Lund University
Computer Science
Lennart Andersson

2009-05-04

Programming Language Theory
Lecture notes

7 Domain theory

Recursion and recursive definitions are abundant in programming and computer science. A recursive definition is in some sense circular; we are defining something in terms of itself. It is far from obvious that such definitions are meaningful, and some of them lead to paradoxes.

An old paradox is the Liars paradox: A Cretensian said: "All Cretensians are liars." Assuming that a liar lies all the time, this statement can be neither true nor false.

Another famous paradox is Bertrand Russel's paradox in intuitive set theory. In this theory there is no restriction on the elements of a set. A set has (zero or more) elements and an element may belong to a set. A set may be a member of a set and it is reasonable to talk about the set of all sets, U. A set belonging to itself is a questionable option, so let us consider the set of all sets not belonging to itself. Formally $M = \{ S \in U \mid S \notin S \}$. Now, does $M \in M$?

Assume that $M \in M$. By the definition of M we conclude that M does not belong to M, i.e. $M \notin M$. This is a contradiction.

Instead, assume that $M \notin M$. It follows that M must be an element of M, i.e. $M \in M$. Again we have a contradiction. This shocked the mathematics community in the early 20th century.

We are familiar with recursive definitions of functions, e.g. the factorial function, $\mathrm{fac} \in \mathbb{N} \to \mathbb{N}$

$$\mathrm{fac}(n) \triangleq \text{ if } n = 0 \text{ then } 1 \text{ else } n * \mathrm{fac}(n-1)$$

or by cases

$$\mathrm{fac}(0) \triangleq 1$$
$$\mathrm{fac}(n) \triangleq n * \mathrm{fac}(n-1), \qquad n > 0$$

We are used to interpret this definition operationally and evaluation for a given argument can be done by repeatedly replacing the left member of the second definition with the right member until finally we can use the first definition. This example follows a simple pattern called *primitive recursion* and there is no problem with termination.

There are other recursive function definitions that are more problematic. Some examples.

$$\mathrm{f}_0(0) \triangleq 1$$
$$\mathrm{f}_0(n) \triangleq \mathrm{f}_0(n+1)/(n+1), \qquad n \geq 0$$

The operational interpretation will not terminate, and there are two definitions of $\mathrm{f}_0(0)$, but they don't seem to be contradictory.

$$\mathrm{f}_1(n) \triangleq \mathrm{f}_1(n)$$

This is a very simple recursive definition, but what is its the meaning? And what about

$$\mathrm{f}_2(0) \triangleq 0$$
$$\mathrm{f}_2(n) \triangleq \mathrm{f}_2(n+1), \qquad n > 0$$

and

$$\mathrm{f}_3(n) \stackrel{\Delta}{=} \mathrm{f}_3(n) + 1 \quad ?$$

Another interesting example is the *Ackerman function*, $\mathrm{Ack} \in \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

$$\begin{aligned}
\mathrm{Ack}(n, 0) &= n + 1 \\
\mathrm{Ack}(0, m + 1) &= \mathrm{Ack}(1, m) \\
\mathrm{Ack}(n + 1, m + 1) &= \mathrm{Ack}(\mathrm{Ack}(n, m + 1), m)
\end{aligned}$$

It is recursive in an intricate way and it is not obvious that the operational interpretation will terminate. If you try to compute $\mathrm{Ack}(5, 5)$ you may loose your patience.

In the operational semantics for **While** we defined the semantic function $\mathcal{S}_{ns} \in \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$. In the *denotational* semantics for **While** we define the semantics by such semantic functions directly. The type of the semantic function will be $\mathcal{S}_{ds} \in \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$, where *ds* indicates that we are using denoational semantics.

The first three axioms are easy. We just have to define how to compute the final state for a given statement and an initial state.

$$\begin{aligned}
\mathcal{S}_{ds}[\![x := a]\!]\sigma &= \sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma] \\
\mathcal{S}_{ds}[\![\texttt{skip}]\!]\sigma &= \sigma \\
\mathcal{S}_{ds}[\![S_1; S_2]\!]\sigma &= \mathcal{S}_{ds}[\![S_2]\!](\mathcal{S}_{ds}[\![S_1]\!]\sigma)
\end{aligned}$$

In the last axiom we must first compute the state after execution of the first statement and then compute the final state using that state and the second statement. Using the composition operator for functions the last axiom could be written as

$$\mathcal{S}_{ds}[\![S_1; S_2]\!] = (\mathcal{S}_{ds}[\![S_2]\!]) \circ (\mathcal{S}_{ds}[\![S_1]\!])$$

