CS422 - Programming Language Design

Denotational Semantics

Grigore Roşu

Department of Computer Science

University of Illinois at Urbana-Champaign

Denotational semantics, also known as *fix-point semantics*, associates to each programming language construct a well-defined and understood *mathematical object*, typically a function. The mathematical object denotes the behavior of the corresponding language construct, so equivalence of programs is immediately translated into equivalence of mathematical objects. The later equivalence can be then shown using the entire arsenal of mathematics.

We next present a denotational semantics for our simple imperative language and show that it is equivalent to the other two semantics discussed so far, namely the equational semantics and SOS. The denotational semantics of arithmetic and boolean expressions is rather straightforward. The unexpectedly difficult part is to properly give semantics to loops, because it involves non-trivial mathematics regarding fix-points of special operators.

Denotational Semantics of Arithmetic Expressions

An arithmetic expression can be seen as a function taking a state to an integer value, the value of the expression in that state. However, note that not any arithmetic expression is well defined in any state, because of division by zero. Thus, we can define an operator

$$\llbracket \_ \rrbracket : \ \mathtt{AExp} \to (\Sigma \rightharpoonup \mathbb{Z})$$

taking arithmetic expressions to partial functions from states to integer numbers. As in operational semantics, states are regarded as maps from names to values. In what follows, we will write $\llbracket a \rrbracket \sigma$ instead of $\llbracket a \rrbracket (\sigma)$.

The denotation of names x and integers i is defined as expected:

$$\llbracket x \rrbracket \sigma = \sigma(x) \tag{1}$$

$$\llbracket i \rrbracket \sigma = i \tag{2}$$

The denotations of addition, subtraction and multiplication are:

$$[\![a_1 + a_2]\!]\sigma = [\![a_1]\!]\sigma + [\![a_2]\!]\sigma \tag{3}$$

$$[\![a_1 - a_2]\!]\sigma = [\![a_1]\!]\sigma - [\![a_2]\!]\sigma \tag{4}$$

$$[\![a_1 * a_2]\!]\sigma = [\![a_1]\!]\sigma * [\![a_2]\!]\sigma \tag{5}$$

For division, we have to consider the situation when the denominator is zero:

$$[\![a_1/a_2]\!]\sigma = \begin{cases} \text{undefined} & \text{if } [\![a_2]\!]\sigma = 0 \\ \text{integer quotient } [\![a_1]\!]\sigma \text{ by } [\![a_2]\!]\sigma & \text{if } [\![a_2]\!]\sigma \neq 0 \end{cases} \tag{6}$$

We implicitly assume that the denotation of an expression in a state is undefined whenever any of its subexpressions is undefined.

Denotational Semantics of Boolean Expressions

Partial functions from states to boolean values are associated to boolean expressions via the denotation operator:

$$\llbracket \_ \rrbracket : \texttt{BExp} \to (\Sigma \rightharpoonup \{\texttt{true}, \texttt{false}\})$$

The denoting functions are partial because the boolean expression can involve arithmetic expressions which may be undefined. This operator can be defined as follows:

$$\llbracket \texttt{true} \rrbracket \sigma = \texttt{true} \tag{7}$$

$$\llbracket \texttt{false} \rrbracket \sigma = \texttt{false} \tag{8}$$

$$\llbracket a_1 \leq a_2 \rrbracket \sigma = \begin{cases} \texttt{true} & \text{if } \llbracket a_1 \rrbracket \sigma \leq \llbracket a_2 \rrbracket \sigma \\ \texttt{false} & \text{otherwise} \end{cases} \tag{9}$$

$$\llbracket a_1 \geq a_2 \rrbracket \sigma = \begin{cases} \text{true} & \text{if } \llbracket a_1 \rrbracket \sigma \geq \llbracket a_2 \rrbracket \sigma \\ \text{false} & \text{otherwise} \end{cases} \tag{10}$$

