5

Denotational Semantics

In the operational approach, we were interested in how a program is executed. This is contrary to the denotational approach, where we are merely interested in the effect of executing a program. By effect we mean here an association between initial states and final states. The idea then is to define a semantic function for each syntactic category. It maps each syntactic construct to a mathematical object, often a function, that describes the effect of executing that construct.

The hallmark of denotational semantics is that semantic functions are defined compositionally; that is,

– there is a semantic clause for each of the basis elements of the syntactic category, and

– for each method of constructing a composite element (in the syntactic category) there is a semantic clause defined in terms of the semantic function applied to the immediate constituents of the composite element.

The functions $A$ and $B$ defined in Chapter 1 are examples of denotational definitions: the mathematical objects associated with arithmetic expressions are functions in $\text{State} \rightarrow \mathbb{Z}$ and those associated with boolean expressions are functions in $\text{State} \rightarrow \mathbb{T}$. The functions $S_{\text{ns}}$ and $S_{\text{ sos}}$ introduced in Chapter 2 associate mathematical objects with each statement, namely partial functions in $\text{State} \hookrightarrow \text{State}$. However, they are not examples of denotational definitions because they are not defined compositionally.
5.1 Direct Style Semantics: Specification

The effect of executing a statement $S$ is to change the state so we shall define the meaning of $S$ to be a partial function on states:

$$S_{ds}: Stm \rightarrow (\text{State} \leftrightarrow \text{State})$$

This is also the functionality of $S_{ns}$ and $S_{sos}$, and the need for partiality is again demonstrated by the statement \textbf{while true do skip}. The definition is summarized in Table 5.1 and we explain it in detail below; in particular, we shall define the auxiliary functions $\text{cond}$ and $\text{FIX}$.

For assignment, the clause

$$S_{ds}[x := a]s = s[x \mapsto A[a]]s$$

ensures that if $S_{ds}[x := a]s = s'$, then $s' x = A[a]s$ and $s' y = s y$ for $y \neq x$.

The clause for \textbf{skip} expresses that no state change takes place: the function $\text{id}$ is the identity function on \text{State} so $S_{ds}[\text{skip}]s = s$.

For sequencing, the clause is

$$S_{ds}[S_1 ; S_2] = S_{ds}[S_2] \circ S_{ds}[S_1]$$

So the effect of executing $S_1 ; S_2$ is the functional composition of the effect of executing $S_1$ and that of executing $S_2$. Functional composition is defined such that if one of the functions is undefined on a given argument, then their composition is undefined as well. Given a state $s$, we therefore have

\begin{tabular}{|c|c|}
\hline
$S_{ds}[x := a]s$ & $s[x \mapsto A[a]]s$ \\
$S_{ds}[\text{skip}]$ & $\text{id}$ \\
$S_{ds}[S_1 ; S_2]$ & $S_{ds}[S_2] \circ S_{ds}[S_1]$ \\
$S_{ds}[\text{if } b \text{ then } S_1 \text{ else } S_2]$ & $\text{cond}(B[b], S_{ds}[S_1], S_{ds}[S_2])$ \\
$S_{ds}[\text{while } b \text{ do } S]$ & $\text{FIX } F$ \\
$\text{where } F g$ & $\text{cond}(B[b], g \circ S_{ds}[S], \text{id})$ \\
\hline
\end{tabular}
5.1 Direct Style Semantics: Specification

\[ S_{ds}[S_1 ; S_2]s = (S_{ds}[S_2] \circ S_{ds}[S_1])s \]

\[
= \begin{cases} 
  s'' & \text{if there exists } s' \text{ such that } S_{ds}[S_1]s = s' \\
  & \text{and } S_{ds}[S_2]s' = s'' \\
  \text{undef} & \text{if } S_{ds}[S_1]s = \text{undef} \\
  & \text{or if there exists } s' \text{ such that } S_{ds}[S_1]s = s' \\
  & \text{but } S_{ds}[S_2]s' = \text{undef} 
\end{cases}
\]

It follows that the sequencing construct will only give a defined result if both components do.

For conditional, the clause is

\[ S_{ds}[\text{if } b \text{ then } S_1 \text{ else } S_2] = \text{cond}(B[b], S_{ds}[S_1], S_{ds}[S_2]) \]

and the auxiliary function cond has functionality

\[ \text{cond}: (\text{State} \to \text{T}) \times (\text{State} \hookrightarrow \text{State}) \times (\text{State} \hookrightarrow \text{State}) \to (\text{State} \hookrightarrow \text{State}) \]

and is defined by

\[
\text{cond}(p, g_1, g_2) s = \begin{cases} 
  g_1 s & \text{if } p s = \text{tt} \\
  g_2 s & \text{if } p s = \text{ff} 
\end{cases}
\]

The first parameter to cond is a function that, when supplied with an argument, will select either the second or the third parameter of cond and then supply that parameter with the same argument. Thus we have

\[ S_{ds}[\text{if } b \text{ then } S_1 \text{ else } S_2] s \]

\[
= \text{cond}(B[b], S_{ds}[S_1], S_{ds}[S_2]) s \]

\[
= \begin{cases} 
  s' & \text{if } B[b]s = \text{tt} \text{ and } S_{ds}[S_1]s = s' \\
  & \text{or if } B[b]s = \text{ff} \text{ and } S_{ds}[S_2]s = s' \\
  \text{undef} & \text{if } B[b]s = \text{tt} \text{ and } S_{ds}[S_1]s = \text{undef} \\
  & \text{or if } B[b]s = \text{ff} \text{ and } S_{ds}[S_2]s = \text{undef} 
\end{cases}
\]

So if the selected branch gives a defined result then so does the conditional. Note that since \( B[b] \) is a total function, \( B[b]s \) cannot be \text{undef}.

Defining the effect of while \( b \) do \( S \) is a major task. To motivate the actual definition, we first observe that the effect of while \( b \) do \( S \) must equal that of

\[ \text{if } b \text{ then } (S; \text{while } b \text{ do } S) \text{ else skip} \]
Using the parts of $S_{ds}$ that have already been defined, this gives

$$S_{ds}[\text{while } b \text{ do } S] = \text{cond}(B[b], S_{ds}[\text{while } b \text{ do } S] \circ S_{ds}[S], \text{id}) \quad (*)$$

Note that we cannot use (*) as the definition of $S_{ds}[\text{while } b \text{ do } S]$ because then $S_{ds}$ would not be a compositional definition. However, (*) expresses that $S_{ds}[\text{while } b \text{ do } S]$ must be a fixed point of the functional $F$ defined by

$$F g = \text{cond}(B[b], g \circ S_{ds}[S], \text{id})$$

that is $S_{ds}[\text{while } b \text{ do } S] = F (S_{ds}[\text{while } b \text{ do } S])$. In this way, we will get a compositional definition of $S_{ds}$ because when defining $F$ we only apply $S_{ds}$ to the immediate constituents of $\text{while } b \text{ do } S$ and not to the construct itself. Thus we write

$$S_{ds}[\text{while } b \text{ do } S] = \text{FIX } F$$

where $F g = \text{cond}(B[b], g \circ S_{ds}[S], \text{id})$ to indicate that $S_{ds}[\text{while } b \text{ do } S]$ is a fixed point of $F$. The functionality of the auxiliary function $\text{FIX}$ is

$$\text{FIX}: ((\text{State} \leftrightarrow \text{State}) \rightarrow (\text{State} \leftrightarrow \text{State})) \rightarrow (\text{State} \leftrightarrow \text{State})$$

**Example 5.1**

Consider the statement

$$\text{while } \neg(x = 0) \text{ do skip}$$

It is easy to verify that the corresponding functional $F'$ is defined by

$$(F' g) s = \begin{cases} 
g s & \text{if } s \ x \neq 0 
\\ s & \text{if } s \ x = 0 
\end{cases}$$

The function $g_1$ defined by

$$g_1 s = \begin{cases} 
\text{undef} & \text{if } s \ x \neq 0 
\\ s & \text{if } s \ x = 0 
\end{cases}$$

is a fixed point of $F'$ because

$$(F' g_1) s = \begin{cases} 
g_1 s & \text{if } s \ x \neq 0 
\\ s & \text{if } s \ x = 0 
\end{cases}$$

$$= \begin{cases} 
\text{undef} & \text{if } s \ x \neq 0 
\\ s & \text{if } s \ x = 0 
\end{cases}$$

$$= g_1 s$$
Next we claim that the function $g_2$ defined by
\[ g_2 s = \text{undef} \text{ for all } s \]
cannot be a fixed point for $F'$. The reason is that if $s'$ is a state with $s' x = 0$, then $(F' g_2) s' = s'$, whereas $g_2 s' = \text{undef}$.

Unfortunately, this does not suffice for defining $S_d[[\text{while } b \text{ do } S]]$. We face two problems:

– there are functionals that have more than one fixed point, and
– there are functionals that have no fixed point at all.

The functional $F'$ of Example 5.1 has more than one fixed point. In fact, every function $g'$ of $\text{State} \hookrightarrow \text{State}$ satisfying $g' s = s$ if $s x = 0$ will be a fixed point of $F'$.

For an example of a functional that has no fixed points, consider $F_1$ defined by
\[
F_1 g = \begin{cases} 
  g_1 & \text{if } g = g_2 \\
  g_2 & \text{otherwise}
\end{cases}
\]

If $g_1 \neq g_2$, then clearly there will be no function $g_0$ such that $F_1 g_0 = g_0$. Thus $F_1$ has no fixed points at all.