For the if statement we borrow the syntax for a conditional expression from C and Java.

$$\mathcal{S}_{ds}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!]\sigma = \mathcal{B}[\![b]\!]\sigma \;\texttt{?}\; \mathcal{S}_{ds}[\![S_1]\!]\sigma \;\texttt{:}\; \mathcal{S}_{ds}[\![S_2]\!]\sigma$$

For the `while` statement we are inspired by the semantic equivalence of `while` b `do` S and `if` b `then` $(S;$`while` b `do` $S)$ `else skip`.

$$\mathcal{S}_{ds}[\![\texttt{while } b \texttt{ do } S]\!]\sigma = \mathcal{B}[\![b]\!]\sigma \;\texttt{?}\; \mathcal{S}_{ds}[\![S;\ \texttt{while } b \texttt{ do } S]\!]\sigma \;\texttt{:}\; \sigma$$

Writing W for $\mathcal{S}_{ds}[\![\texttt{while } b \texttt{ do } S]\!]$ we thus have

$$W\sigma = \mathcal{B}[\![b]\!]\sigma \;\texttt{?}\; W(\mathcal{S}_{ds}[\![S]\!]\sigma) \;\texttt{:}\; \sigma$$

or

$$W = \lambda\sigma \,.\, (\mathcal{B}[\![b]\!]\sigma \;\texttt{?}\; W(\mathcal{S}_{ds}[\![S]\!]\sigma) \;\texttt{:}\; \sigma)$$

This is a recursive definition of W and we are not sure that the recursion will terminate; in fact we know that for some choices of b, S and σ it will not. Is such a recursive definition meaningful?

In this chapter we will investigate when such recursive definitions are meaningful and what their meanings are.

7.1 Ordered sets

A *partially ordered set* or shorter a *poset* is a set with some *order relation.* Some familiar examples are the set of natural numbers and the $\leq$ relation and the set of all subsets of N and the subset relation $\subseteq$. For the first example all elements in N are related; if $m, n \in \mathbb{N}$ then either $m \leq n$ or $n \leq m$ (or both). Such a poset is called a *totally ordered set.* For the second example some sets are ordered with respect to each other while others are not, e.g neither $\{ 0, 1 \} \subseteq \{ 1, 2 \}$ nor $\{ 1, 2 \} \subseteq \{ 0, 1 \}$. We will use the symbol $\sqsubseteq$ as a generic order symbol. In some of our applications it will order elements with partial information and even if we will carelessly read it as “is less than”. It should rather be read as “is less defined than or equal to”.

We have carefully defined what a function $f \in A \to B$ is; it is a subset of $A \times B$ where there is exactly one pair (a, b) for each $a \in A$. $A \times B$ is the set of all pairs with a first component from A and a second component from B. As an example, the logical not function is the set $\{ (\mathbf{tt}, \mathbf{ff}), (\mathbf{ff}, \mathbf{tt}) \}$.

A *(binary) relation from A to B* is a subset of $A \times B$. Thus $\leq$ on $\{ 0, 1, 2 \} \times \{ 0, 1, 2 \}$ is the set $\{ (0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 2) \}$. If $A = B$ we call it a *(binary) relation on A.*

In general, an *order relation on* a set D is a relation, $\sqsubseteq$, that is

1. *reflexive*: $d_0 \sqsubseteq d_0$
2. *transitive*: if $d_0 \sqsubseteq d_1$ and $d_1 \sqsubseteq d_2$ then $d_0 \sqsubseteq d_2$
3. *anti-symmetric*: if $d_0 \sqsubseteq d_1$ and $d_1 \sqsubseteq d_0$ then $d_0 = d_1$

for all $d_0, d_1, d_2 \in D$

We will write $P = \langle D_P, \sqsubseteq_P \rangle$ for a generic poset. We will drop the subscripts when there is little room for confusion.

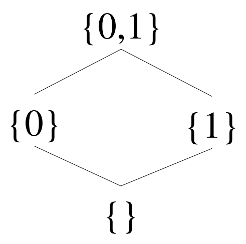
If there is an element $d_0 \in D$ such that $d_0 \sqsubseteq d$ for all $d \in D$ then d_0 is called a *least element* or a *bottom element.* It is an exercise to show that if there is a bottom element it is unique.