$$\llbracket a_1 \text{ equals } a_2 \rrbracket \sigma = \begin{cases} \text{true} & \text{if } \llbracket a_1 \rrbracket \sigma = \llbracket a_2 \rrbracket \sigma \\ \text{false} & \text{otherwise} \end{cases} \tag{11}$$

$$\llbracket b_1 \text{ and } b_2 \rrbracket \sigma = \begin{cases} \text{true} & \text{if } \llbracket b_1 \rrbracket \sigma \text{ and } \llbracket b_2 \rrbracket \sigma \text{ are true} \\ \text{false} & \text{otherwise} \end{cases} \tag{12}$$

$$\llbracket b_1 \text{ or } b_2 \rrbracket \sigma = \begin{cases} \text{true} & \text{if } \llbracket b_1 \rrbracket \sigma \text{ or } \llbracket b_2 \rrbracket \sigma \text{ is true} \\ \text{false} & \text{otherwise} \end{cases} \tag{13}$$

$$\llbracket \text{not } b \rrbracket \sigma = \begin{cases} \text{true} & \text{if } \llbracket b \rrbracket \sigma = \text{false} \\ \text{false} & \text{if } \llbracket b \rrbracket \sigma = \text{true} \end{cases} \tag{14}$$

Denotational Semantics of Statements

In addition to partiality due to division by zero in expressions that statements may involve, partiality in the denotation of statements may also occur for another important reason: *loops may not terminate*. For example, the statement `while (x > y) skip` will not terminate in those states in which the value that x denotes is larger than that of y. Mathematically, we say that the function from states to states that this loop statement denotes is undefined in those states in which the loop statement does not terminate. The denotation of statements is therefore an operator

$$\llbracket \_ \rrbracket : \texttt{Stmt} \to (\Sigma \rightharpoonup \Sigma).$$

We next define the denotational semantics of all statements except loops. Loops will be discussed separately later, because their denotational semantics is less trivial.

The following definitions are natural:

$$\llbracket \texttt{skip} \rrbracket = 1_\Sigma \text{ (the identity function on states)} \qquad (15)$$

$$\llbracket x = a \rrbracket \sigma = \begin{cases} \sigma[x \leftarrow \llbracket a \rrbracket \sigma] & \text{if } \llbracket a \rrbracket \sigma \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases} \qquad (16)$$

$$\llbracket s_1; s_2 \rrbracket = \llbracket s_2 \rrbracket \circ \llbracket s_1 \rrbracket \text{ (the composition of the two functions)} \qquad (17)$$

$$\llbracket \{s\} \rrbracket = \llbracket s \rrbracket \qquad (18)$$

$$\llbracket \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \rrbracket \sigma = \begin{cases} \llbracket s_1 \rrbracket \sigma & \text{if } \llbracket b \rrbracket \sigma = \texttt{true} \\ \llbracket s_2 \rrbracket \sigma & \text{if } \llbracket b \rrbracket \sigma = \texttt{false} \\ \text{undef} & \text{if } \llbracket b \rrbracket \sigma \text{ undefined} \end{cases} \qquad (19)$$

Examples

What is the denotation of $x = 1;\ y = 2;\ x = 3$, i.e., the function

$$[\![x = 1;\ y = 2;\ x = 3]\!]\ ?$$

By the denotation of composition, we get
$[\![x = 3]\!] \circ [\![y = 2]\!] \circ [\![x = 1]\!]$. Applying it on a state σ, one gets
$([\![x = 3]\!] \circ [\![y = 2]\!] \circ [\![x = 1]\!])\sigma$ equals $((\sigma[x \leftarrow 1])[y \leftarrow 2])[x \leftarrow 3]$; let
us denote the later state by σ'. By the definition of function
update, one can easily see that σ' can be defined as

$$\sigma'(z) = \begin{cases} 3 & \text{if } z = x \\ 2 & \text{if } z = y \\ \sigma(z) & \text{otherwise,} \end{cases}$$

which is nothing but $\sigma[x \leftarrow 3][y \leftarrow 2]$.

