- It will develop each of the approaches for a simple language **While** of `while`-programs.
- It will illustrate the power and weakness of each of the approaches by extending **While** with other programming constructs.
- It will prove the relationship between the approaches for **While**.
- It will give examples of applications of the semantic descriptions in order to illustrate their merits.

## 1.2 The Example Language While

This book illustrates the various forms of semantics on a very simple imperative programming language called **While**. As a first step, we must specify its syntax.

The syntactic notation we use is based on BNF. First we list the various *syntactic categories* and give a meta-variable that will be used to range over *constructs* of each category. For our language, the meta-variables and categories are as follows:

$n$ will range over numerals, **Num**,

$x$ will range over variables, **Var**,

$a$ will range over arithmetic expressions, **Aexp**,

$b$ will range over boolean expressions, **Bexp**, and

$S$ will range over statements, **Stm**.

The meta-variables can be primed or subscripted. So, for example, $n$, $n'$, $n_1$, and $n_2$ all stand for numerals.

We assume that the structure of numerals and variables is given elsewhere; for example, numerals might be strings of digits, and variables might be strings of letters and digits starting with a letter. The structure of the other constructs is:

$$
\begin{array}{lcl}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & ::= & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & x := a \mid \texttt{skip} \mid S_1 \texttt{ ; } S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \\
 & \mid & \texttt{while } b \texttt{ do } S
\end{array}
$$

Thus, a boolean expression $b$ can only have one of six forms. It is called a *basis element* if it is `true` or `false` or has the form $a_1 = a_2$ or $a_1 \leq a_2$, where $a_1$ and $a_2$ are arithmetic expressions. It is called a *composite element* if it has the

$$
\begin{array}{rcl}
\mathcal{A}[\![n]\!]s & = & \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s & = & s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s \cdot \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s & = & \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{array}
$$

**Table 1.1** The semantics of arithmetic expressions

in state $s$ is the value bound to $x$ in $s$; that is, $s\ x$. The value of the composite expression $a_1+a_2$ in $s$ is the sum of the values of $a_1$ and $a_2$ in $s$. Similarly, the value of $a_1 \star a_2$ in $s$ is the product of the values of $a_1$ and $a_2$ in $s$, and the value of $a_1 - a_2$ in $s$ is the difference between the values of $a_1$ and $a_2$ in $s$. Note that $+$ and $-$ occurring on the right of these equations are the usual arithmetic operations, while on the left they are just pieces of syntax; this is analogous to the distinction between numerals and numbers, but we shall not bother to use different symbols.

## Example 1.6

Suppose that $s\ \mathtt{x} = \mathbf{3}$. Then we may calculate:

$$
\begin{array}{rcl}
\mathcal{A}[\![\mathtt{x+1}]\!]s & = & \mathcal{A}[\![\mathtt{x}]\!]s + \mathcal{A}[\![\mathtt{1}]\!]s \\
& = & (s\ \mathtt{x}) + \mathcal{N}[\![\mathtt{1}]\!] \\
& = & \mathbf{3} + \mathbf{1} \\
& = & \mathbf{4}
\end{array}
$$

Note that here 1 is a numeral (enclosed in the brackets '⟦' and '⟧'), whereas **1** is a number. □

## Example 1.7

Suppose we add the arithmetic expression $-\ a$ to our language. An acceptable semantic clause for this construct would be

$$\mathcal{A}[\![-\ a]\!]s = \mathbf{0} - \mathcal{A}[\![a]\!]s$$

whereas the alternative clause $\mathcal{A}[\![-\ a]\!]s = \mathcal{A}[\![0 - a]\!]s$ would contradict the compositionality requirement. □

$$
\begin{aligned}
\mathcal{B}[\![\texttt{true}]\!]s &= \mathbf{tt} \\
\mathcal{B}[\![\texttt{false}]\!]s &= \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg\, b]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}
\end{aligned}
$$

**Table 1.2** The semantics of boolean expressions

## Exercise 1.8

Prove that the equations of Table 1.1 define a total function $\mathcal{A}$ in $\mathbf{Aexp} \to (\mathbf{State} \to \mathbf{Z})$: First argue that it is sufficient to prove that for each $a \in \mathbf{Aexp}$ and each $s \in \mathbf{State}$ there is exactly one value $\mathbf{v} \in \mathbf{Z}$ such that $\mathcal{A}[\![a]\!]s = \mathbf{v}$. Next use structural induction on the arithmetic expressions to prove that this is indeed the case. □

The values of boolean expressions are truth values, so in a similar way we shall define their meanings by a (total) function from **State** to **T**:

$$\mathcal{B}\text{: } \mathbf{Bexp} \to (\mathbf{State} \to \mathbf{T})$$

Here **T** consists of the truth values **tt** (for true) and **ff** (for false).