**Exercise 5.2**

Determine the functional $F$ associated with the statement
\[
\text{while } \neg(x=0) \text{ do } x := x - 1
\]
using the semantic equations of Table 5.1. Consider the following partial functions of $\text{State} \hookrightarrow \text{State}$:
\[
\begin{align*}
  g_1 s &= \text{undef} \text{ for all } s \\
  g_2 s &= \begin{cases} 
  s[x\mapsto 0] & \text{if } s x \geq 0 \\
  \text{undef} & \text{if } s x < 0
\end{cases} \\
  g_3 s &= \begin{cases} 
  s[x\mapsto 0] & \text{if } s x \geq 0 \\
  s & \text{if } s x < 0
\end{cases} \\
  g_4 s &= s[x\mapsto 0] \text{ for all } s \\
  g_5 s &= s \text{ for all } s
\end{align*}
\]
Determine which of these functions are fixed points of $F$. □
Exercise 5.3

Consider the following fragment of the factorial statement:

\[
\text{while } \neg(x = 1) \text{ do } (y := y \cdot x; \ x := x - 1)
\]

Determine the functional \( F \) associated with this statement. Determine at least two different fixed points for \( F \). □

Requirements on the Fixed Point

Our solution to the two problems listed above will be to develop a framework where

– we impose requirements on the fixed points and show that there is at most one fixed point fulfilling these requirements, and

– all functionals originating from statements in While do have a fixed point that satisfies these requirements.

To motivate our choice of requirements, let us consider the execution of a statement \( \text{while } b \text{ do } S \) from a state \( s_0 \). There are three possible outcomes:

A: It terminates.
B: It loops locally; that is, there is a construct in \( S \) that loops.
C: It loops globally; that is, the outer while-construct loops.

We shall now investigate what can be said about the functional \( F \) and its fixed points in each of the three cases.

The case A: In this case, the execution of \( \text{while } b \text{ do } S \) from \( s_0 \) terminates. This means that there are states \( s_1, \cdots, s_n \) such that

\[
B[b] s_i = \begin{cases} 
\text{tt} & \text{if } i < n \\
\text{ff} & \text{if } i = n
\end{cases}
\]

and

\[
S_{ds}[S] s_i = s_{i+1} \text{ for } i < n
\]

An example of a statement and a state satisfying these conditions is the statement

\[
\text{while } 0 \leq x \text{ do } x := x - 1
\]

and any state where \( x \) has a non-negative value.

Let \( g_0 \) be any fixed point of \( F \); that is, assume that \( F g_0 = g_0 \). In the case where \( i < n \), we calculate
\[ g_0 \cdot s_i = (F \cdot g_0) \cdot s_i \]
\[ = \text{cond}(B[b], g_0 \circ S_{ds}[S], \text{id}) \cdot s_i \]
\[ = g_0 \cdot (S_{ds}[S] \cdot s_i) \]
\[ = g_0 \cdot s_{i+1} \]

In the case where \( i=n \), we get
\[ g_0 \cdot s_n = (F \cdot g_0) \cdot s_n \]
\[ = \text{cond}(B[b], g_0 \circ S_{ds}[S], \text{id}) \cdot s_n \]
\[ = \text{id} \cdot s_n \]
\[ = s_n \]

Thus every fixed point \( g_0 \) of \( F \) will satisfy
\[ g_0 \cdot s_0 = s_n \]
so in this case we do not obtain any additional requirements that will help us to choose one of the fixed points as the preferred one.

The case \( B \): In this case, the execution of \textbf{while } \( b \) \textbf{do } \( S \) from \( s_0 \) loops \emph{locally}. This means that there are states \( s_1, \ldots, s_n \) such that
\[ B[b] \cdot s_i = \text{tt} \text{ for } i \leq n \]
and
\[ S_{ds}[S] \cdot s_i = \begin{cases} 
  s_{i+1} & \text{for } i < n \\
  \text{undef} & \text{for } i = n
\end{cases} \]

An example of a statement and a state satisfying these conditions is the statement
\[
\text{while } 0 \leq x \text{ do (if } x=0 \text{ then (while true do skip)} \text{ else } x := x-1) }
\]
and any state where \( x \) has a non-negative value.

Let \( g_0 \) be any fixed point of \( F \); that is, \( F \cdot g_0 = g_0 \). In the case where \( i<n \), we obtain
\[ g_0 \cdot s_i = g_0 \cdot s_{i+1} \]
just as in the previous case. However, in the case where \( i=n \), we get
\[ g_0 \cdot s_n = (F \cdot g_0) \cdot s_n \]
\[ = \text{cond}(B[b], g_0 \circ S_{ds}[S], \text{id}) \cdot s_n \]
\[ = (g_0 \circ S_{ds}[S]) \cdot s_n \]
\[ = \text{undef} \]
Thus any fixed point $g_0$ of $F$ will satisfy

$$g_0\ s_0 = \text{undef}$$

so, again, in this case we do not obtain any additional requirements that will help us to choose one of the fixed points as the preferred one.

**The case C:** The potential difference between fixed points comes to light when we consider the possibility that the execution of \textbf{while} $b$ \textbf{do} $S$ from $s_0$ loops \textit{globally}. This means that there is an infinite sequence of states $s_1, \ldots$ such that

$$B[b]s_i = \texttt{tt} \text{ for all } i$$

and

$$S_{db}[S]s_i = s_{i+1} \text{ for all } i.$$ 

An example of a statement and a state satisfying these conditions is the statement

\textbf{while} $(\neg(x=0))$ \textbf{do} \textbf{skip}

and any state where $x$ is not equal to $0$.

Let $g_0$ be any fixed point of $F$; that is, $F\ g_0 = g_0$. As in the previous cases, we get

$$g_0\ s_i = g_0\ s_{i+1}$$

for all $i \geq 0$. Thus we have

$$g_0\ s_0 = g_0\ s_i \text{ for all } i$$

and we cannot determine the value of $g_0\ s_0$ in this way. This is the situation in which the various fixed points of $F$ may differ.

This is not surprising because the statement \textbf{while} $(\neg(x=0))$ \textbf{do} \textbf{skip} of Example 5.1 has the functional $F'$ given by

$$(F'\ g)\ s = \begin{cases} g & \text{if } s\ x \neq 0 \\ s & \text{if } s\ x = 0 \end{cases}$$

and any partial function $g$ of $\text{State} \hookrightarrow \text{State}$ satisfying $g\ s = s$ if $s\ x = 0$ will indeed be a fixed point of $F'$. However, our computational experience tells us that we want

$$S_{db}[\textbf{while} \ (\neg(x=0)) \textbf{ do } \textbf{skip}]s_0 = \begin{cases} \text{undef} & \text{if } s_0\ x \neq 0 \\ s_0 & \text{if } s_0\ x = 0 \end{cases}$$

in order to record the looping. Thus our preferred fixed point of $F'$ is the function $g_0$ defined by
\[ g_0 \ s = \begin{cases} \text{undef} & \text{if } s \ x \neq 0 \\ s & \text{if } s \ x = 0 \end{cases} \]

The property that distinguishes \( g_0 \) from some other fixed point \( g' \) of \( F' \) is that whenever \( g_0 \ s = s' \) then we also have \( g' \ s = s' \) but not vice versa.

Generalizing this experience leads to the following requirement: the desired fixed point \( \text{FIX} \ F \) should be a partial function \( g_0 : \text{State} \hookrightarrow \text{State} \) such that

- \( g_0 \) is a fixed point of \( F \) (that is, \( F \ g_0 = g_0 \)), and
- if \( g \) is another fixed point of \( F \) (that is, \( F \ g = g \)), then
  \[ g_0 \ s = s' \text{ implies } g \ s = s' \]

for all choices of \( s \) and \( s' \).

Note that if \( g_0 \ s = \text{undef} \), then there are no requirements on \( g \ s \).

Exercise 5.4

Determine which of the fixed points considered in Exercise 5.2, if any, is the desired fixed point.

Exercise 5.5

Determine the desired fixed point of the functional from Exercise 5.3.

5.2 Fixed Point Theory

To prepare for a framework that guarantees the existence of the desired fixed point \( \text{FIX} \ F \), we shall reformulate the requirements to \( \text{FIX} \ F \) in a slightly more formal way. The first step will be to formalize the requirement that \( \text{FIX} \ F \) shares its results with all other fixed points. To do so, we define an ordering \( \sqsubseteq \) on partial functions of \( \text{State} \hookrightarrow \text{State} \). We set

\[ g_1 \sqsubseteq g_2 \]

when the partial function \( g_1 : \text{State} \hookrightarrow \text{State} \) shares its results with the partial function \( g_2 : \text{State} \hookrightarrow \text{State} \) in the sense that

\[ \text{if } g_1 \ s = s' \text{ then } g_2 \ s = s' \]

for all choices of \( s \) and \( s' \).
Example 5.6

Let \( g_1, g_2, g_3, \) and \( g_4 \) be partial functions in \( \text{State} \hookrightarrow \text{State} \) defined as follows:

\[
\begin{align*}
g_1(s) &= s \text{ for all } s \\
g_2(s) &= \begin{cases} s & \text{if } s \cdot x \geq 0 \\ \text{undef} & \text{otherwise} \end{cases} \\
g_3(s) &= \begin{cases} s & \text{if } s \cdot x = 0 \\ \text{undef} & \text{otherwise} \end{cases} \\
g_4(s) &= \begin{cases} s & \text{if } s \cdot x \leq 0 \\ \text{undef} & \text{otherwise} \end{cases}
\end{align*}
\]

Then we have

\[
\begin{align*}
g_1 &\sqsubseteq g_1, \\
g_2 &\sqsubseteq g_1, \; g_2 \sqsubseteq g_2, \\
g_3 &\sqsubseteq g_1, \; g_3 \sqsubseteq g_2, \; g_3 \sqsubseteq g_3, \; g_3 \sqsubseteq g_4, \; \text{and} \\
g_4 &\sqsubseteq g_1, \; g_4 \sqsubseteq g_4.
\end{align*}
\]

It is neither the case that \( g_2 \sqsubseteq g_4 \) nor that \( g_4 \sqsubseteq g_2 \). Pictorially, the ordering may be expressed by the following diagram (sometimes called a Hasse diagram):

\[
\begin{tikzpicture}
    \node (g1) at (0,1) {$g_1$};
    \node (g2) at (-1,-1) {$g_2$};
    \node (g3) at (1,-1) {$g_3$};
    \node (g4) at (0,-1) {$g_4$};
    \draw (g1) -- (g2);
    \draw (g1) -- (g3);
    \draw (g1) -- (g4);
\end{tikzpicture}
\]

The idea is that the smaller elements are at the bottom of the picture and that the lines indicate the order between the elements. However, we shall not draw lines when there already is a “broken line”, so the fact that \( g_3 \sqsubseteq g_1 \) is left implicit in the picture.