Similarly, we can show that

$$[\![(\texttt{if } y > z \texttt{ then } x = 1 \texttt{ else } x = 2); x = 3]\!] = \lambda\sigma.\sigma[x \leftarrow 3],$$

where by abuse of notation we let $\lambda\sigma.\sigma[x \leftarrow 3]$ denote the function taking states σ to states $\sigma[x \leftarrow 3]$. Indeed,
$[\![(\texttt{if } y > z \texttt{ then } x = 1 \texttt{ else } x = 2); x = 3]\!]\sigma = \sigma'[x \leftarrow 3]$, where

$$\sigma' = \begin{cases} \sigma[x \leftarrow 1] & \text{if } [\![y > z]\!]\sigma = \texttt{true} \\ \sigma[x \leftarrow 2] & \text{otherwise,} \end{cases}$$

which can be easily shown equal to $\sigma[x \leftarrow 3]$.

Denotational Semantics of Programs

Before we discuss the denotational semantics of loop statements, we can discuss the denotation of programs, which is much easier. A program consists of a statement followed by an arithmetic expression, with the meaning that the statement is evaluated in the initial state and then the expression is evaluated in the obtained state. Thus, we can define the denotation of programs simply as

$$\llbracket \_ \rrbracket \; : \; \texttt{Pgm} \to \mathbb{Z}$$

$$\llbracket s; a \rrbracket = \llbracket a \rrbracket (\llbracket s \rrbracket \phi)$$

where ϕ is the initial state, which, in our current definition of the language, assigns zero to any name. By removing the equation "`eq S[X] = 0 [owise]`", one can define the initial state as a partial function from names to integer numbers.

Equivalence of Expressions, Statements, Programs

Within denotational semantics, it is quite straightforward to define equivalence of expressions, statements or programs. For example, two statements s_1 and s_2 are equivalent if and only if they denote the same mathematical objects, that is, if and only if $[\![s_1]\!] = [\![s_2]\!]$.

One can thus very easily show the equivalence of statements

$$(\texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2); s \equiv \texttt{if } b \texttt{ then } s_1; s \texttt{ else } s_2; s$$

for any statements s_1, s_2 and s_3, and for any boolean expression b. Indeed, the first denotes the function

$$\lambda\sigma.[\![s]\!]\left(\begin{cases} [\![s_1]\!]\sigma & \text{if } [\![b]\!]\sigma = \texttt{true} \\ [\![s_2]\!]\sigma & \text{if } [\![b]\!]\sigma = \texttt{false} \\ \text{undefined} & \text{if } [\![b]\!]\sigma \text{ undefined} \end{cases} \right)$$

while the second denotes the function

$$
\lambda\sigma. \begin{cases} [\![s]\!]([\![s_1]\!]\sigma) & \text{if } [\![b]\!]\sigma = \mathtt{true} \\ [\![s]\!]([\![s_2]\!]\sigma) & \text{if } [\![b]\!]\sigma = \mathtt{false} \\ \text{undefined} & \text{if } [\![b]\!]\sigma \text{ undefined} \end{cases}
$$

It is clear now that the two are equal as mathematical objects. Prove it rigorously!

The major point of denotational semantics that you have to remain with after this class is the following:

> Denotational semantics is an abstract technique to assign meaning to programs, in which programming language constructs are mapped into mathematical objects. The intermediate states are ignored in denotational semantics.