Using $\mathcal{A}$, we can define $\mathcal{B}$ by the semantic clauses of Table 1.2. Again we have the distinction between syntax (e.g., $\leq$ on the left-hand side) and semantics (e.g., $\leq$ on the right-hand side).

## Exercise 1.9

Assume that $s$ $\mathtt{x} = \mathbf{3}$, and determine $\mathcal{B}[\![\neg(\mathtt{x} = 1)]\!]s$. □

## Exercise 1.10

Prove that Table 1.2 defines a total function $\mathcal{B}$ in $\mathbf{Bexp} \to (\mathbf{State} \to \mathbf{T})$. □

# 2
# Operational Semantics

The role of a statement in **While** is to change the state. For example, if x is bound to **3** in $s$ and we execute the statement $x := x + 1$, then we get a new state where x is bound to **4**. So while the semantics of arithmetic and boolean expressions only *inspect* the state in order to determine the value of the expression, the semantics of statements will *modify* the state as well.

In an operational semantics, we are concerned with *how* to execute programs and not merely what the results of execution are. More precisely, we are interested in how the states are modified during the execution of the statement. We shall consider two different approaches to operational semantics:

- *Natural semantics*: Its purpose is to describe how the *overall* results of executions are obtained; sometimes it is called a *big-step* operational semantics.
- *Structural operational semantics*: Its purpose is to describe how the *individual steps* of the computations take place; sometimes it is called a *small-step* operational semantics.

We shall see that for the language **While** we can easily specify both kinds of semantics and that they will be "equivalent" in a sense to be made clear later. However, in the next chapter we shall also give examples of programming constructs where one of the approaches is superior to the other.

For both kinds of operational semantics, the meaning of statements will be specified by a *transition system*. It will have two types of configurations:

| | |
|---|---|
| $[\text{ass}_{\text{ns}}]$ | $\langle x := a, s\rangle \rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$ |
| $[\text{skip}_{\text{ns}}]$ | $\langle \texttt{skip}, s\rangle \rightarrow s$ |
| $[\text{comp}_{\text{ns}}]$ | $\dfrac{\langle S_1, s\rangle \rightarrow s', \langle S_2, s'\rangle \rightarrow s''}{\langle S_1;S_2, s\rangle \rightarrow s''}$ |
| $[\text{if}^{\text{tt}}_{\text{ns}}]$ | $\dfrac{\langle S_1, s\rangle \rightarrow s'}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \rightarrow s'}$ if $\mathcal{B}[\![b]\!]s = \mathbf{tt}$ |
| $[\text{if}^{\text{ff}}_{\text{ns}}]$ | $\dfrac{\langle S_2, s\rangle \rightarrow s'}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \rightarrow s'}$ if $\mathcal{B}[\![b]\!]s = \mathbf{ff}$ |
| $[\text{while}^{\text{tt}}_{\text{ns}}]$ | $\dfrac{\langle S, s\rangle \rightarrow s', \langle \texttt{while } b \texttt{ do } S, s'\rangle \rightarrow s''}{\langle \texttt{while } b \texttt{ do } S, s\rangle \rightarrow s''}$ if $\mathcal{B}[\![b]\!]s = \mathbf{tt}$ |
| $[\text{while}^{\text{ff}}_{\text{ns}}]$ | $\langle \texttt{while } b \texttt{ do } S, s\rangle \rightarrow s$ if $\mathcal{B}[\![b]\!]s = \mathbf{ff}$ |

**Table 2.1** Natural semantics for **While**

$\langle S, s\rangle$ representing that the statement $S$ is to be executed from the state $s$ and

$s$ representing a terminal (that is final) state.