A bottom element of a poset P is often denoted by $\bot_P$ or just $\bot$. If there is no bottom element one often adds a bottom element to the set and redefines the relation making this new element less than all other elements. This operation is called *lifting* the poset. If $P = (D, \sqsubseteq)$ is a poset then $P_\bot \triangleq \langle D_\bot, \sqsubseteq_\bot \rangle$, where $D_\bot \triangleq D \cup \{ \bot \}$, where $\bot$ is a new element not in D and $d_1 \sqsubseteq_\bot d_2$ iff $d_1 \sqsubseteq d_2$ or $d_1 = \bot$.

7.2 Hasse diagrams

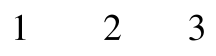
Sometimes posets are drawn as *Hasse diagrams.* Such a diagram is like a directed graph but may contain an infinite number of nodes.

Consider the poset $\langle \{ \emptyset, \{ 0 \}, \{ 1 \}, \{ 0, 1 \} \}, \subseteq \rangle$. It will be described by

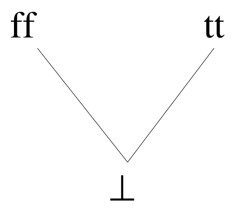


To construct a Hasse diagram one uses one node in the graph for each set element and draws an edge from d_0 to d_1 for all elements such that $d_1 \subseteq d_0$. Instead of drawing an arrow on each edge one puts d_0 below d_1. Finally one removes all edges which can be deduced as a consequence of the transitivity property of the relation. In the above example the edge from $\{ 0, 1 \}$ to $\emptyset$ is removed (or not drawn in the first place). Technically one draws a graph describing the least order relation whose "transitive closure" is the given order relation.

For any set D we have the *discrete* poset $(D, =)$ since $=$ satisfies the requirements for an order relation. Example: $(\{ 1, 2, 3 \}, =)$

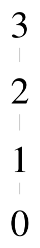


Next consider a poset with three elements, $\{ \mathrm{ff}, \mathrm{tt}, \perp \}$ and an order relation given by the diagram. This poset is called the *flat boolean poset.*



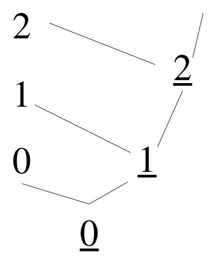
This will model a situation where we want to reason about a boolean value. Either we know nothing about the value, represented by $\perp$ or we know exactly what the value is, ff or tt For any set, D, we define the flat poset of D to be $D_\flat \triangleq (D, =)_\perp$.

For the set of natural numbers the conventional order, $\leq$, may be used. In this poset 0 is the bottom element and the Hasse diagram is a single chain of nodes.



In the domain theory other orderings for the natural numbers are more interesting. You could add a bottom element to N and define a flat ordering as we did for the boolean values; either you know nothing of a natural number or you know exactly what it is.

Another ordering describes a "lazy" representation of the natural numbers. We consider the set $\mathbb{N} \cup \{ \underline{n} \mid n \in \mathbb{N} \}$, where $\underline{0}$ is the bottom element and Hasse diagram is



The underlined element $\underline{n}$ represents the knowledge that a number is greater than or equal to n, but we don't know which is the case.

7.3 Poset examples

We are going to define a number of operations and properties for posets in general. For some familiar posets these operations will be well known and we will use these posets in examples to follow.

1. Let A be a set. The set of all subsets of A ordered with the subset relation is a poset, $(2^A, \subseteq)$.
2. The set of partial functions on N, $\mathbb{N} \hookrightarrow \mathbb{N}$, ordered by the following relation. Let $f, g \in \mathbb{N} \hookrightarrow \mathbb{N}$ and define $f \sqsubseteq g$ if and only if $f(n)$ is defined implies that $g(n)$ is defined and has the same value, $f(n) = g(n)$. We have considered a function as a set of pairs or even defined the concept as such. This relation between functions is just the ordinary subset relation for those sets. Thus $(\mathbb{N} \hookrightarrow \mathbb{N}, \subseteq)$ is a poset.
3. Less interesting in this context is the poset $(\mathbb{N}, \leq)$.
4. Next consider the set of predicates on a set A, i.e. boolean valued functions, $A \to \mathbb{B}$. Then $\langle\, A \to \mathbb{B}, \Rightarrow \,\rangle$ where $p_1 \Rightarrow p_2$ iff $p_1(a) \Rightarrow p_2(a)$ for all $a \in A$, is a poset.

7.4 Bounds

Let $P \triangleq \langle\, D, \sqsubseteq \,\rangle$ be a poset and let $A \subseteq D$. Then $d \in D$ is an *upper bound* of A if $a \sqsubseteq d$ for all $a \in A$. We will write $A \sqsubseteq_P d$ or just $A \sqsubseteq d$ if d is an upper bound of A.