Denotational Semantics of Loops

The only definition left is the denotational semantics of loops, or in other words the partial functions

$$\llbracket \texttt{while}(b)\ s \rrbracket : \ \Sigma \rightharpoonup \Sigma$$

where $\llbracket \texttt{while}(b)\ s \rrbracket(\sigma) = \sigma'$ if and only if the while loop correctly terminates in state σ' when executed in state σ. Such a σ' may not always exist for two reasons: (1) because the denotation of b or s in some appropriate state encounters a division by zero, and (2) because s (which may contain nested loops) or the while loop itself does not terminate. If we let $\mathcal{W}$ denote the partial function

$[\![\mathtt{while}(b)\ s]\!]$, then the most natural definition of $\mathcal{W}$ is:

$$\mathcal{W}(\sigma) = \begin{cases} \sigma & \text{if } [\![b]\!]\sigma = \mathtt{false} \\ \mathcal{W}([\![s]\!]\sigma) & \text{if } [\![b]\!]\sigma = \mathtt{true} \\ \text{undefined} & \text{if } [\![b]\!]\sigma \text{ undefined} \end{cases}$$

Mathematically speaking, this is a problematic definition for several reasons:

- $\mathcal{W}$ is defined in terms of itself;

- it is not clear that such a $\mathcal{W}$ exists;

- in case it exists, it is not clear that such a $\mathcal{W}$ is unique.

We next develop the mathematical machinery needed to rigorously define and reason about partial functions like the $\mathcal{W}$ above.

Partial Functions as Information Bearers

One very convenient interpretation of partial functions that significantly eases the understanding of the subsequent mathematics is as *information* or *knowledge bearers*. More precisely, a partial function $\mathcal{I} : \Sigma \rightharpoonup \Sigma$ can be thought of as carrying knowledge about some states in Σ, namely exactly those on which it is defined. For such a state σ, the knowledge that it carries is $\mathcal{I}(\sigma)$. If $\mathcal{I}$ is not defined on a state $\sigma \in \Sigma$ then we can think of it as "$\mathcal{I}$ *does not have any information about* σ".

Recall that a *partial order* on a set, say D, is a binary relation, say $\sqsubseteq$, which is

- *reflexive*, i.e., $x \sqsubseteq x$,

- *transitive*, i.e., $x \sqsubseteq y$ and $y \sqsubseteq z$ implies $x \sqsubseteq z$, and

- *anti-symmetric*, i.e., $x \sqsubseteq y$ and $y \sqsubseteq x$ implies $x = y$.

A partial order relation occurs naturally on partial functions. If $\mathcal{I}, \mathcal{J} : \Sigma \rightharpoonup \Sigma$ are partial functions, then we say that $\mathcal{I}$ *is less informative then or as informative as* $\mathcal{J}$, written $\mathcal{I} \preceq \mathcal{J}$, if and only if for any $\sigma \in \Sigma$, it is either the case that $\mathcal{I}(\sigma)$ is not defined, or both $\mathcal{I}(\sigma)$ and $\mathcal{J}(\sigma)$ are defined and $\mathcal{I}(\sigma) = \mathcal{J}(\sigma)$. If $\mathcal{I} \preceq \mathcal{J}$ then we may also say that $\mathcal{J}$ *refines* $\mathcal{I}$ or that $\mathcal{J}$ *extends* $\mathcal{I}$.

Intuition for the Denotation of Loops

One can, and should, think of each possible iteration of a while loop as an opportunity to refine the knowledge about its denotation. Before the boolean expression b of the while loop $\texttt{while}(b)\ s$ is evaluated the first time, the knowledge that one has about $\mathcal{W}$ is $\mathcal{W}_0 := \bot : \Sigma \rightharpoonup \Sigma$, which denotes the function which is not defined in any state. Therefore, $\mathcal{W}_0$ corresponds to no information.

Now suppose that we evaluate the boolean expression b in some state σ and that it is false. Then the denotation of the while loop should return σ, which suggests that we can refine our knowledge about $\mathcal{W}$ from $\mathcal{W}_0$ to the partial function $\mathcal{W}_1 : \Sigma \rightharpoonup \Sigma$, which is an identity on all those states $\sigma \in \Sigma$ for which $[\![b]\!]\sigma = \texttt{false}$.