The *terminal configurations* will be those of the latter form. The *transition relation* will then describe how the execution takes place. The difference between the two approaches to operational semantics amounts to different ways of specifying the transition relation.

## 2.1 Natural Semantics

In a natural semantics we are concerned with the relationship between the *initial* and the *final* state of an execution. Therefore the transition relation will specify the relationship between the initial state and the final state for each statement. We shall write a transition as

$$\langle S, s\rangle \rightarrow s'$$

Intuitively this means that the execution of $S$ from $s$ will terminate and the resulting state will be $s'$.

The definition of $\rightarrow$ is given by the rules of Table 2.1. A *rule* has the general

form

$$\frac{\langle S_1, s_1\rangle \to s_1', \cdots, \langle S_n, s_n\rangle \to s_n'}{\langle S, s\rangle \to s'} \text{ if } \cdots$$

where $S_1, \cdots, S_n$ are *immediate constituents* of $S$ or are statements *constructed from* the immediate constituents of $S$. A rule has a number of *premises* (written above the solid line) and one *conclusion* (written below the solid line). A rule may also have a number of *conditions* (written to the right of the solid line) that have to be fulfilled whenever the rule is applied. Rules with an empty set of premises are called *axioms* and the solid line is then omitted.

Intuitively, the axiom $[\text{ass}_{\text{ns}}]$ says that in a state $s$, $x := a$ is executed to yield a final state $s[x \mapsto \mathcal{A}[\![a]\!]s]$, which is like $s$ except that $x$ has the value $\mathcal{A}[\![a]\!]s$. This is really an *axiom schema* because $x$, $a$, and $s$ are meta-variables standing for arbitrary variables, arithmetic expressions, and states but we shall simply use the term *axiom* for this. We obtain an *instance* of the axiom by selecting particular variables, arithmetic expressions, and states. As an example, if $s_0$ is the state that assigns the value **0** to all variables, then

$$\langle \texttt{x := x+1}, s_0\rangle \to s_0[\texttt{x} \mapsto \mathbf{1}]$$

is an instance of $[\text{ass}_{\text{ns}}]$ because $x$ is instantiated to x, $a$ to x+1, and $s$ to $s_0$, and the value $\mathcal{A}[\![\texttt{x+1}]\!]s_0$ is determined to be **1**.

Similarly, $[\text{skip}_{\text{ns}}]$ is an axiom and, intuitively, it says that skip does not change the state. Letting $s_0$ be as above, we obtain

$$\langle \texttt{skip}, s_0\rangle \to s_0$$

as an instance of the axiom $[\text{skip}_{\text{ns}}]$.

Intuitively, the rule $[\text{comp}_{\text{ns}}]$ says that to execute $S_1;S_2$ from state $s$ we must first execute $S_1$ from $s$. Assuming that this yields a final state $s'$, we shall then execute $S_2$ from $s'$. The premises of the rule are concerned with the two statements $S_1$ and $S_2$, whereas the conclusion expresses a property of the composite statement itself. The following is an *instance* of the rule:

$$\frac{\langle \texttt{skip}, s_0\rangle \to s_0, \langle \texttt{x := x+1}, s_0\rangle \to s_0[\texttt{x} \mapsto \mathbf{1}]}{\langle \texttt{skip; x := x+1}, s_0\rangle \to s_0[\texttt{x} \mapsto \mathbf{1}]}$$

Here $S_1$ is instantiated to skip, $S_2$ to x := x + 1, $s$ and $s'$ are both instantiated to $s_0$, and $s''$ is instantiated to $s_0[\texttt{x} \mapsto \mathbf{1}]$. Similarly

$$\frac{\langle \texttt{skip}, s_0\rangle \to s_0[\texttt{x} \mapsto \mathbf{5}], \langle \texttt{x := x+1}, s_0[\texttt{x} \mapsto \mathbf{5}]\rangle \to s_0}{\langle \texttt{skip; x := x+1}, s_0\rangle \to s_0}$$

is an instance of $[\text{comp}_{\text{ns}}]$, although it is less interesting because its premises can never be derived from the axioms and rules of Table 2.1.