If the set of upper bounds of A has a least element then we call it the *supremum* or *join* of A. It is denoted by $\bigsqcup_P A$ or just $\bigsqcup A$. When A has two elements we may write $d_0 \sqcup d_1 \triangleq \bigsqcup\{\, d_0, d_1 \,\}$.

7.5 Monotone functions

If $f \in \mathrm{R} \to \mathrm{R}$ is a real *monotone* function then it is either non increasing or nondecreasing at all points. In the first case we would require that $x_1 \leq x_2$ implies $f(x_1) \leq f(x_2)$ for all $x_1, x_2 \in R$.

This concept is carried over to functions between posets. Let P and Q be posets and $F \in D_P \to D_Q$. We say that F is *monotone* if $d_0 \sqsubseteq_P d_1$ implies $F(d_0) \sqsubseteq_Q F(d_1)$ for all $d_0, d_1 \in D_P$. It would have been more appropriate to call this *order preserving*.

7.6 Continuous functions

A real function is said to be *continuous* at a if $\lim_{x \to a} f(x) = f(a)$. This could be written

$$\lim_{x \to a} f(x) = f(\lim_{x \to a} x)$$

i.e. for a continuous function we may switch the order of taking the limit and applying a function.

A similar property is defined for monotone poset functions, but with $\sqcup$ taking the role of lim. A *chain* in a poset P is a subset $C \subseteq D_P$ which is totally ordered by $\sqsubseteq_P$. Thus if $c_1, c_2 \in C$ then $c_1 \sqsubseteq c_2$ or $c_2 \sqsubseteq c_1$. A poset is *chain complete*, abbreviated *ccpo*, if every chain has a least upper bound.

This chapter is named *domain theory*. A definition of a domain is thus required. There are different definitions of the concept *domain*. They all share the property that they are posets that are "complete" in some way. The ccpo above is just one choice.

A function $f \in D_P \to D_Q$ is *continuous* if it is monotone and $\bigsqcup_Q \{\, f(c) \mid c \in C \,\} = f(\bigsqcup_P C)$ for all nonempty chains C in P.

It is convenient to extend f so that it also can take arguments that are subsets of D_P. Let $F \in (D_P \times 2^{D_P}) \to (D_Q \times 2^{D_Q})$ be that function. Thus $F(d) \triangleq f(d)$ if $d \in D_P$.

If $A \subseteq D_P$ then $F(A) \triangleq \{\, f(a) \mid a \in A \,\}$. With this extension we can write the requirement for continuity as $\bigsqcup F(C) = F(\bigsqcup C)$ for all nonempty chains C in D_P.

In real analysis it is sometimes possible to find a *fixed point* of $f \in \mathrm{R} \to \mathrm{R}$ by iterating $x_{i+1} \triangleq f(x_i)$ from some suitable starting value, x_0. If $|f'(x)| < 1$ in some interval containing both x_0 and a fixed point of f then $\lim_{i \to \infty} x_i$ will exist and be equal to the fixed point. The limit could also be written as $\lim_{i \to \infty} f^i(x_0)$. We will see a similar result in a ccpo setting.

Theorem [**Least fixed point**]. Let $F \in D_P \to D_P$ be a continuous function on a ccpo P with least element $\bot_P$. Then

$$d \triangleq \bigsqcup \{\, F^n(\bot_P) \mid n \in \mathbb{N} \,\}$$

exists and is the least fixed point of F. ■

Proof. First we show that d exists. Since F is continuous and hence monotone $F^n \bot_P \sqsubseteq_p F^{n+1} \bot_P$ for all n and $\{\, F^n \bot_P \mid n \in \mathbb{N} \,\}$ is a chain in D_P. Since D_P is a ccpo the least upper bound of this chain exists.