So far we have not considered any state in which the loop needs to evaluate its body. Suppose now that for some state σ, it is the case that $[\![b]\!]\sigma = \texttt{true}$, $[\![s]\!]\sigma = \sigma'$, and $[\![b]\!]\sigma' = \texttt{false}$, that is, that the while loop terminates in one iteration. Then we can extend $\mathcal{W}_1$ to a partial function $\mathcal{W}_2 : \Sigma \rightharpoonup \Sigma$, which, in addition to being an identity on those states on which $\mathcal{W}_1$ is defined, takes each σ as above to $\mathcal{W}_2(\sigma) = \sigma'$. Therefore, $\mathcal{W}_1 \preceq \mathcal{W}_2$.

By iterating this process, one can define for any natural number k a partial function $\mathcal{W}_k : \Sigma \rightharpoonup \Sigma$, which is defined on all those states on which the while loop terminates in *at most k* evaluations of its boolean condition (i.e., $k-1$ executions of its body). An

immediate property of the partial functions $\mathcal{W}_0$, $\mathcal{W}_1$, $\mathcal{W}_2$, ..., $\mathcal{W}_k$ is that they increasingly refine each other, that is,
$$\mathcal{W}_0 \preceq \mathcal{W}_1 \preceq \mathcal{W}_2 \preceq \cdots \preceq \mathcal{W}_k.$$

Informally, the partial functions $\mathcal{W}_k$ approximate $\mathcal{W}$ as k increases; more precisely, for any $\sigma \in \Sigma$, if $\mathcal{W}(\sigma) = \sigma'$, that is, if the while loop terminates and σ' is the resulting state, then one can show that there is some k such that $\mathcal{W}_k(\sigma) = \sigma'$. Moreover, it follows easily that $\mathcal{W}_n(\sigma) = \sigma'$ for any $n \geq k$.

But the main question still remains unanswered: how can we define the denotation $\mathcal{W} : \Sigma \rightharpoonup \Sigma$ of while loops? According to the intuitions above, $\mathcal{W}$ should be "some sort of limit" of the (infinite) sequence of partial functions $\mathcal{W}_0 \preceq \mathcal{W}_1 \preceq \mathcal{W}_2 \preceq \ldots \preceq \mathcal{W}_k \preceq \ldots$, but the notion of limit of partial functions partially ordered by the "more informative" partial ordering does not seem to be immediate.

In the next lecture we develop a general theory of fix-points, which

will then be elegantly applied to our special case and thus properly define the denotation $\mathcal{W} : \Sigma \rightharpoonup \Sigma$ of while loops. Before that, let us first define a total function $\mathcal{F} : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ taking partial functions to partial functions as follows:

$$\mathcal{F}(\alpha)(\sigma) = \begin{cases} \sigma & \text{if } [\![b]\!]\sigma = \mathtt{false} \\ \alpha([\![s]\!]\sigma) & \text{if } [\![b]\!]\sigma = \mathtt{true} \\ \text{undefined} & \text{if } [\![b]\!]\sigma \text{ undefined} \end{cases}$$

Notice that the informal partial functions $\mathcal{W}_k$ above can be now rigorously defined as $\mathcal{F}^k(\bot)$, where $\mathcal{F}^k$ stays for k compositions of $\mathcal{F}$ and $\mathcal{F}^0 = \bot$ by convention. Indeed, one can show by induction on k the following property, where $[\![s]\!]^i$ stays for i compositions of $[\![s]\!] : \Sigma \rightharpoonup \Sigma$ and $[\![s]\!]^0$ is by convention the identity (total) function

on Σ:

$$
\mathcal{F}^k(\bot)(\sigma) =
\begin{cases}
[\![s]\!]^i \sigma & \text{if there is } 0 \leq i < k \text{ s.t. } ([\![b]\!] \circ [\![s]\!]^i)\sigma = \texttt{false} \\
& \text{and } ([\![b]\!] \circ [\![s]\!]^j)\sigma = \texttt{true} \text{ for all } 0 \leq j < i \\
\text{undef} & \text{otherwise}
\end{cases}
$$

Note that the function $\mathcal{F}^k(\bot)$ is well defined, in the sense that if an i as above exists then it is unique.