For the if-construct, we have two rules. The first one, $[\text{if}^{\text{tt}}_{\text{ns}}]$, says that to execute if $b$ then $S_1$ else $S_2$ we simply execute $S_1$ provided that $b$ evaluates

to tt in the state. The other rule, $[\text{if}^{\text{ff}}_{\text{ns}}]$, says that if $b$ evaluates to ff, then to execute if $b$ then $S_1$ else $S_2$ we just execute $S_2$. Taking $s_0$ x = 0,

$$\frac{\langle \texttt{skip}, s_0\rangle \rightarrow s_0}{\langle \texttt{if x = 0 then skip else x := x+1}, s_0\rangle \rightarrow s_0}$$

is an instance of the rule $[\text{if}^{\text{tt}}_{\text{ns}}]$ because $\mathcal{B}[\![\text{x} = 0]\!]s_0 = \text{tt}$. However, had it been the case that $s_0$ x $\neq 0$, then it would not be an instance of the rule $[\text{if}^{\text{tt}}_{\text{ns}}]$ because then $\mathcal{B}[\![\text{x} = 0]\!]s_0$ would amount to ff. Furthermore, it would not be an instance of the rule $[\text{if}^{\text{ff}}_{\text{ns}}]$ because the premise would contain the wrong statement.

Finally, we have one rule and one axiom expressing how to execute the while-construct. Intuitively, the meaning of the construct while $b$ do $S$ in the state $s$ can be explained as follows:

- If the test $b$ evaluates to true in the state $s$, then we first execute the body of the loop and then continue with the loop itself from the state so obtained.
- If the test $b$ evaluates to false in the state $s$, then the execution of the loop terminates.

The rule $[\text{while}^{\text{tt}}_{\text{ns}}]$ formalizes the first case where $b$ evaluates to tt and it says that then we have to execute $S$ followed by while $b$ do $S$ again. The axiom $[\text{while}^{\text{ff}}_{\text{ns}}]$ formalizes the second possibility and states that if $b$ evaluates to ff, then we terminate the execution of the while-construct, leaving the state unchanged. Note that the rule $[\text{while}^{\text{tt}}_{\text{ns}}]$ specifies the meaning of the while-construct in terms of the meaning of the very same construct, so we do *not* have a compositional definition of the semantics of statements.

When we use the axioms and rules to derive a transition $\langle S, s\rangle \rightarrow s'$, we obtain a *derivation tree*. The *root* of the derivation tree is $\langle S, s\rangle \rightarrow s'$ and the *leaves* are instances of axioms. The *internal nodes* are conclusions of instantiated rules, and they have the corresponding premises as their immediate sons. We request that all the instantiated conditions of axioms and rules be satisfied. When displaying a derivation tree, it is common to have the root at the bottom rather than at the top; hence the son is *above* its father. A derivation tree is called *simple* if it is an instance of an axiom; otherwise it is called *composite*.

## Example 2.1

Let us first consider the statement of Chapter 1:

$$(\text{z:=x; x:=y}); \text{y:=z}$$

Let $s_0$ be the state that maps all variables except x and y to 0 and has $s_0$ x = 5 and $s_0$ y = 7. Then an example of a derivation tree is

$$\frac{\dfrac{\langle z{:=}x,\ s_0\rangle \to s_1 \qquad \langle x{:=}y,\ s_1\rangle \to s_2}{\langle z{:=}x;\ x{:=}y,\ s_0\rangle \to s_2} \qquad \langle y{:=}z,\ s_2\rangle \to s_3}{\langle (z{:=}x;\ x{:=}y);\ y{:=}z,\ s_0\rangle \to s_3}$$

where we have used the abbreviations:

$$\begin{aligned} s_1 &= s_0[z \mapsto 5] \\ s_2 &= s_1[x \mapsto 7] \\ s_3 &= s_2[y \mapsto 5] \end{aligned}$$