Next we show that d is a fixed point of F.

$$\begin{array}{rcll}
F(d) & = & F(\bigsqcup \{\, F^n(\bot_P) \mid n \geq 0 \,\}) & \text{by definition of } d \\
& = & \bigsqcup \{\, F(F^n(\bot_P)) \mid n \geq 0 \,\} & \text{since } F \text{ is continuous} \\
& = & \bigsqcup \{\, F^{n+1}(\bot_P) \mid n \geq 0 \,\} & \text{by definition of } F^{n+1} \\
& = & \bigsqcup \{\, F^n(\bot_P) \mid n \geq 0 \,\} & \text{since } F^0(\bot_P) = \bot_P \sqsubseteq_P F^1(\bot_P) \\
& = & d & \text{by definition of } d
\end{array}$$

Finally we show that d is the least fixed point. Assume that d' is any fixed point of F. We have that $\bot_P \sqsubseteq_P d'$ and since F is monotone $F^n \bot_P \sqsubseteq_P F^n d'$ for all n. But $F^n d' = d'$. Hence d' is an upper bound of $\{\, F^n(\bot_P) \mid n \in \mathbb{N} \,\}$. Now d is the least upper bound of this chain. It follows that $d \sqsubseteq_P d'$. □

7.7 Applications

The factorial function fac satisfies the equation

$$\text{fac}(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n * \text{fac}(n-1)$$

which may be written as

$$\text{fac} = (\lambda f \,.\, (\lambda n \,.\, \text{if } n = 0 \text{ then } 1 \text{ else } n * \text{f}(n-1)))\,\text{fac}$$

Thus fac must be a fixed point of $F(f) \triangleq \lambda n \,.\, \text{if } n = 0 \text{ then } 1 \text{ else } n * \text{f}(n-1)$. Now $F \in (\mathbb{N} \hookrightarrow \mathbb{N}) \to (\mathbb{N} \hookrightarrow \mathbb{N})$ and $(\mathbb{N} \hookrightarrow \mathbb{N}, \subseteq)$ is a ccpo with a least element, the totally undefined function, $\emptyset$. Disregard the fact that we don't know if F is continuous and start iterating. Thus

$$f_0 \triangleq \emptyset$$
$$f_{i+1} \triangleq F(f_i)$$

We get $f_1 = \{ (0,1) \}$, $f_2 = \{ (0,1), (1,1) \}$, $f_3 = \{ (0,1), (1,1), (2,2) \}$, $f_4 = \{(0,1), (1,1), (2,2), (3,6)\}$, ... We could justify that $f_n = \{ (k, k!) \mid 0 \leq k < n-1 \}$ by induction. We see that $\bigsqcup\{ f_n \mid n \in \mathbb{N} \}$ denotes the factorial function. We could proceed to show that F is indeed continuous, but we refrain from doing it.

7.8 A dual theory

For a binary relation $\rho \subseteq A \times B$ one defines the *inverse* relation $\rho^{-1} \triangleq \{ (b,a) \mid (a,b) \in \rho \}$. In $(\mathbb{N}, \leq)$ the inverse relation is $\geq$. In a poset $P = (D, \sqsubseteq)$ we will use $\sqsupseteq$ to denote the inverse relation of $\sqsubseteq$. It is obvious that $P^{-1} \triangleq (D_P, \sqsupseteq_P)$ is a poset. By inverting the relation we are "turning everything upside down". The least element in P will become the *largest element* in P^{-1}, *top*, denoted by $\top$. The least upper bound in P will become the *largest lower bound* in P^{-1}. The operation to compute this bound is denoted by $\bigsqcap$.

In this setting we also have a fixed point theorem.

> **Theorem** [**Largest fixed point**]. Let $F \in D_P \to D_P$ be a continuous function on a poset $P = (D_P, \sqsubseteq_p)$ where each chain has a largest lower bound, with largest element $\top_P$. Then
> $$d \triangleq \bigsqcap\{ F^n(\top_P) \mid n \in \mathbb{N} \}$$
> exists and is the largest fixed point of F. ■

A poset $P = (D, \sqsubseteq)$ with the property that both least upper and largest lower bounds exist for all subsets of D will be called a *complete lattice*. This condition is stronger than the condition for being a ccpo.

7.9 An informal computation

Consider the language **While** with an `abort` statement, but without the `while` statement. Next consider a "statement equation" `W = if b then (S; W) else skip` If there are "solutions" to this equation one of them should be "equal to" `W = while b do S`.

Define

$$w_0 \triangleq \texttt{abort}$$

$$w_{i+1} \triangleq \texttt{if}\ b\ \texttt{then}\ (S; w_i)\ \texttt{else skip}$$

It is tempting to define

$$\text{while } b\ \texttt{do}\ S \triangleq \lim_{n\to\infty} w_i$$

but this requires a definition of $\lim_{n\to\infty}$.

$$\texttt{while}\ b\ \texttt{do}\ S \triangleq \bigsqcup\{\, w_i \mid i \in \mathbb{N}\,\}$$

but in order to do that we need a ccpo $(\mathbf{Stm}, \sqsubseteq)$. We will not try this. However we will take a similar approach when defining the meaning function for the `while` statement in denotational semantics.