Exercise 5.7

Let \( g_1, g_2, \) and \( g_3 \) be defined as follows:

\[
\begin{align*}
g_1(s) &= \begin{cases} s & \text{if } s \cdot x \text{ is even} \\ \text{undef} & \text{otherwise} \end{cases}
\end{align*}
\]
\( g_2 s = \begin{cases} s & \text{if } s \times \text{ is a prime} \\ \text{undef} & \text{otherwise} \end{cases} \)

\( g_3 s = s \)

First, determine the ordering among these partial functions. Next, determine a partial function \( g_4 \) such that \( g_4 \sqsubseteq g_1, g_4 \sqsubseteq g_2, \) and \( g_4 \sqsubseteq g_3. \) Finally, determine a partial function \( g_5 \) such that \( g_1 \sqsubseteq g_5, g_2 \sqsubseteq g_5, \) and \( g_5 \sqsubseteq g_3 \) but \( g_5 \) is equal to neither \( g_1, g_2, \) nor \( g_3. \)

\[ \square \]

**Exercise 5.8 (Essential)**

An alternative characterization of the ordering \( \sqsubseteq \) on \( \text{State} \hookrightarrow \text{State} \) is

\[ g_1 \sqsubseteq g_2 \text{ if and only if } \text{graph}(g_1) \subseteq \text{graph}(g_2) \quad (*) \]

where \( \text{graph}(g) \) is the graph of the partial function \( g \) as defined in Appendix A. Prove that \( (*) \) is indeed correct.

\[ \square \]

The set \( \text{State} \hookrightarrow \text{State} \) equipped with the ordering \( \sqsubseteq \) is an example of a partially ordered set, as we shall see in Lemma 5.13 below. In general, a partially ordered set is a pair \( (D, \sqsubseteq_D) \), where \( D \) is a set and \( \sqsubseteq_D \) is a relation on \( D \) satisfying

\[
\begin{align*}
d \sqsubseteq_D d & \quad \text{(reflexivity)} \\
d_1 \sqsubseteq_D d_2 \text{ and } d_2 \sqsubseteq_D d_3 & \implies d_1 \sqsubseteq_D d_3 & \quad \text{(transitivity)} \\
d_1 \sqsubseteq_D d_2 \text{ and } d_2 \sqsubseteq_D d_1 & \implies d_1 = d_2 & \quad \text{(anti-symmetry)}
\end{align*}
\]

for all \( d, d_1 \) and \( d_1 \) in \( D. \)

The relation \( \sqsubseteq_D \) is said to be a partial order on \( D \) and we shall often omit the subscript \( D \) of \( \sqsubseteq_D \) and write \( \sqsubseteq. \) Occasionally, we may write \( d_1 \sqsubseteq d_2 \) instead of \( d_2 \sqsubseteq d_1, \) and we shall say that \( d_2 \) shares its information with \( d_1. \) An element \( d \) of \( D \) satisfying

\[ d \sqsubseteq d' \text{ for all } d' \text{ of } D \]

is called a least element of \( D, \) and we shall say that it contains no information.

**Fact 5.9**

If a partially ordered set \( (D, \sqsubseteq) \) has a least element \( d, \) then \( d \) is unique.

**Proof:** Assume that \( D \) has two least elements \( d_1 \) and \( d_2. \) Since \( d_1 \) is a least element, we have \( d_1 \sqsubseteq d_2. \) Since \( d_2 \) is a least element, we also have \( d_2 \sqsubseteq d_1. \)
The anti-symmetry of the ordering \( \sqsubseteq \) then gives that \( d_1 = d_2 \). 

This fact permits us to talk about the least element of \( D \), if one exists, and we shall denote it by \( \bot_D \) or simply \( \bot \) (pronounced “bottom”).

**Example 5.10**

For simplicity, let \( S \) be a non-empty set, and define 
\[
\mathcal{P}(S) = \{ K \mid K \subseteq S \}
\]
Then \( (\mathcal{P}(S), \subseteq) \) is a partially ordered set because
- \( \subseteq \) is reflexive: \( K \subseteq K \)
- \( \subseteq \) is transitive: if \( K_1 \subseteq K_2 \) and \( K_2 \subseteq K_3 \) then \( K_1 \subseteq K_3 \)
- \( \subseteq \) is anti-symmetric: if \( K_1 \subseteq K_2 \) and \( K_2 \subseteq K_1 \) then \( K_1 = K_2 \)

In the case where \( S = \{a,b,c\} \), the ordering can be depicted as follows:

Also, \( (\mathcal{P}(S), \subseteq) \) has a least element, namely \( \emptyset \).

**Exercise 5.11**

Show that \( (\mathcal{P}(S), \supseteq) \) is a partially ordered set, and determine the least element.

Draw a picture of the ordering when \( S = \{a,b,c\} \).

**Exercise 5.12**

Let \( S \) be a non-empty set, and define 
\[
\mathcal{P}_{\text{fin}}(S) = \{ K \mid K \text{ is finite and } K \subseteq S \}
\]
Verify that \((\mathcal{P}_{\text{fin}}(S), \subseteq)\) and \((\mathcal{P}_{\text{fin}}(S), \supseteq)\) are partially ordered sets. Do both partially ordered sets have a least element for all choices of \(S\)?

Lemma 5.13

\((\text{State} \hookrightarrow \text{State}, \subseteq)\) is a partially ordered set. The partial function \(\bot : \text{State} \hookrightarrow \text{State}\) defined by

\[\bot s = \text{undef}\]

for all \(s\) is the least element of \(\text{State} \hookrightarrow \text{State}\).

Proof: We shall first prove that \(\subseteq\) fulfils the three requirements to a partial order. Clearly, \(g \subseteq g\) holds because \(g s = s'\) trivially implies that \(g s = s'\) so \(\subseteq\) is a reflexive ordering.

To see that it is a transitive ordering, assume that \(g_1 \subseteq g_2\) and \(g_2 \subseteq g_3\) and we shall prove that \(g_1 \subseteq g_3\). Assume that \(g_1 s = s'\). From \(g_1 \subseteq g_2\), we get \(g_2 s = s'\), and then \(g_2 \subseteq g_3\) gives that \(g_3 s = s'\).

To see that it is an anti-symmetric ordering, assume that \(g_1 \subseteq g_2\) and \(g_2 \subseteq g_1\), and we shall then prove that \(g_1 = g_2\). Assume that \(g_1 s = s'\). Then \(g_2 s = s'\) follows from \(g_1 \subseteq g_2\), so \(g_1\) and \(g_2\) are equal on \(s\). If \(g_1 s = \text{undef}\), then it must be the case that \(g_2 s = \text{undef}\) since otherwise \(g_2 s = s'\) and the assumption \(g_2 \subseteq g_1\) then gives \(g_1 s = s'\), which is a contradiction. Thus \(g_1\) and \(g_2\) will be equal on \(s\).

Finally, we shall prove that \(\bot\) is the least element of \(\text{State} \hookrightarrow \text{State}\). It is easy to see that \(\bot\) is indeed an element of \(\text{State} \hookrightarrow \text{State}\), and it is also obvious that \(\bot \subseteq g\) holds for all \(g\) since \(\bot s = s'\) vacuously implies that \(g s = s'\).

Having introduced an ordering on the partial functions, we can now give a more precise statement of the requirements to \(\text{FIX} F\):

\(\text{–}\) \(\text{FIX} F\) is a fixed point of \(F\) (that is, \(F(\text{FIX} F) = \text{FIX} F\)), and

\(\text{–}\) \(\text{FIX} F\) is a least fixed point of \(F\); that is,

\[\text{if } F g = g \text{ then } \text{FIX} F \subseteq g.\]

Exercise 5.14

By analogy with Fact 5.9, show that if \(F\) has a least fixed point \(g_0\), then \(g_0\) is unique.
The next task will be to ensure that all functionals $F$ that may arise do indeed have least fixed points. We shall do so by developing a general theory that gives more structure to the partially ordered sets and that imposes restrictions on the functionals so that they have least fixed points.

Exercise 5.15
Determine the least fixed points of the functionals considered in Exercises 5.2 and 5.3. Compare them with Exercises 5.4 and 5.5.

Complete Partially Ordered Sets
Consider a partially ordered set $(D, \sqsubseteq)$ and assume that we have a subset $Y$ of $D$. We shall be interested in an element of $D$ that summarizes all the information of $Y$, and this is called an upper bound of $Y$; formally, it is an element $d$ of $D$ such that

$$\forall d' \in Y : d' \sqsubseteq d$$

An upper bound $d$ of $Y$ is a least upper bound if and only if $d'$ is an upper bound of $Y$ implies that $d \sqsubseteq d'$

Thus a least upper bound of $Y$ will add as little extra information as possible to that already present in the elements of $Y$.

Exercise 5.16
By analogy with Fact 5.9, show that if $Y$ has a least upper bound $d$, then $d$ is unique.

If $Y$ has a (necessarily unique) least upper bound, we shall denote it by $\bigsqcup Y$. Finally, a subset $Y$ is called a chain if it is consistent in the sense that if we take any two elements of $Y$, then one will share its information with the other; formally, this is expressed by

$$\forall d_1, d_2 \in Y : d_1 \sqsubseteq d_2 \text{ or } d_2 \sqsubseteq d_1$$

Example 5.17
Consider the partially ordered set $(\mathcal{P}(\{a,b,c\}), \subseteq)$ of Example 5.10. Then the subset
\[ Y_0 = \{ \emptyset, \{a\}, \{a,c\} \} \]
is a chain. Both \{a,b,c\} and \{a,c\} are upper bounds of \(Y_0\), and \{a,c\} is the least upper bound. The element \{a,b\} is not an upper bound because \{a,c\} \not\subseteq \{a,b\}. In general, the least upper bound of a non-empty chain in \(\mathcal{P}\{a,b,c\}\) will be the largest element of the chain.

The subset \{\emptyset, \{a\}, \{c\}, \{a,c\}\} is not a chain because \{a\} and \{c\} are unrelated by the ordering. However, it does have a least upper bound, namely \{a,c\}.