In what follows we develop a theoretical framework which will allow us to prove an important fix-point theorem stating the existence of least fix-points of certain operators. Applied to our total function

$$\mathcal{F} : \ (\Sigma \rightharpoonup \Sigma) \rightarrow (\Sigma \rightharpoonup \Sigma)$$

this theorem will give us the denotation of while loops.

Posets and (Least) Upper Bounds

Let $(D, \sqsubseteq)$ be a partial order, that is, a set D together with a binary relation $\sqsubseteq$ on it which is reflexive, transitive and anti-symmetric. Partial orders are also called *posets*. Given a set of elements $X \subseteq D$, an element $p \in D$ is called an *upper bound (ub)* of X if and only if $x \sqsubseteq p$ for any $x \in X$. Furthermore, $p \in D$ is called a *least upper bound (lub)* of X if and only if p is an upper bound and for any other upper bound q of X it is the case that $p \sqsubseteq q$.

Note that upper bounds and least upper bounds may not always exist. For example, if $D = X = \{x, y\}$ and $\sqsubseteq$ is the identity relation, then X has no upper bounds. Least upper bounds may not exist even though upper bounds exist. For example, if $D = \{a, b, c, d, e\}$ and $\sqsubseteq$ is defined by $a \sqsubseteq c$, $a \sqsubseteq d$, $b \sqsubseteq c$, $b \sqsubseteq d$, $c \sqsubseteq e$, $d \sqsubseteq e$, then any subset X of D admits upper bounds, but the set $X = \{a, b\}$ does not have a least upper bound.

Due to the anti-symmetry property, least upper bounds are unique when they exist. For that reason, we let $\sqcup X$ denote the lub of X.

Chains

Given a poset $(D, \sqsubseteq)$, a *chain* in D is an infinite sequence $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ of elements in D, also written using set notation as $\{d_n \mid n \in \mathbb{N}\}$. Such a chain is called *stationary* when

there is some $n \in \mathbb{N}$ such that $d_m = d_{m+1}$ for all $m \geq n$.

Recall that the infinite sequence of partial functions $\Sigma \rightharpoonup \Sigma$

$$\bot \preceq \mathcal{F}(\bot) \preceq \mathcal{F}^2(\bot) \preceq \cdots \preceq \mathcal{F}^n(\bot) \preceq \cdots$$

incrementally approximates the desired denotation of a while loop. Thus, we would like to define the denotation of the while loop as $\sqcup\{\mathcal{F}^n(\bot) \mid n \in \mathbb{N}\}$, in case it exists. In the simple case when this sequence regarded as a chain in $(\Sigma \rightharpoonup \Sigma, \preceq)$ is stationary, with the intuition that the while loop terminates in any state in some fixed number of iterations which does not depend on the state, then the denotation of the while loop is the partial function in which the chain stabilizes, that is, its lub. Unfortunately, in most of the practical situations this chain is not stationary. For example, the simple loop `while (k > 0) k = k - 1` terminates in any state, but there is no bound on the number of iterations. Consequently, there is no n such that $\mathcal{F}^n(\bot) = \mathcal{F}^{n+1}(\bot)$. Indeed, the later has

strictly more information than the former: $\mathcal{F}^{n+1}$ is defined on all those states σ with $\sigma(k) = n + 1$, while $\mathcal{F}^n$ is not.

Complete Partial Orders (cpos)

A poset $(D, \sqsubseteq)$ is called a *complete partial order (cpo)* if and only if any of its chains has a lub. $(D, \sqsubseteq)$ is said to *have bottom* if and only if it has a minimal element. Such element is typically denoted by $\bot$, and the poset with bottom $\bot$ is written $(D, \sqsubseteq, \bot)$. If $\{d_n \mid n \in \mathbb{N}\}$ is a chain in $(D, \sqsubseteq)$, then we also let $\bigsqcup_{n \in \mathbb{N}} d_n$ or even $\sqcup d_n$ denote its lub $\sqcup \{d_n \mid n \in \mathbb{N}\}$.