7.10 Which functions are continuous?

The answer to the above question is: Most interesting and computable functions are continuous. There are extremely many ways to define functions on ccpo's. Most of them will not even be monotone and most of them will not be interesting. We will describe one most interesting but uncomputable function that is not monotone.

Consider the language **While**. We will make it a poset $(\mathbf{Stm}, \sqsubseteq_{ns})$ by defining $S_1 \sqsubseteq_{ns} S_2$ iff $\mathcal{S}_{ns}[\![S_1]\!] \subseteq \mathcal{S}_{ns}[\![S_2]\!]$ and equality is defined by $=_{ns}$. Strictly speaking we are considering "equivalence classes" in **While**, not the individual elements in **Stm**. There is, for example, the equivalence class containing all statements that are equivalent to `skip`. This class will contain e.g. `skip` and `skip; skip` and `x:=x` and `skip; x:=x`. There is a bottom element in this set containing all statements that will loop from any state. It would be nice to have a function that decides if a statement belongs to this class,

$$\mathit{loops} \in \texttt{Stm} \to \mathrm{T}$$

such that

$$\mathit{loops}(S) \triangleq \mathrm{tt}, \qquad \text{if } S \text{ loops}$$

$$\mathit{loops}(S) \triangleq \mathrm{ff}, \qquad \text{otherwise}$$

We have to specify some order relation for T. It is reasonable to choose the discrete order and the poset $(\mathrm{T}, =)$. Now `while true do skip` $\sqsubseteq_{ns}$ `skip` but *loops*(`while true do skip`) $\not\sqsubseteq$ *loops*(`skip`) since tt and ff are not related by =. It is well known that *loops* is uncomputable; it is the *halting problem* and you cannot write a program that decides if another program loops forever or not.

Another reasonable poset choice would be to use $(\mathrm{T}, =)_\perp$. This would not change the conclusion. A less reasonable choice would be to let T have an ordering relation with tt $\sqsubseteq$ ff. But then the negation of *loops*, *terminates*, would not be monotone.

In order to apply the fixed point theorem we need to prove that F is continuous. If our main object was to study domain theory we could consider different ways to construct ccpo's and functions on them and we would find that most useful constructions yield continuous functions. The first result on such a course follows below.

A function $F \in D \to E$ is said to be *strict* if $F(\perp_D) = \perp_E$.

Theorem. Any flat poset $D_\flat = (D, =)_\perp$ is a ccpo and any strict function $F \in D_\flat \to E$ is continuous. ■

Proof. Since every chain in D has at most two elements it has a least upper bound. If it has two elements then one of them is $\perp_D$.

In each case it is immediate that $F(\bigsqcup C) = \bigsqcup F(C)$. □

It follows that all arithmetic functions, such as $+ \in \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, are continuous when we use the discrete ordering.

Since our main objective is to study semantics of programming languages we will just consider constructions needed to define the semantic function for a `while` statement and similar statements.

This is a good place to look at the denotational semantics of **While**. We will return here after discussing Table 4.1 in Nielson.

7.11 Continuity properties

Theorem [**4.24**].[1] Every ccpo has a least element. ■

Proof. Let $P = (D, \sqsubseteq)$ be a ccpo. By definition $\emptyset$ is a chain; since there are no elements in $\emptyset$ the requirements are vacuously fulfilled. Thus $\bigsqcup \emptyset$ exists. Every element in D will vacuously be an upper bound of $\emptyset$. It follows that $\bigsqcup \emptyset \sqsubseteq d$ for all $d \in D$. So there is a least element. □

Theorem [**4.25**]. Let A, B be sets. Then $(A \hookrightarrow B, \subseteq)$ is a ccpo. If C is a chain in $A \hookrightarrow B$ then $\bigsqcup C = \bigcup C$. ■

Proof. First we have to show that $\bigcup C$ is an element in $A \hookrightarrow B$, i.e. a function. The union of a number of functions is not always a function. Assume that $(a_1, b_1), (a_1, b_2) \in \bigcup C$. We have to show that $b_1 = b_2$. There must be functions $f_1, f_2 \in C$ such that $(a_1, b_1) \in f_1$ and $(a_1, b_2) \in f_2$. But since C is a chain $f_1 \subseteq f_2$ or $f_2 \subseteq f_1$. In either case (a_1, b_1) and (a_1, b_2) must both belong to f_1 or f_2. It follows that $b_1 = b_2$.