The subset \(\emptyset\) of \(\mathcal{P}\{a,b,c\}\) is a chain and has any element of \(\mathcal{P}\{a,b,c\}\) as an upper bound. Its least upper bound is the element \(\emptyset\).

**Exercise 5.18**

Let \(S\) be a non-empty set, and consider the partially ordered set \((\mathcal{P}(S), \subseteq)\). Show that every subset of \(\mathcal{P}(S)\) has a least upper bound. Repeat the exercise for the partially ordered set \((\mathcal{P}(S), \supseteq)\).

**Exercise 5.19**

Let \(S\) be a non-empty set, and consider the partially ordered set \((\mathcal{P}_{\text{fin}}(S), \subseteq)\) as defined in Exercise 5.12. Show by means of an example that there are choices of \(S\) such that \((\mathcal{P}_{\text{fin}}(S), \subseteq)\) has a chain with no upper bound and therefore no least upper bound.

**Example 5.20**

Let \(g_n: \text{State} \leftrightarrow \text{State}\) be defined by

\[
 g_n \ s = \begin{cases} 
 \text{undef} & \text{if } s \times > n \\
 s[x \mapsto -1] & \text{if } 0 \leq s \times \text{ and } s \times \leq n \\
 s & \text{if } s \times < 0 
\end{cases}
\]

It is straightforward to verify that \(g_n \subseteq g_m\) whenever \(n \leq m\) because \(g_n\) will be undefined for more states than \(g_m\). Now define \(Y_0\) to be

\[ Y_0 = \{ g_n \mid n \geq 0 \} \]

Then \(Y_0\) is a chain because \(g_n \subseteq g_m\) whenever \(n \leq m\). The partial function

\[
 g \ s = \begin{cases} 
 s[x \mapsto -1] & \text{if } 0 \leq s \times \\
 s & \text{if } s \times < 0 
\end{cases}
\]

is the least upper bound of \(Y\).
Exercise 5.21

Construct a subset $Y$ of $\text{State} \hookrightarrow \text{State}$ such that $Y$ has no upper bound and hence no least upper bound.

Exercise 5.22

Let $g_n$ be the partial function defined by

$$g_n \ s = \begin{cases} 
 s[y \mapsto (s \ x)!![x \mapsto 1] & \text{if } 0 < s \ x \text{ and } s \ x \leq n \\
 \text{undef} & \text{if } s \ x \leq 0 \text{ or } s \ x > n
\end{cases}$$  

(where $m!$ denotes the factorial of $m$). Define $Y_0 = \{ g_n \mid n \geq 0 \}$ and show that it is a chain. Characterize the upper bounds of $Y_0$ and determine the least upper bound.

A partially ordered set $(D, \sqsubseteq)$ is called a \textit{chain complete} partially ordered set (abbreviated \textit{ccpo}) whenever $\bigsqcup Y$ exists for all chains $Y$. It is a \textit{complete lattice} if $\bigsqcup Y$ exists for all subsets $Y$ of $D$.

Example 5.23

Exercise 5.18 shows that $(\mathcal{P}(S), \subseteq)$ and $(\mathcal{P}(S), \supseteq)$ are complete lattices; it follows that they both satisfy the ccpo-property. Exercise 5.19 shows that $(\mathcal{P}_{\text{fin}}(S), \subseteq)$ need not be a complete lattice nor a ccpo.

Fact 5.24

If $(D, \sqsubseteq)$ is a ccpo, then it has a least element $\bot$ given by $\bot = \bigsqcup \emptyset$.

Proof: It is straightforward to check that $\emptyset$ is a chain, and since $(D, \sqsubseteq)$ is a ccpo we get that $\bigsqcup \emptyset$ exists. Using the definition of $\bigsqcup \emptyset$, we see that for any element $d$ of $D$, we have $\bigsqcup \emptyset \sqsubseteq d$. This means that $\bigsqcup \emptyset$ is the least element of $D$.

Exercise 5.21 shows that $\text{State} \hookrightarrow \text{State}$ is not a complete lattice. Fortunately, we have the following lemma.

Lemma 5.25

$(\text{State} \hookrightarrow \text{State}, \sqsubseteq)$ is a ccpo. The least upper bound $\bigsqcup Y$ of a chain $Y$ is given by

$$\text{graph}(\bigsqcup Y) = \bigcup \{ \text{graph}(g) \mid g \in Y \}$$
that is, \((\bigcup Y)s = s'\) if and only if \(g s = s'\) for some \(g \in Y\).

Proof: The proof is in three parts. First we prove that 

\[
\bigcup \{ \text{graph}(g) \mid g \in Y \}
\]

is indeed a graph of a partial function in \(\text{State} \hookrightarrow \text{State} \). Second, we prove that this function will be an upper bound of \(Y\). Thirdly, we prove that it is less than any other upper bound of \(Y\); that is, it is the least upper bound of \(Y\).

To verify that (*) specifies a partial function, we only need to show that if \(\langle s, s' \rangle\) and \(\langle s, s'' \rangle\) are elements of 

\[
X = \bigcup \{ \text{graph}(g) \mid g \in Y \}
\]

then \(s' = s''\). When \(\langle s, s' \rangle \in X\), there will be a partial function \(g \in Y\) such that \(g s = s'\). Similarly, when \(\langle s, s'' \rangle \in X\), then there will be a partial function \(g' \in Y\) such that \(g' s = s''\). Since \(Y\) is a chain, we will have that either \(g \sqsubseteq g'\) or \(g' \sqsubseteq g\). In any case, we get \(g s = g' s\), and this means that \(s' = s''\) as required. This completes the first part of the proof.

In the second part of the proof, we define the partial function \(g_0\) by

\[
\text{graph}(g_0) = \bigcup \{ \text{graph}(g) \mid g \in Y \}
\]

To show that \(g_0\) is an upper bound of \(Y\), let \(g\) be an element of \(Y\). Then we have \(\text{graph}(g) \subseteq \text{graph}(g_0)\), and using the result of Exercise 5.8 we see that \(g \sqsubseteq g_0\) as required and we have completed the second part of the proof.

In the third part of the proof, we show that \(g_0\) is the least upper bound of \(Y\). So let \(g_1\) be some upper bound of \(Y\). Using the definition of an upper bound, we get that \(g \sqsubseteq g_1\) must hold for all \(g \in Y\). Exercise 5.8 gives that \(\text{graph}(g) \subseteq \text{graph}(g_1)\). Hence it must be the case that

\[
\bigcup \{ \text{graph}(g) \mid g \in Y \} \subseteq \text{graph}(g_1)
\]

But this is the same as \(\text{graph}(g_0) \subseteq \text{graph}(g_1)\), and Exercise 5.8 gives that \(g_0 \sqsubseteq g_1\). This shows that \(g_0\) is the least upper bound of \(Y\) and thereby we have completed the proof.

Continuous Functions

Let \((D, \sqsubseteq)\) and \((D', \sqsubseteq')\) satisfy the ccpo-property, and consider a (total) function \(f: D \rightarrow D'\). If \(d_1 \sqsubseteq d_2\), then the intuition is that \(d_1\) shares its information with \(d_2\). So when the function \(f\) has been applied to the two elements \(d_1\) and \(d_2\), we shall expect that a similar relationship holds between the results. That
is, we shall expect that \( f d_1 \sqsubseteq f d_2 \), and when this is the case we say that \( f \) is monotone. Formally, \( f \) is monotone if and only if

\[
d_1 \sqsubseteq d_2 \text{ implies } f d_1 \sqsubseteq f d_2
\]

for all choices of \( d_1 \) and \( d_2 \).

**Example 5.26**

Consider \((P(\{a,b,c\}), \subseteq)\) and \((P(\{d,e\}), \subseteq)\). The function \( f_1: P(\{a,b,c\}) \rightarrow P(\{d,e\}) \) defined by the table

\[
\begin{array}{c|cccccc}
X & \{a,b,c\} & \{a\} & \{b\} & \{c\} & \emptyset \\
\hline
f_1 X & \{d,e\} & \{d\} & \{d,e\} & \{d\} & \{e\} & \emptyset
\end{array}
\]

is monotone: it simply changes a's and b's to d's and c's to e's.

The function \( f_2: P(\{a,b,c\}) \rightarrow P(\{d,e\}) \) defined by the table

\[
\begin{array}{c|cccccc}
X & \{a,b,c\} & \{a\} & \{b\} & \{c\} & \emptyset \\
\hline
f_2 X & \{d\} & \{d\} & \{e\} & \{e\} & \{e\}
\end{array}
\]

is not monotone because \( \{b,c\} \subseteq \{a,b,c\} \) but \( f_2 \{b,c\} \nsubseteq f_2 \{a,b,c\} \). Intuitively, all sets that contain an a are mapped to \( \{d\} \), whereas the others are mapped to \( \{e\} \), and since the elements \( \{d\} \) and \( \{e\} \) are incomparable this does not give a monotone function. However, if we change the definition such that sets with an a are mapped to \( \{d\} \) and all other sets to \( \emptyset \), then the function will indeed be monotone.  

**Exercise 5.27**

Consider \((\mathcal{P}(\mathbb{N}), \subseteq)\). Determine which of the following functions in \( \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(\mathbb{N}) \) are monotone:

\[
\begin{align*}
f_1 X &= \mathbb{N} \setminus X \\
f_2 X &= X \cup \{27\} \\
f_3 X &= X \cap \{7, 9, 13\} \\
f_4 X &= \{ n \in X \mid n \text{ is a prime} \} \\
f_5 X &= \{ 2 \cdot n \mid n \in X \}
\end{align*}
\]

**Exercise 5.28**

Determine which of the following functionals of

\[
(\text{State} \hookrightarrow \text{State}) \rightarrow (\text{State} \hookrightarrow \text{State})
\]
are monotone:

\[
F_0 g = g \\
F_1 g = \begin{cases} 
g_1 & \text{if } g = g_2 \\
g_2 & \text{otherwise} \end{cases} \quad \text{where } g_1 \neq g_2 \\
(F') s = \begin{cases} 
g s & \text{if } s \neq 0 \\
s & \text{if } s = 0 \end{cases}
\]

The monotone functions have a couple of interesting properties. First we prove that the composition of two monotone functions is a monotone function.