Examples

$(\mathcal{P}(S), \subseteq, \emptyset)$ is a cpo with bottom, where $\mathcal{P}(S)$ is the set of subsets of a set S and $\emptyset$ is the empty set.

$(\mathbb{N}, \leq)$, the set of natural numbers ordered by "less than or equal to", has bottom 0 but is not complete: the sequence $0 \leq 1 \leq 2 \leq \cdots \leq n \leq \cdots$ has no upper bound in $\mathbb{N}$.

$(\mathbb{N} \cup \{\infty\}, \leq, 0)$, the set of natural numbers plus infinity, where infinity is larger than any number, is a cpo with bottom 0. It is a cpo because any chain is either stationary, in which case its lub is obvious, or is unbounded by any natural number, in which case ∞ is its lub.

$(\mathbb{N}, \geq)$ is a cpo but has no bottom.

$(\mathbb{Z}, \leq)$ is not a cpo and has no bottom.

$(S, =)$, a flat set S where the only partial ordering is the identity, is

a cpo. It has bottom if and only if S has only one element.

Most importantly, $(\Sigma \rightharpoonup \Sigma, \preceq, \bot)$, the set of partial functions $\Sigma \rightharpoonup \Sigma$ ordered by the informativeness relation $\preceq$ is a cpo with bottom $\bot: \Sigma \rightharpoonup \Sigma$, the function which is undefined in each state. You will have to prove this as part of your homework.

Monotone and Continuous Functions

If $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ are two posets and $f: D \rightarrow D'$ is a function, then f is called *monotone* if and only if $f(x) \sqsubseteq' f(y)$ for any $x, y \in D$ with $x \sqsubseteq y$. If f is monotone, then we simply write $f: (D, \sqsubseteq) \rightarrow (D', \sqsubseteq')$.

Monotone functions *preserve chains*, that is, $\{f(d_n) \mid n \in \mathbb{N}\}$ is a chain in $(D', \sqsubseteq')$ whenever $\{d_n \mid n \in \mathbb{N}\}$ is a chain in $(D, \sqsubseteq)$. Moreover, if $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ are cpos then for any chain

$\{d_n \mid n \in \mathbb{N}\}$ in $(D, \sqsubseteq)$, we have

$$\bigsqcup_{n \in \mathbb{N}} f(d_n) \sqsubseteq' f(\bigsqcup_{n \in \mathbb{N}} d_n)$$

Indeed, since f is monotone and since $d_n \sqsubseteq \sqcup d_n$ for each $n \in \mathbb{N}$, it follows that $f(d_n) \sqsubseteq' f(\sqcup d_n)$ for each $n \in \mathbb{N}$. Therefore, $f(\sqcup d_n)$ is an upper bound for the chain $\{f(d_n) \mid n \in \mathbb{N}\}$. The rest follows because $\sqcup f(d_n)$ is the lub of $\{f(d_n) \mid n \in \mathbb{N}\}$.

Note that $\sqcup f(d_n) \sqsupseteq' f(\sqcup d_n)$ does not hold in general. Let, for example, $(D, \sqsubseteq)$ be $(\mathbb{N} \cup \{\infty\}, \leq)$, $(D', \sqsubseteq')$ be $(\{0, \infty\}, 0 \leq \infty)$, and f be the monotone function taking any natural number to 0 and ∞ to ∞. For the chain $\{n \mid n \in \mathbb{N}\}$, note that $\sqcup n = \infty$, so $f(\sqcup n) = \infty$. On the other hand, the chain $\{f(n) \mid n \in \mathbb{N}\}$ is stationary in 0, so $\sqcup f(n) = 0$.