Next we observe that $\bigcup\{ f \in C \}$ is an upper bound of C.

Finally we must show $\bigcup C$ is the least upper bound. Let $g \in A \hookrightarrow B$ be an upper bound of C. Then $f \subseteq g$ for all $f \in C$. It follows that $(\bigcup C) \subseteq g$. □

Composition of functions preserves monotonicity and continuity as seen by the following theorems.

Theorem [**4.29**]. Let $(D_1, \sqsubseteq_1)$, $(D_2, \sqsubseteq_2)$ and $(D_3, \sqsubseteq_3)$ be ccpo's and $f \in D_1 \to D_2$, $g \in D_2 \to D_3$ be monotone. Then $h \in D_1 \to D_3$ where $h(d) \triangleq g(f(d)) = (g \circ f)(d)$ is monotone. ■

Proof. Let $d \sqsubseteq d'$ in D_1. Then $f(d) \sqsubseteq_1 f(d')$ and $g(f(d)) \sqsubseteq_2 g(f(d'))$, i.e. $h(d) \sqsubseteq_2 h(d')$. □

[1]Theorem numbers refer to Nielson.

Chains are mapped into chains by monotone functions.

Theorem [**4.30**]. Let $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$ be ccpo's. If $f \in D_1 \to D_2$ is monotone and C is a chain in D_1 then $f(C)$ is a chain in D_2. Furthermore $\bigsqcup_2 f(C) \sqsubseteq_2 f(\bigsqcup_1 C)$ ■

Proof. Remember that $f(C) = \{ f(c) \mid c \in C \}$. If C is empty then $\bigsqcup_1 C = \perp_1$ and $\bigsqcup_2 f(C) = \perp_2$ and $\perp_2 \sqsubseteq_2 f(\perp_1)$.

If C is nonempty and $f(c_1), f(c_2) \in f(C)$ then either $c_1 \sqsubseteq_1 c_2$ or $c_2 \sqsubseteq_1 c_1$. It follows that $f(c_1) \sqsubseteq_2 f(c_2)$ or $f(c_2) \sqsubseteq_2 f(c_1)$ and $f(C)$ is a chain.

To prove that $\bigsqcup_2 f(C) \sqsubseteq_2 f(\bigsqcup_1 C)$ let c be any element in C. Then $c \sqsubseteq_1 \bigsqcup_1 C$ and by monotonicity $f(c) \sqsubseteq_2 f(\bigsqcup_1 C)$. So $f(\bigsqcup_1 C)$ is an upper bound for $f(C)$ and it must be greater than the least upper bound. □

Continuity would change the last relation into an equality.

Theorem [**4.35**]. If f and g are continuous then so is $g \circ f$. ■

Proof. By 4.29 we know that $g \circ f$ is monotone. Let C be a chain. Since f is continuous $\bigsqcup f(C) = f(\bigsqcup C)$. f is monotone. Thus $f(C)$ is a chain. It follows that $\bigsqcup g(f(C)) = g(\bigsqcup f(C))$. Combining these equalities we get $\bigsqcup g(f(C)) = g(f(\bigsqcup C))$ or $\bigsqcup (g \circ f)(C)) = (g \circ f)(\bigsqcup C)$.

It remains to show that $\bigsqcup (g \circ f)(C) \sqsubseteq (g \circ f)(\bigsqcup C)$ for every chain C. □

The next two theorems are required to define the denotational meaning of the `while` statement.

Theorem [**4.43**]. Let $g_0 \in \textbf{State} \hookrightarrow \textbf{State}$, $p \in \textbf{State} \to \mathrm{T}$ and $F \in (\textbf{State} \hookrightarrow \textbf{State}) \to (\textbf{State} \hookrightarrow \textbf{State})$ be defined by

$$F(g) \triangleq \lambda\sigma \,.\, \text{if } p\ \sigma \text{ then } g\ \sigma \text{ else } g_0\ \sigma$$

Then F is continuous on $(\textbf{State} \hookrightarrow \textbf{State}, \subseteq)$. ■

In the proof we will not use any property of **State** so it could be replaced by any set. When we use the theorem g and g_0 will be semantic functions. The predicate p will be the semantic function for a boolean expression.

Proof. First we show that F is monotone. Assume that $g_1 \subseteq g_2$. We have to show that $F(g_1) \subseteq F(g_2)$. Let $(\sigma, \sigma') \in F(g_1)$. Now there are two cases.