**Fact 5.29**

Let \((D, \sqsubseteq), (D', \sqsubseteq'), (D'', \sqsubseteq'')\) satisfy the ccpo-property, and let \(f: D \to D'\) and \(f': D' \to D''\) be monotone functions. Then \(f' \circ f: D \to D''\) is a monotone function.

**Proof:** Assume that \(d_1 \sqsubseteq d_2\). The monotonicity of \(f\) gives that \(f d_1 \sqsubseteq' f d_2\). The monotonicity of \(f'\) then gives \(f' (f d_1) \sqsubseteq'' f' (f d_2)\) as required. \(\Box\)

Next we prove that the image of a chain under a monotone function is itself a chain.

**Lemma 5.30**

Let \((D, \sqsubseteq)\) and \((D', \sqsubseteq')\) satisfy the ccpo-property, and let \(f: D \to D'\) be a monotone function. If \(Y\) is a chain in \(D\), then \(\{ f d \mid d \in Y \}\) is a chain in \(D'\). Furthermore,

\[
\bigsqcup \{ f d \mid d \in Y \} \sqsubseteq' f(\bigsqcup Y)
\]

**Proof:** If \(Y = \emptyset\), then the result holds immediately since \(\bot' \sqsubseteq' f \bot\); so for the rest of the proof we may assume that \(Y \neq \emptyset\). We shall first prove that \(\{ f d \mid d \in Y \}\) is a chain in \(D'\). So let \(d_1'\) and \(d_2'\) be two elements of \(\{ f d \mid d \in Y \}\). Then there are elements \(d_1\) and \(d_2\) in \(Y\) such that \(d_1' = f d_1\) and \(d_2' = f d_2\). Since \(Y\) is a chain, we have that either \(d_1 \sqsubseteq d_2\) or \(d_2 \sqsubseteq d_1\). In either case, we get that the same order holds between \(d_1'\) and \(d_2'\) because of the monotonicity of \(f\). This proves that \(\{ f d \mid d \in Y \}\) is a chain.

To prove the second part of the lemma, consider an arbitrary element \(d\) of \(Y\). Then it is the case that \(d \sqsubseteq \bigsqcup Y\). The monotonicity of \(f\) gives that \(f d \sqsubseteq' f(\bigsqcup Y)\). Since this holds for all \(d \in Y\), we get that \(f(\bigsqcup Y)\) is an upper bound on \(\{ f d \mid d \in Y \}\); that is, \(\bigsqcup \{ f d \mid d \in Y \} \sqsubseteq' f(\bigsqcup Y)\). \(\Box\)
In general, we cannot expect that a monotone function preserves least upper bounds on chains; that is, \( \bigsqcup \{ f \, d \mid d \in Y \} = f(\bigsqcup Y) \). This is illustrated by the following example.

**Example 5.31**
From Example 5.23, we get that \( (\mathcal{P}(\mathbb{N} \cup \{a\}), \subseteq) \) is a ccpo. Now consider the function \( f: \mathcal{P}(\mathbb{N} \cup \{a\}) \to \mathcal{P}(\mathbb{N} \cup \{a\}) \) defined by

\[
f X = \begin{cases} X & \text{if } X \text{ is finite} \\ X \cup \{a\} & \text{if } X \text{ is infinite} \end{cases}
\]

Clearly, \( f \) is a monotone function: if \( X_1 \subseteq X_2 \), then also \( f X_1 \subseteq f X_2 \). However, \( f \) does not preserve the least upper bounds of chains. To see this, consider the set

\[
Y = \{ \{0,1,\cdots,n\} \mid n \geq 0 \}
\]

It consists of the elements \( \{0\} \), \( \{0,1\} \), \( \{0,1,2\} \), \( \cdots \) and it is straightforward to verify that it is a chain with \( \mathbb{N} \) as its least upper bound; that is, \( \bigsqcup Y = \mathbb{N} \). When we apply \( f \) to the elements of \( Y \), we get

\[
\bigsqcup \{ f \, X \mid X \in Y \} = \bigsqcup Y = \mathbb{N}
\]

However, we also have

\[
f (\bigsqcup Y) = f \, \mathbb{N} = \mathbb{N} \cup \{a\}
\]

showing that \( f \) does not preserve the least upper bounds of chains.

We shall be interested in functions that preserve least upper bounds of chains; that is, functions \( f \) that satisfy

\[
\bigsqcup \{ f \, d \mid d \in Y \} = f(\bigsqcup Y)
\]

Intuitively, this means that we obtain the same information independently of whether we determine the least upper bound before or after applying the function \( f \).

We shall say that a function \( f: D \to D' \) defined on \( (D, \sqsubseteq) \) and \( (D', \sqsubseteq') \) is *continuous* if it is monotone and

\[
\bigsqcup \{ f \, d \mid d \in Y \} = f(\bigsqcup Y)
\]

holds for all *non-empty* chains \( Y \). If \( \bigsqcup \{ f \, d \mid d \in Y \} = f(\bigsqcup Y) \) holds for the empty chain (that is, \( \bot = f \, \bot \)), then we shall say that \( f \) is *strict*. 
Example 5.32

The function \( f_1 \) of Example 5.26 is also continuous. To see this, consider a non-empty chain \( Y \) in \( P(\{a,b,c\}) \). The least upper bound of \( Y \) will be the largest element, say \( X_0 \), of \( Y \) (see Example 5.17). Therefore we have

\[
f_1 \left( \bigcup Y \right) = f_1 \left( X_0 \right) \quad \text{because } X_0 = \bigcup Y
\]

\[
\subseteq \bigcup \{ f_1 X \mid X \in Y \} \quad \text{because } X_0 \in Y
\]

Using that \( f_1 \) is monotone, we get from Lemma 5.30 that \( \bigcup \{ f_1 X \mid X \in Y \} \subseteq f_1 \left( \bigcup Y \right) \). It follows that \( f_1 \) is continuous. Also, \( f_1 \) is a strict function because \( f_1 \emptyset = \emptyset \).

The function \( f \) of Example 5.31 is not a continuous function because there is a chain for which it does not preserve the least upper bound. \( \square \)

Exercise 5.33

Show that the functional \( F' \) of Example 5.1 is continuous. \( \square \)

Exercise 5.34

Assume that \( (D, \sqsubseteq) \) and \( (D', \sqsubseteq') \) satisfy the ccpo-property, and assume that the function \( f: D \rightarrow D' \) satisfies

\[
\bigcup' \{ f \ d \mid d \in Y \} = f(\bigcup Y)
\]

for all non-empty chains \( Y \) of \( D \). Show that \( f \) is monotone. \( \square \)

We can extend the result of Lemma 5.29 to show that the composition of two continuous functions will also be continuous, as follows.

Lemma 5.35

Let \( (D, \sqsubseteq) \), \( (D', \sqsubseteq') \), and \( (D'', \sqsubseteq'') \) satisfy the ccpo-property, and let the functions \( f: D \rightarrow D' \) and \( f': D' \rightarrow D'' \) be continuous. Then \( f' \circ f: D \rightarrow D'' \) is a continuous function.

Proof: From Fact 5.29 we get that \( f' \circ f \) is monotone. To prove that it is continuous, let \( Y \) be a non-empty chain in \( D \). The continuity of \( f \) gives

\[
\bigcup' \{ f \ d \mid d \in Y \} = f(\bigcup Y)
\]

Since \( \{ f \ d \mid d \in Y \} \) is a (non-empty) chain in \( D' \), we can use the continuity of \( f' \) and get
\[
\bigcup'' \{ f' \ d' \mid d' \in \{ f \ d \mid d \in Y \} \} = f' \ (\bigcup' \{ f \ d \mid d \in Y \})
\]

which is equivalent to
\[
\bigcup'' \{ f' \ (f \ d) \mid d \in Y \} = f' \ (f \ (\bigcup Y))
\]
This proves the result. \qed

**Exercise 5.36**

Prove that if \( f \) and \( f' \) are strict functions, then so is \( f' \circ f \).

We can now define the required fixed point operator \( \text{FIX} \) as follows.

**Theorem 5.37**

Let \( f: D \rightarrow D \) be a continuous function on the ccpo \((D, \sqsubseteq)\) with least element \( \bot \). Then
\[
\text{FIX} f = \bigcup \{ f^n \bot \mid n \geq 0 \}
\]
defines an element of \( D \), and this element is the least fixed point of \( f \).

Here we have used the notation that \( f^0 = \text{id} \), and \( f^{n+1} = f \circ f^n \) for \( n \geq 0 \).

**Proof:** We first show the well-definedness of \( \text{FIX} f \). Note that \( f^0 \bot = \bot \) and that \( \bot \subseteq d \) for all \( d \in D \). By induction on \( n \), one may show that
\[
f^n \bot \subseteq f^n d
\]
for all \( d \in D \) since \( f \) is monotone. It follows that \( f^n \bot \subseteq f^m \bot \) whenever \( n \leq m \). Hence \( \{ f^n \bot \mid n \geq 0 \} \) is a (non-empty) chain in \( D \), and \( \text{FIX} f \) exists because \( D \) is a ccpo.