The derivation tree has three leaves, denoted $\langle z{:=}x,\ s_0\rangle \to s_1$, $\langle x{:=}y,\ s_1\rangle \to s_2$, and $\langle y{:=}z,\ s_2\rangle \to s_3$, corresponding to three applications of the axiom $[\mathrm{ass}_{\mathrm{ns}}]$. The rule $[\mathrm{comp}_{\mathrm{ns}}]$ has been applied twice. One instance is

$$\frac{\langle z{:=}x,\ s_0\rangle \to s_1,\ \langle x{:=}y,\ s_1\rangle \to s_2}{\langle z{:=}x;\ x{:=}y,\ s_0\rangle \to s_2}$$

which has been used to combine the leaves $\langle z{:=}x,\ s_0\rangle \to s_1$ and $\langle x{:=}y,\ s_1\rangle \to s_2$ with the internal node labelled $\langle z{:=}x;\ x{:=}y,\ s_0\rangle \to s_2$. The other instance is

$$\frac{\langle z{:=}x;\ x{:=}y,\ s_0\rangle \to s_2,\ \langle y{:=}z,\ s_2\rangle \to s_3}{\langle (z{:=}x;\ x{:=}y);\ y{:=}z,\ s_0\rangle \to s_3}$$

which has been used to combine the internal node $\langle z{:=}x;\ x{:=}y,\ s_0\rangle \to s_2$ and the leaf $\langle y{:=}z,\ s_2\rangle \to s_3$ with the root $\langle (z{:=}x;\ x{:=}y);\ y{:=}z,\ s_0\rangle \to s_3$. □

Consider now the problem of constructing a derivation tree for a given statement $S$ and state $s$. The best way to approach this is to try to construct the tree from the root upwards. So we will start by finding an axiom or rule with a conclusion where the left-hand side matches the configuration $\langle S,\ s\rangle$. There are two cases:

- If it is an *axiom* and if the conditions of the axiom are satisfied, then we can determine the final state and the construction of the derivation tree is completed.
- If it is a *rule*, then the next step is to try to construct derivation trees for the premises of the rule. When this has been done, it must be checked that the conditions of the rule are fulfilled, and only then can we determine the final state corresponding to $\langle S,\ s\rangle$.

Often there will be more than one axiom or rule that matches a given configuration, and then the various possibilities have to be inspected in order to find a derivation tree. We shall see later that for **While** there will be at most one

derivation tree for each transition $\langle S, s\rangle \to s'$ but that this need not hold in extensions of **While**.

## Example 2.2

Consider the factorial statement

$$\texttt{y:=1; while } \neg\texttt{(x=1) do (y:=y} \star \texttt{x; x:=x}-1\texttt{)}$$

and let $s$ be a state with $s\ \texttt{x} = \mathbf{3}$. In this example, we shall show that

$$\langle \texttt{y:=1; while } \neg\texttt{(x=1) do (y:=y} \star \texttt{x; x:=x}-1\texttt{)}, s\rangle \to s[\texttt{y}\mapsto\mathbf{6}][\texttt{x}\mapsto\mathbf{1}] \quad (*)$$

To do so, we shall show that (*) can be obtained from the transition system of Table 2.1. This is done by constructing a derivation tree with the transition (*) as its root.

Rather than presenting the complete derivation tree $T$ in one go, we shall build it in an upwards manner. Initially, we only know that the root of $T$ is of the form (where we use an auxiliary state $s_{61}$ to be defined later)

$$\langle \texttt{y:=1; while } \neg\texttt{(x=1) do (y:=y} \star \texttt{x; x:=x}-1\texttt{)}, s\rangle \to s_{61}$$

However, the statement

$$\texttt{y:=1; while } \neg\texttt{(x=1) do (y:=y} \star \texttt{x; x:=x}-1\texttt{)}$$

is of the form $S_1; S_2$, so the only rule that could have been used to produce the root of $T$ is $[\text{comp}_{\text{ns}}]$. Therefore $T$ must have the form

$$\frac{\langle \texttt{y:=1}, s\rangle \to s_{13} \qquad T_1}{\langle \texttt{y:=1; while } \neg\texttt{(x=1) do (y:=y}\star\texttt{x; x:=x}-1\texttt{)}, s\rangle \to s_{61}}$$

for some state $s_{13}$ and some derivation tree $T_1$ that has root

$$\langle \texttt{while } \neg\texttt{(x=1) do (y:=y}\star\texttt{x; x:=x}-1\texttt{)}, s_{13}\rangle \to s_{61} \quad (**)$$

Since $\langle \texttt{y:=1}, s\rangle \to s_{13}$ has to be an instance of the axiom $[\text{ass}_{\text{ns}}]$, we get that $s_{13} = s[\texttt{y}\mapsto\mathbf{1}]$.