- If $p\ \sigma = \mathrm{tt}$ then $\sigma' = g_1\ \sigma$. Since $g_1 \subseteq g_2$ it follows that $\sigma' = g_2\ \sigma$ and $(\sigma, \sigma') \in F(g_2)$.
- If $p\ \sigma = \mathrm{ff}$ then $\sigma' = g_0\ \sigma$ and $(\sigma, \sigma') \in F(g_2)$.

To prove that F is continuous, let C be a nonempty chain in $\textbf{State} \hookrightarrow \textbf{State}$. We must show that $\bigsqcup F(C) = F(\bigsqcup C)$. By 4.30 we know $\bigsqcup F(C) \sqsubseteq F(\bigsqcup C)$ so it suffices to show that $F(\bigsqcup C) \sqsubseteq \bigsqcup F(C)$. Let $(\sigma, \sigma') \in F(\bigsqcup C)$. Again there are two cases.

- If $p\ \sigma = \mathrm{tt}$ then $\sigma' = (\bigsqcup C)\ \sigma$. Since C is a chain of partial functions, for every $g \in C$ either $g\,\sigma = \sigma'$ or $g\,\sigma$ is undefined and there is some $g \in C$ such that $g\,\sigma = \sigma'$. It follows that $(\sigma, \sigma') \in \bigsqcup C = F(\bigsqcup C)$.

- If $p\,\sigma = \mathrm{ff}$ then $\sigma' = g_0\,\sigma$ and $(\sigma, \sigma') \in F(\bigsqcup C)$.

□

Finally we need

Theorem [**4.45**]. Let $g_0 \in \mathbf{State} \hookrightarrow \mathbf{State}$, and $F \in (\mathbf{State} \hookrightarrow \mathbf{State}) \to (\mathbf{State} \hookrightarrow \mathbf{State})$ be defined by

$$F(g) \triangleq g \circ g_0 = \lambda\sigma\,.\,g(g_0(\sigma))$$

Then F is continuous. ■

This means that the operation of composition of a function with a fixed function is continuous. The theorem makes no assumptions on the function argument and this result is independent of 4.35.

Proof. First we show that F is monotone. Assume that $g_1 \subseteq g_2$. We have to show that $F(g_1) \subseteq F(g_2)$. Let $(\sigma, \sigma') \in F(g_1)$. This means that there is a σ_0 such that $g_1(\sigma) = \sigma_0$ and $g_0(\sigma_0) = \sigma'$. It follows that $g_2(\sigma) = \sigma_0$ and $(\sigma, \sigma') \in F(g_2)$.

To prove that F is continuous, let C be a nonempty chain in $\mathbf{State} \hookrightarrow \mathbf{State}$. We must show that $\bigsqcup F(C) = F(\bigsqcup C)$. Again by Lemma 4.30 it suffices to show that $F(\bigsqcup C) \sqsubseteq \bigsqcup F(C)$. Let $(\sigma, \sigma') \in (\bigsqcup C) \circ g_0$. Thus there is a σ_0 such that $g_0(\sigma) = \sigma_0$ and $(\bigsqcup C)(\sigma_0) = \sigma'$. It follows that for each $g \in C$ either $g\,\sigma_0$ is undefined or $g\,\sigma_0 = \sigma'$ and that the latter case occurs at least once. We conclude that either $(g \circ g_0)(\sigma)$ is undefined or $(g \circ g_0)(\sigma) = \sigma'$. This proves that $(\sigma, \sigma') \in \bigsqcup F(C)$. □

7.12 Why choose the least fixed point?

If we apply the least fixed point construction to

$$\mathrm{f}_2(0) \triangleq 0$$
$$\mathrm{f}_2(n) \triangleq \mathrm{f}_2(n+1), \qquad n > 0$$

we will get

$$\mathrm{f}_2(0) \triangleq 0$$
$$\mathrm{f}_2(n) \triangleq \text{undefined if} \qquad n > 0$$

However for any $k \in \mathbb{N}$

$$\mathrm{f}_2(0) \triangleq 0$$
$$\mathrm{f}_2(n) \triangleq k \qquad n > 0$$

will be a fixed point. So if we do not choose the least fixed point we have to make some arbitrary choice to make the function more defined. This is not reasonable for a definition.

7.13 References

These notes are inspired by

1. S. Abramsky, A. Jung: *Domain Theory*, `www.cs.bham.ac.uk/~axj/pub/papers/handy1.pdf`
2. D.A. Schmidt: *Denotational Semantics*, Allyn and Bacon, 1986.
3. J.E. Stoy: *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1979.