We next show that \( \text{FIX} f \) is a fixed point; that is, \( f \ (\text{FIX} f) = \text{FIX} f \). We calculate
\[
f \ (\text{FIX} f) = f \ (\bigcup \{ f^n \bot \mid n \geq 0 \}) \quad \text{(definition of \( \text{FIX} f \))}
= \bigcup \{ f(f^n \bot) \mid n \geq 0 \} \quad \text{(continuity of \( f \))}
= \bigcup \{ f^n \bot \mid n \geq 1 \}
= \bigcup \{ f^n \bot \mid n \geq 1 \} \cup \{ \bot \} \quad \text{(for all chains \( Y \))}
= \bigcup \{ f^n \bot \mid n \geq 0 \} \quad \text{(for all chains \( Y \))}
= \text{FIX} f \quad \text{(definition of \( \text{FIX} f \))}
\]
To see that \( \text{FIX} f \) is the least fixed point, assume that \( d \) is some other fixed point. Clearly \( \bot \sqsubseteq d \) so the monotonicity of \( f \) gives \( f^n \bot \sqsubseteq f^n d \) for \( n \geq 0 \), and as \( d \) was a fixed point, we obtain \( f^n \bot \sqsubseteq d \) for all \( n \geq 0 \). Hence \( d \) is an upper bound of the chain \( \{ f^n \bot | n \geq 0 \} \), and using that \( \text{FIX} f \) is the least upper bound, we have \( \text{FIX} f \sqsubseteq d \). \( \square \)

**Example 5.38**

Consider the function \( F' \) of Example 5.1:

\[
(F' g) s = \begin{cases} 
  g & s \neq 0 \\
  s & s = 0 
\end{cases}
\]

We shall determine its least fixed point using the approach of Theorem 5.37. The least element \( \bot \) of \( \text{State} \hookrightarrow \text{State} \) is given by Lemma 5.13 and has \( \bot s = \text{undef} \) for all \( s \). We then determine the elements of the set \( \{ F'^n \bot | n \geq 0 \} \) as follows:

\[
(F'^0 \bot) s = (\text{id} \bot) s = \text{id}(\bot) s = \text{undef} \quad \text{(definition of } F'^0 \bot) \\
(F'^1 \bot) s = (F' \bot) s = \begin{cases} 
  \bot & s \neq 0 \\
  s & s = 0 
\end{cases} \quad \text{(definition of } F'^1 \bot) \\
= \begin{cases} 
  \text{undef} & s \neq 0 \\
  s & s = 0 
\end{cases} \quad \text{(definition of } \bot) \\
(F'^2 \bot) s = (F' (F'^1 \bot)) s = (F' (\bot)) s = \begin{cases} 
  (F'^1 \bot) s & s \neq 0 \\
  s & s = 0 
\end{cases} \quad \text{(definition of } F' \text{)} \\
= \begin{cases} 
  \text{undef} & s \neq 0 \\
  s & s = 0 
\end{cases} \quad \text{(definition of } F'^1 \bot) \\
\vdots
\]

In general, we have \( F'^n \bot = F'^{n+1} \bot \) for \( n > 0 \). Therefore

\[
\bigsqcup \{ F'^n \bot | n \geq 0 \} = \bigsqcup \{ F'^0 \bot, F'^1 \bot \} = F'^1 \bot
\]

because \( F'^0 \bot = \bot \). Thus the least fixed point of \( F' \) will be the function
\[ g_1 s = \begin{cases} \text{undef} & \text{if } s \neq 0 \\ s & \text{if } s = 0 \end{cases} \]

Exercise 5.39

Redo Exercise 5.15 using the approach of Theorem 5.37; that is, deduce the general form of the iterands, \( F^n \perp \), for the functional, \( F \), of Exercises 5.2 and 5.3.

Exercise 5.40 (Essential)

Let \( f : D \to D \) be a continuous function on a ccpo \((D, \sqsubseteq)\) and let \( d \in D \) satisfy \( f d \sqsubseteq d \). Show that \( \text{FIX} f \sqsubseteq d \).

The table below summarizes the development we have performed in order to demonstrate the existence of least fixed points:

<table>
<thead>
<tr>
<th>Fixed Point Theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: We restrict ourselves to chain complete partially ordered sets (abbreviated ccpo).</td>
</tr>
<tr>
<td>2: We restrict ourselves to continuous functions on chain complete partially ordered sets.</td>
</tr>
<tr>
<td>3: We show that continuous functions on chain complete partially ordered sets always have least fixed points (Theorem 5.37).</td>
</tr>
</tbody>
</table>

Exercise 5.41 (*)

Let \((D, \sqsubseteq)\) be a ccpo and define \((D \to D, \sqsubseteq')\) by setting
\[ f_1 \sqsubseteq' f_2 \text{ if and only if } f_1 d \sqsubseteq f_2 d \text{ for all } d \in D \]
Show that \((D \to D, \sqsubseteq')\) is a ccpo and that \( \text{FIX} \) is “continuous” in the sense that
\[ \text{FIX}(\bigsqcup \mathcal{F}) = \bigsqcup \{ \text{FIX} f \mid f \in \mathcal{F} \} \]
holds for all non-empty chains \( \mathcal{F} \subseteq D \to D \) of continuous functions.
Exercise 5.42 (** For Mathematicians)

Given a ccpo \((D, \sqsubseteq)\), we define an open set of \(D\) to be a subset \(Y\) of \(D\) satisfying

1. if \(d_1 \in Y\) and \(d_1 \sqsubseteq d_2\) then \(d_2 \in Y\), and
2. if \(Y'\) is a non-empty chain satisfying \(\bigcup Y' \in Y\), then there exists an element \(d\) of \(Y'\) that also is an element of \(Y\).

The set of open sets of \(D\) is denoted \(O_D\). Show that this is indeed a topology on \(D\); that is, show that
- \(\emptyset\) and \(D\) are members of \(O_D\),
- the intersection of two open sets is an open set, and
- the union of any collection of open sets is an open set.

Let \((D, \sqsubseteq)\) and \((D', \sqsubseteq')\) satisfy the ccpo-property. A function \(f : D \to D'\) is topologically continuous if and only if the function \(f^{-1} : P(D') \to P(D)\) defined by

\[
  f^{-1}(Y') = \{ d \in D \mid f(d) \in Y' \}
\]

maps open sets to open sets; that is, specializes to \(f^{-1} : O_{D'} \to O_D\). Show that \(f\) is a continuous function between \(D\) and \(D'\) if and only if it is a topologically continuous function between \(D\) and \(D'\).  

5.3 Direct Style Semantics: Existence

We have now obtained the mathematical foundations needed to prove that the semantic clauses of Table 5.1 do indeed define a function. So consider once again the clause

\[
  S_{ds} \llbracket \text{while } b \text{ do } S \rrbracket = \text{FIX } F
\]

where \(F \ g = \text{cond}(B[b], g \circ S_{ds}[S], \text{id})\)

For this to make sense, we must show that \(F\) is continuous. To do so, we first observe that

\[
  F \ g = F_1 (F_2 \ g)
\]

where

\[
  F_1 \ g = \text{cond}(B[b], g, \text{id})
\]

and
\[ F_2 g = g \circ S_{ds}[S] \]

Using Lemma 5.35, we then obtain the continuity of \( F \) by showing that \( F_1 \) and \( F_2 \) are continuous. We shall first prove that \( F_1 \) is continuous:

**Lemma 5.43**

Let \( g_0 : \text{State} \hookrightarrow \text{State} \), \( p : \text{State} \rightarrow T \), and define

\[ F g = \text{cond}(p, g, g_0) \]

Then \( F \) is continuous.

**Proof:** We shall first prove that \( F \) is monotone. So assume that \( g_1 \sqsubseteq g_2 \) and we shall show that \( F g_1 \sqsubseteq F g_2 \). It suffices to consider an arbitrary state \( s \) and show that

\[ (F g_1) s = s' \text{ implies } (F g_2) s = s' \]

If \( p s = \text{tt} \), then \( (F g_1) s = g_1 s \), and from \( g_1 \sqsubseteq g_2 \) we get that \( g_1 s = s' \) implies \( g_2 s = s' \). Since \( (F g_2) s = g_2 s \), we have proved the result. So consider the case where \( p s = \text{ff} \). Then \( (F g_1) s = g_0 s \) and similarly \( (F g_2) s = g_0 s \) and the result is immediate.

To prove that \( F \) is continuous, we shall let \( Y \) be a non-empty chain in \( \text{State} \hookrightarrow \text{State} \). We must show that

\[ F (\bigsqcup Y) \sqsubseteq \bigsqcup \{ F g \mid g \in Y \} \]

since \( F (\bigsqcup Y) \supseteq \bigsqcup \{ F g \mid g \in Y \} \) follows from the monotonicity of \( F \) (see Lemma 5.30). Thus we have to show that

\[ \text{graph}(F(\bigsqcup Y)) \subseteq \bigsqcup \{ \text{graph}(F g) \mid g \in Y \} \]

using the characterization of least upper bounds of chains in \( \text{State} \hookrightarrow \text{State} \) given in Lemma 5.25. So assume that \( (F (\bigsqcup Y)) s = s' \) and let us determine \( g \in Y \) such that \( (F g) s = s' \). If \( p s = \text{ff} \), we have \( F (\bigsqcup Y) s = g_0 s = s' \) and clearly, for every element \( g \) of the non-empty set \( Y \) we have \( (F g) s = g_0 s = s' \). If \( p s = \text{tt} \), then we get \( (F (\bigsqcup Y)) s = (\bigsqcup Y) s = s' \) so \( (s, s') \in \text{graph}(\bigsqcup Y) \). Since

\[ \text{graph}(\bigsqcup Y) = \bigsqcup \{ \text{graph}(g) \mid g \in Y \} \]

(according to Lemma 5.25), we therefore have \( g \in Y \) such that \( g s = s' \), and it follows that \( (F g) s = s' \). This proves the result. \( \square \)
Exercise 5.44 (Essential)

Prove that (in the setting of Lemma 5.43) $F$ defined by $F g = \text{cond}(p, g_0, g)$ is continuous; that is, cond is continuous in its second and third arguments.

Lemma 5.45

Let $g_0: \text{State} \hookrightarrow \text{State}$, and define

$$F g = g \circ g_0$$

Then $F$ is continuous.

Proof: We first prove that $F$ is monotone. If $g_1 \subseteq g_2$, then $\text{graph}(g_1) \subseteq \text{graph}(g_2)$ according to Exercise 5.8, so that $\text{graph}(g_0) \circ \text{graph}(g_1)$, which is the relational composition of $\text{graph}(g_0)$ and $\text{graph}(g_1)$ (see Appendix A), satisfies $\text{graph}(g_0) \circ \text{graph}(g_1) \subseteq \text{graph}(g_0) \circ \text{graph}(g_2)$ and this shows that $F_1 \subseteq F_2$. Next we shall prove that $F$ is continuous. If $Y$ is a non-empty chain, then

$$\text{graph}(F(\bigsqcup Y)) = \text{graph}((\bigsqcup Y) \circ g_0)$$

$$= \text{graph}(g_0) \circ \text{graph}(\bigsqcup Y)$$

$$= \text{graph}(g_0) \circ \bigcup\{\text{graph}(g) \mid g \in Y\}$$

$$= \bigcup\{\text{graph}(g_0) \circ \text{graph}(g) \mid g \in Y\}$$

$$= \text{graph}(\bigsqcup\{F g \mid g \in Y\})$$

where we have used Lemma 5.25 twice. Thus $F(\bigsqcup Y) = \bigsqcup\{F g \mid g \in Y\}$.

