Show that $G\,X$ $\underline{\text{cnv}}$ X for a λ-expression X if and only if $X\,f$ $\underline{\text{cnv}}$ f (i.e., X is a fixed point combinator). Deduce that $Y\,G$ is semantically equivalent to Y, and hence that all the Y_n defined in Exercise 7.1.4 are semantically equivalent.

7.2.5 In Section 5.2 we showed how, by using a nonstandard function space constructor $\mapsto$, we could create nontrivial domains that were isomorphic to their function spaces. Indeed, since in general $\mathbf{A} \mapsto \mathbf{B} \equiv \mathbf{A} \to \mathbf{B}$ we know that, given a solution to $\mathbf{A} = \mathbf{A} \mapsto \mathbf{A}$ there is a natural information system isomorphism ϕ from $\mathbf{A} \to \mathbf{A}$ to $\mathbf{A}$. ($\phi\langle X, a\rangle = \langle X, a\rangle$ unless $X = \emptyset$ and a is not an ordered pair, in which case $\phi\langle X, a\rangle = a$.)
One can give a 'classical' semantics $\mathcal{E}'$ to the λ calculus in such a domain, where one no longer lifts a λ-abstraction but just represents it as a pure function. One clause is

$$\mathcal{E}'[\![\lambda x.E]\!]\rho = \{\phi\langle X, a\rangle \mid a \in \mathcal{E}'[\![E]\!]\rho[\overline{X}/x]\}\,.$$

(a) Complete this by giving the clauses for application and a single identifier. Prove that η-conversion is valid in this semantics. (A much more considerable task, should the reader want one, is to re-prove the substitution lemma and hence get that α- and β-conversion are still valid.)
(b) Prove an analogue of Lemma 7.2.6 for this semantics and hence, assuming that all conversion rules are valid, prove an analogue of Theorem 7.2.7. (Be careful with the base case of the main induction.)

7.2.6 Show that, if $D_n \subseteq \rho[\![y]\!]$ (where D_n is the token set of the nth canonical approximation to the solution of

$$\mathbf{D} = (\mathbf{D} \to \mathbf{D})_\uparrow$$

then $D_{n+1} \subseteq \mathcal{E}[\![\lambda x.y]\!]\rho$. By using this fact and Theorem 7.2.7, construct a λ-expression E such that

$$\mathcal{E}[\![E]\!]\rho' = D \qquad \text{for all environments } \rho',$$

where D is the set of tokens of $\mathbf{D}$.
What is the value of your expression E under the semantic function $\mathcal{E}'$ of Exercise 7.2.5?