One can think of a lub of a chain as a "limit" of that chain. Inspired by the analogous notion of continuous function in

mathematical analysis, which is characterized by the property of preserving limits, we say that a monotone function $f : (D, \sqsubseteq) \to (D', \sqsubseteq')$ is *continuous* if and only if $\sqcup f(d_n) \sqsupseteq' f(\sqcup d_n)$, which is equivalent to $\sqcup f(d_n) = f(\sqcup d_n)$, for any chain $\{d_n \mid n \in \mathbb{N}\}$ in $(D, \sqsubseteq)$.

The key property of the domain of partial functions, allowing us to use the general theory of cpos with continuous functions and especially the next fix-point theorem, is stated below as a homework exercise.

Homework Exercise 1 $(\Sigma \rightharpoonup \Sigma, \preceq, \bot)$ *is a cpo with bottom. Moreover, the functions* $\mathcal{F} : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ *naturally associated to while loops as shown in the previous lecture are continuous.*

The Fix-Point Theorem

Any monotone function $f : (D, \sqsubseteq, \bot) \to (D, \sqsubseteq, \bot)$ defined on a cpo with bottom to itself admits an implicit and important chain, namely $\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\bot) \sqsubseteq \cdot \sqsubseteq f^n(\bot) \sqsubseteq \cdots$, where f^n denotes n compositions of f with itself. The next is a key result in denotational semantics.

Theorem. *Let $(D, \sqsubseteq, \bot)$ be a cpo with bottom, let $f : (D, \sqsubseteq, \bot) \to (D, \sqsubseteq, \bot)$ be a continuous function, and let $\mathit{fix}(f)$ be the lub of the chain $\{f^n(\bot) \mid n \in \mathbb{N}\}$. Then $\mathit{fix}(f)$ is the least fix-point of f.*

Proof. We first show that $\mathit{fix}(f)$ is a fix-point of f:

$$
\begin{aligned}
f(\mathit{fix}(f)) \;&=\; f(\textstyle\bigsqcup_{n\in\mathbb{N}} f^n(\bot)) \\
&=\; \textstyle\bigsqcup_{n\in\mathbb{N}} f^{n+1}(\bot) \\
&=\; \textstyle\bigsqcup_{n\in\mathbb{N}} f^n(\bot) \\
&=\; \mathit{fix}(f).
\end{aligned}
$$

Next we show that $\mathit{fix}(f)$ is the least fix-point of f. Let d be another fix-point of f, that is, $f(d) = d$. We can show by induction that $f^n(\bot) \sqsubseteq d$ for any $n \in \mathbb{N}$: first note that $f^0(\bot) = \bot\bot \sqsubseteq d$; assume $f^n(\bot) \sqsubseteq d$ for some $n \in \mathbb{N}$; since f is monotone, it follows that $f(f^n(\bot)) \sqsubseteq f(d) = d$, that is, $f^{n+1}(\bot) \sqsubseteq d$. Thus d is an upper bound of the chain $\{f^n(\bot) \mid n \in \mathbb{N}\}$, so $\mathit{fix}(f) \sqsubseteq d$.

Corollary. *Functions* $\mathcal{F}: (\Sigma \rightharpoonup \Sigma) \rightarrow (\Sigma \rightharpoonup \Sigma)$ *associated to while loops as shown in the previous lecture admit fix-points.*

Proof. It follows by the fix-point theorem, thanks to the

properties proved in your homework exercise above.

Denotational Semantics of Loops

We are now ready to define the denotational semantics of loops.

If $\mathcal{F} : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ is the function associated to the statement $\texttt{while}(b)\ s$ as shown at the end of the previous lecture, then we define $[\![\texttt{while}(b)\ s]\!]$ to be $\mathit{fix}(\mathcal{F})$.

Therefore, we gave the denotational semantics of the entire simple imperative language. A natural question is whether this new semantics is equivalent to the other semantics.

Homework Exercise 2 *Formally state and prove the equivalence between the denotational semantics defined in this lecture notes and the small-step SOS for the same language defined in previous lecture notes.*