Exercise 5.46 (Essential)

Prove that (in the setting of Lemma 5.45) $F$ defined by $F g = g_0 \circ g$ is continuous; that is, $\circ$ is continuous in both arguments.

We have now established the results needed to show that the equations of Table 5.1 define a function $S_{ds}$ as follows.

Proposition 5.47

The semantic equations of Table 5.1 define a total function $S_{ds}$ in $\text{Stm} \rightarrow (\text{State} \hookrightarrow \text{State})$. 
Proof: The proof is by structural induction on the statement \( S \).

The case \( x := a \): Clearly the function that maps a state \( s \) to the state \( s[x \mapsto A[a]]s \) is well-defined.

The case \( \text{skip} \): Clearly the function \( \text{id} \) is well-defined.

The case \( S_1 ; S_2 \): The induction hypothesis gives that \( S_{ds}[S_1] \) and \( S_{ds}[S_2] \) are well-defined, and clearly their composition will be well-defined.

The case \( \text{if } b \text{ then } S_1 \text{ else } S_2 \): The induction hypothesis gives that \( S_{ds}[S_1] \) and \( S_{ds}[S_2] \) are well-defined functions, and clearly this property is preserved by the function \( \text{cond} \).

The case \( \text{while } b \text{ do } S \): The induction hypothesis gives that \( S_{ds}[S] \) is well-defined. The functions \( F_1 \) and \( F_2 \) defined by

\[
F_1 \ g = \text{cond}(\mathcal{B}[b]. \ g, \ \text{id})
\]

and

\[
F_2 \ g = g \circ S_{ds}[S]
\]

are continuous according to Lemmas 5.43 and 5.45. Thus Lemma 5.35 gives that \( F \ g = F_1 \ (F_2 \ g) \) is continuous. From Theorem 5.37, we then have that \( \text{FIX} \ F \) is well-defined and thereby that \( S_{ds}[\text{while } b \text{ do } S] \) is well-defined. This completes the proof.

Example 5.48

Consider the denotational semantics of the factorial statement:

\[
S_{ds}[y := 1; \ \text{while } \neg(x = 1) \ do \ (y := y \star x; \ x := x - 1)]
\]

We shall be interested in applying this function to a state \( s_0 \) where \( x \) has the value 3. To do that, we shall first apply the clauses of Table 5.1, and we then get that

\[
S_{ds}[y := 1; \ \text{while } \neg(x = 1) \ do \ (y := y \star x; \ x := x - 1)] \ s_0 = (\text{FIX} \ F) \ s_0[y\mapsto 1]
\]

where

\[
F \ g \ s = \begin{cases} 
\ g \ (S_{ds}[y := y \star x; \ x := x - 1] \ s) & \text{ if } \mathcal{B}[\neg(x = 1)] \ s = \text{tt} \\
\ s & \text{ if } \mathcal{B}[\neg(x = 1)] \ s = \text{ff}
\end{cases}
\]

or, equivalently,
5.3 Direct Style Semantics: Existence

\[ F \ g \ s = \begin{cases} 
  g \ (s[y \mapsto (s \ y) \cdot (s \ x)]\{x \mapsto (s \ x) - 1\}) & \text{if } s \ x \neq 1 \\
  s & \text{if } s \ x = 1 
\end{cases} \]

We can now calculate the various functions \( F^n \bot \) used in the definition of \( \text{FIX} \ F \) in Theorem 5.37:

\[
(F^0 \bot) \ s = \text{undef} \\
(F^1 \bot) \ s = \begin{cases} 
  \text{undef} & \text{if } s \ x \neq 1 \\
  s & \text{if } s \ x = 1 
\end{cases} \\
(F^2 \bot) \ s = \begin{cases} 
  \text{undef} & \text{if } s \ x \neq 1 \text{ and } s \ x \neq 2 \\
  s[y \mapsto (s \ y) \cdot 2][x \mapsto 1] & \text{if } s \ x = 2 \\
  s & \text{if } s \ x = 1 
\end{cases}
\]

Thus, if \( x \) is 1 or 2, then \( F^2 \bot \) will give the correct value for \( y \), and for all other values of \( x \) the result is undefined. This is a general pattern: the \( n \)'th \( \text{iterand} \) \( F^n \bot \) will determine the correct value if it can be computed with \textit{at most} \( n \) \textit{unfoldings} of the \texttt{while}-loop (that is, \( n \) evaluations of the boolean condition). The general formula is

\[
(F^n \bot) \ s = \begin{cases} 
  \text{undef} & \text{if } s \ x < 1 \text{ or } s \ x > n \\
  s[y \mapsto (s \ y) \cdot j \ldots 2 \cdot 1][x \mapsto 1] & \text{if } s \ x = j \text{ and } 1 \leq j \text{ and } j \leq n 
\end{cases}
\]

We then have

\[
(\text{FIX} \ F) \ s = \begin{cases} 
  \text{undef} & \text{if } s \ x < 1 \\
  s[y \mapsto (s \ y) \cdot n \ldots 2 \cdot 1][x \mapsto 1] & \text{if } s \ x = n \text{ and } n \geq 1 
\end{cases}
\]

So in the state \( s_0 \) where \( x \) has the value 3, we get that the value computed by the factorial statement is

\[
(\text{FIX} \ F) \ (s_0[y \mapsto 1]) \ y = 1 \cdot 3 \cdot 2 \cdot 1 = 6
\]

as expected.

\[ \square \]

Exercise 5.49

Consider the statement

\[
z := 0; \textbf{while } y \leq x \textbf{ do } (z := z + 1; x := x - y)
\]

and perform a development analogous to that of Example 5.48.

\[ \square \]

Exercise 5.50

Show that \( S_{dc}[\texttt{while true do skip}] \) is the totally undefined function \( \bot \).

\[ \square \]
Exercise 5.51
Extend the language with the statement `repeat S until b` and give the new (compositional) clause for $S_{ds}$. Validate the well-definedness of the extended version of $S_{ds}$.

Exercise 5.52
Extend the language with the statement `for x := a_1 to a_2 do S` and give the new (compositional) clause for $S_{ds}$. Validate the well-definedness of the extended version of $S_{ds}$.

To summarize, the well-definedness of $S_{ds}$ relies on the following results established above:

<table>
<thead>
<tr>
<th>Proof Summary for While:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well-definedness of Denotational Semantics</td>
</tr>
<tr>
<td>1: The set $\text{State} \mapsto \text{State}$ equipped with an appropriate order $\sqsubseteq$ is a cpo (Lemmas 5.13 and 5.25).</td>
</tr>
<tr>
<td>2: Certain functions $\Psi: (\text{State} \mapsto \text{State}) \rightarrow (\text{State} \mapsto \text{State})$ are continuous (Lemmas 5.43 and 5.45).</td>
</tr>
<tr>
<td>3: In the definition of $S_{ds}$, we only apply the fixed point operation to continuous functions (Proposition 5.47).</td>
</tr>
</tbody>
</table>

Properties of the Semantics
In the operational semantics, we defined a notion of two statements being semantically equivalent. A similar notion can be defined based on the denotational semantics: $S_1$ and $S_2$ are `semantically equivalent` if and only if

$$S_{ds}[S_1] = S_{ds}[S_2]$$

Exercise 5.53
Show that the following statements of `While` are semantically equivalent in the sense above:

- $S;\text{skip}$ and $S$
5.4 An Equivalence Result

- $S_1; (S_2; S_3)$ and $(S_1; S_2); S_3$
- while $b$ do $S$ and if $b$ then $(S; \text{while } b \text{ do } S)$ else skip

Exercise 5.54 (*)
Prove that repeat $S$ until $b$ and S; while $\neg b$ do $S$ are semantically equivalent using the denotational approach. The semantics of the repeat-construct is given in Exercise 5.51.

5.4 An Equivalence Result

Having produced yet another semantics of the language While, we shall be interested in its relation to the operational semantics, and for this we shall focus on the structural operational semantics given in Section 2.2.

Theorem 5.55
For every statement $S$ of While, we have $S_{\text{sos}}[S] = S_{\text{ds}}[S]$.

Both $S_{\text{ds}}[S]$ and $S_{\text{sos}}[S]$ are functions in State $\mapsto$ State; that is, they are elements of a partially ordered set. To prove that two elements $d_1$ and $d_2$ of a partially ordered set are equal, it is sufficient to prove that $d_1 \sqsubseteq d_2$ and that $d_2 \sqsubseteq d_1$. Thus, to prove Theorem 5.55, we shall show that
- $S_{\text{sos}}[S] \sqsubseteq S_{\text{ds}}[S]$ and
- $S_{\text{ds}}[S] \sqsubseteq S_{\text{sos}}[S]$.

The first result is expressed by the following lemma.

Lemma 5.56
For every statement $S$ of While, we have $S_{\text{sos}}[S] \sqsubseteq S_{\text{ds}}[S]$.

Proof: It is sufficient to prove that for all states $s$ and $s'$
\[ \langle S, s \rangle \Rightarrow^* s' \text{ implies } S_{\text{ds}}[S]s = s' \] (*)

To do so, we shall need to establish the following property
\[ \langle S, s \rangle \Rightarrow s' \text{ implies } S_{\text{ds}}[S]s = s' \] (**)
\[ \langle S, s \rangle \Rightarrow \langle S', s' \rangle \text{ implies } S_{\text{ds}}[S]s = S_{\text{ds}}[S']s' \] (**)
Assuming that (***) holds, the proof of (*) is a straightforward induction on the length \(k\) of the derivation sequence \(\langle S, s \rangle \Rightarrow^k s'\) (see Section 2.2).

We now turn to the proof of (**), and for this we shall use induction on the shape of the derivation tree for \(\langle S, s \rangle \Rightarrow s'\) or \(\langle S, s \rangle \Rightarrow \langle S', s' \rangle\).

**The case** [assos]: We have

\[
\langle x := a, s \rangle \Rightarrow s[x\rightarrow A]s
\]

and since \(S_{ds}[x := a]s = s[x\rightarrow A]s\), the result follows.

**The case** [skipkos]: Analogous.

**The case** [comp₁kos]: Assume that

\[
\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle
\]

because \(\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle\). Then the induction hypothesis applied to the latter transition gives \(S_{ds}[S_1]s = S_{ds}[S'_1]s'\) and we get

\[
S_{ds}[S_1; S_2]s = S_{ds}[S_2](S_{ds}[S_1]s)
\]

\[
= S_{ds}[S_2](S_{ds}[S'_1]s')
\]

\[
= S_{ds}[S'_1; S_2]s'
\]

as required.

**The case** [comp₂kos]: Assume that

\[
\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle
\]

because \(\langle S_1, s \rangle \Rightarrow s'\). Then the induction hypothesis applied to that transition gives \(S_{ds}[S_1]s = s'\) and we get

\[
S_{ds}[S_1; S_2]s = S_{ds}[S_2](S_{ds}[S_1]s)
\]

\[
= S_{ds}[S_2]s'
\]

where the first equality comes from the definition of \(S_{ds}\) and we just argued for the second equality. This proves the result.

**The case** [if₁tos]: Assume that

\[
\text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \Rightarrow \langle S_1, s \rangle
\]

because \(B[b] s = \text{tt}\). Then

\[
S_{ds}[\text{if } b \text{ then } S_1 \text{ else } S_2]s = \text{cond}(B[b], S_{ds}[S_1], S_{ds}[S_2])s
\]

\[
= S_{ds}[S_1]s
\]

as required.

**The case** [if₂tos]: Analogous.

**The case** [whiletos]: Assume that
5.4 An Equivalence Result

\[(\text{while } b \text{ do } S, s) \Rightarrow (\text{if } b \text{ then } (S; \text{while } b \text{ do } S) \text{ else skip, } s)\]

From the definition of \(S_{ds}\), we have \(S_{ds}[\text{while } b \text{ do } S] = \text{FIX } F\), where \(F \ g = \text{cond}(B[b], \ g \circ S_{ds}[S], \text{id})\). We therefore get

\[
S_{ds}[\text{while } b \text{ do } S] = (\text{FIX } F)
= F (\text{FIX } F)
= \text{cond}(B[b], (\text{FIX } F) \circ S_{ds}[S], \text{id})
= \text{cond}(B[b], S_{ds}[\text{while } b \text{ do } S] \circ S_{ds}[S], \text{id})
= \text{cond}(B[b], S_{ds}[S; \text{while } b \text{ do } S], S_{ds}[\text{skip}])
= S_{ds}[\text{if } b \text{ then } (S; \text{while } b \text{ do } S) \text{ else skip}]
\]

as required. This completes the proof of (**). \(\square\)

Note that (*) does not imply that \(S_{sos}[S] = S_{ds}[S]\), as we have only proved that \(\text{if } S_{sos}[S] \neq \text{undef, then } S_{sos}[S] = S_{ds}[S]\). Still, there is the possibility that \(S_{ds}[S]\) may be defined for more arguments than \(S_{sos}[S]\). However, this is ruled out by the following lemma.

**Lemma 5.57**

For every statement \(S\) of While, we have \(S_{ds}[S] \subseteq S_{sos}[S]\).

**Proof:** We proceed by structural induction on the statement \(S\).

**The case** \(x := a\): Clearly \(S_{ds}[x := a] s = S_{sos}[x := a] s\). Note that this means that \(S_{sos}\) satisfies the clause defining \(S_{ds}\) in Table 5.1.

**The case** \(\text{skip}\): Clearly \(S_{ds}[\text{skip}] s = S_{sos}[\text{skip}] s\).

**The case** \(S_1 ; S_2\): Recall that \(\circ\) is monotone in both arguments (Lemma 5.45 and Exercise 5.46). We then have

\[
S_{ds}[S_1 ; S_2] = S_{ds}[S_2] \circ S_{ds}[S_1]
\subseteq S_{sos}[S_2] \circ S_{sos}[S_1]
\]

because the induction hypothesis applied to \(S_1\) and \(S_2\) gives \(S_{ds}[S_1] \subseteq S_{sos}[S_1]\) and \(S_{ds}[S_2] \subseteq S_{sos}[S_2]\). Furthermore, Exercise 2.21 gives that if \(\langle S_1, s \rangle \Rightarrow^* s'\) then \(\langle S_1 ; S_2, s \rangle \Rightarrow^* \langle S_2, s' \rangle\) and hence

\[
S_{sos}[S_2] \circ S_{sos}[S_1] \subseteq S_{sos}[S_1 ; S_2]
\]

and this proves the result. Note that in this case \(S_{sos}\) fulfils a weaker version of the clause defining \(S_{ds}\) in Table 5.1.
The case if \( b \) then \( S_1 \) else \( S_2 \): Recall that \( \text{cond} \) is monotone in its second and third arguments (Lemma 5.43 and Exercise 5.44). We then have

\[
\text{Sds}[\text{if } b \text{ then } S_1 \text{ else } S_2] = \text{cond}(B[b], \text{Sds}[S_1], \text{Sds}[S_2]) \\
\subseteq \text{cond}(B[b], \text{Ssos}[S_1], \text{Ssos}[S_2])
\]

because the induction hypothesis applied to \( S_1 \) and \( S_2 \) gives \( \text{Sds}[S_1] \subseteq \text{Ssos}[S_1] \) and \( \text{Sds}[S_2] \subseteq \text{Ssos}[S_2] \). Furthermore, it follows from \([\text{if } \text{tt} sos] \) and \([\text{if } \text{ff} sos] \) that

\[
\text{Ssos}[\text{if } b \text{ then } S_1 \text{ else } S_2] s = \text{Ssos}[S_1] s \quad \text{if } B[b] s = \text{tt} \\
\text{Ssos}[\text{if } b \text{ then } S_1 \text{ else } S_2] s = \text{Ssos}[S_2] s \quad \text{if } B[b] s = \text{ff}
\]

so that

\[
\text{cond}(B[b], \text{Ssos}[S_1], \text{Ssos}[S_2]) = \text{Ssos}[\text{if } b \text{ then } S_1 \text{ else } S_2]
\]

and this proves the result. Note that in this case \( \text{Ssos} \) fulfils the clause defining \( \text{Sds} \) in Table 5.1.

The case while \( b \) do \( S \): We have

\[
\text{Sds}[\text{while } b \text{ do } S] = \text{FIX } F
\]

where \( F g = \text{cond}(B[b], g \circ \text{Sds}[S], \text{id}) \), and we recall that \( F \) is continuous. It is sufficient to prove that

\[
\text{FIX } F \subseteq \text{Ssos}[\text{while } b \text{ do } S]
\]

because then Exercise 5.40 gives \( \text{FIX } F \subseteq \text{Ssos}[\text{while } b \text{ do } S] \) as required. From Exercise 2.21, we get

\[
\text{Ssos}[\text{while } b \text{ do } S] = \text{cond}(B[b], \text{Ssos}[S ; \text{while } b \text{ do } S], \text{id}) \\
\subseteq \text{cond}(B[b], \text{Ssos}[\text{while } b \text{ do } S] \circ \text{Ssos}[S], \text{id})
\]

The induction hypothesis applied to \( S \) gives \( \text{Sds}[S] \subseteq \text{Ssos}[S] \), so using the monotonicity of \( \circ \) and \( \text{cond} \), we get

\[
\text{Ssos}[\text{while } b \text{ do } S] \subseteq \text{cond}(B[b], \text{Ssos}[\text{while } b \text{ do } S] \circ \text{Ssos}[S], \text{id}) \\
\subseteq \text{cond}(B[b], \text{Ssos}[\text{while } b \text{ do } S] \circ \text{Sds}[S], \text{id}) \\
= \text{F}(\text{Ssos}[\text{while } b \text{ do } S])
\]

Note that in this case \( \text{Ssos} \) also fulfils a weaker version of the clause defining \( \text{Sds} \) in Table 5.1.

The key technique used in the proof can be summarized as follows:
### Proof Summary for While:

**Equivalence of Operational and Denotational Semantics**

1: Prove that $S_{\text{sos}}[S] \subseteq S_{\text{ds}}[S]$ by first using *induction on the shape of derivation trees* to show that

- if a statement is executed *one step* in the structural operational semantics and does not terminate, then this does not change the meaning in the denotational semantics, and
- if a statement is executed *one step* in the structural operational semantics and does terminate, then the same result is obtained in the denotational semantics

and secondly by using *induction on the length of derivation sequences*.

2: Prove that $S_{\text{ds}}[S] \subseteq S_{\text{sos}}[S]$ by showing that

- $S_{\text{sos}}$ fulfils slightly weaker versions of the clauses defining $S_{\text{ds}}$ in Table 5.1, that is, if
  
  $S_{\text{ds}}[S] = \Psi(\cdots S_{\text{ds}}[S'] \cdots)$

  then $S_{\text{sos}}[S] \sqsupseteq \Psi(\cdots S_{\text{sos}}[S'] \cdots)$

  A proof by *structural induction* then gives that $S_{\text{ds}}[S] \subseteq S_{\text{sos}}[S]$.

---

**Exercise 5.58**

Give a detailed argument showing that

$S_{\text{sos}}[\text{while } b \text{ do } S] \sqsupseteq \text{cond}(b[S], S_{\text{sos}}[\text{while } b \text{ do } S] \circ S_{\text{sos}}[S], \text{id})$.

**Exercise 5.59**

Extend the proof of Theorem 5.55 so that it applies to the language when augmented with *repeat S until b*.

**Exercise 5.60**

Extend the proof of Theorem 5.55 so that it applies to the language when augmented with *for x:=a_1 to a_2 do S*.
Exercise 5.61

Combining the results of Theorem 2.26 and Theorem 5.55, we get that $S_{ns}[S] = S_{ds}[S]$ holds for every statement $S$ of While. Give a direct proof of this (that is, without using the two theorems).