The missing part $T_1$ of $T$ is a derivation tree with root (**). Since the statement of (**) has the form $\texttt{while } b \texttt{ do } S$, the derivation tree $T_1$ must have been constructed by applying either the rule $[\text{while}^{\text{tt}}_{\text{ns}}]$ or the axiom $[\text{while}^{\text{ff}}_{\text{ns}}]$. Since $\mathcal{B}[\![\neg\texttt{(x=1)}]\!]s_{13} = \mathbf{tt}$, we see that only the rule $[\text{while}^{\text{tt}}_{\text{ns}}]$ could have been applied so $T_1$ will have the form

$$\frac{T_2 \qquad T_3}{\langle \texttt{while } \neg\texttt{(x=1) do (y:=y}\star\texttt{x; x:=x}-1\texttt{)}, s_{13}\rangle \to s_{61}}$$

where $T_2$ is a derivation tree with root

$$\langle \texttt{y:=y*x; x:=x-1}, s_{13}\rangle \rightarrow s_{32}$$

and $T_3$ is a derivation tree with root

$$\langle \texttt{while } \neg(\texttt{x=1}) \texttt{ do } (\texttt{y:=y*x; x:=x-1}), s_{32}\rangle \rightarrow s_{61} \qquad (***)$$

for some state $s_{32}$.

Using that the form of the statement y:=y*x; x:=x−1 is $S_1;S_2$, it is now easy to see that the derivation tree $T_2$ is

$$\frac{\langle \texttt{y:=y*x}, s_{13}\rangle \rightarrow s_{33} \qquad \langle \texttt{x:=x-1}, s_{33}\rangle \rightarrow s_{32}}{\langle \texttt{y:=y*x; x:=x-1}, s_{13}\rangle \rightarrow s_{32}}$$

where $s_{33} = s[\texttt{y}\mapsto\mathbf{3}]$ and $s_{32} = s[\texttt{y}\mapsto\mathbf{3}][\texttt{x}\mapsto\mathbf{2}]$. The leaves of $T_2$ are instances of $[\text{ass}_{\text{ns}}]$ and are combined using $[\text{comp}_{\text{ns}}]$. So now $T_2$ is fully constructed.

In a similar way, we can construct the derivation tree $T_3$ with root (***) and we get

$$\frac{\dfrac{\langle \texttt{y:=y*x}, s_{32}\rangle \rightarrow s_{62} \qquad \langle \texttt{x:=x-1}, s_{62}\rangle \rightarrow s_{61}}{\langle \texttt{y:=y*x; x:=x-1}, s_{32}\rangle \rightarrow s_{61}} \quad T_4}{\langle \texttt{while } \neg(\texttt{x=1}) \texttt{ do } (\texttt{y:=y*x; x:=x-1}), s_{32}\rangle \rightarrow s_{61}}$$

where $s_{62} = s[\texttt{y}\mapsto\mathbf{6}][\texttt{x}\mapsto\mathbf{2}]$, $s_{61} = s[\texttt{y}\mapsto\mathbf{6}][\texttt{x}\mapsto\mathbf{1}]$, and $T_4$ is a derivation tree with root

$$\langle \texttt{while } \neg(\texttt{x=1}) \texttt{ do } (\texttt{y:=y*x; x:=x-1}), s_{61}\rangle \rightarrow s_{61}$$

Finally, we see that the derivation tree $T_4$ is an instance of the axiom $[\text{while}^{\text{ff}}_{\text{ns}}]$ because $\mathcal{B}[\![\neg(\texttt{x=1})]\!]s_{61} = \mathbf{ff}$. This completes the construction of the derivation tree $T$ for (*). □

## Exercise 2.3

Consider the statement

$$\texttt{z:=0; while y}\leq\texttt{x do (z:=z+1; x:=x-y)}$$

Construct a derivation tree for this statement when executed in a state where x has the value **17** and y has the value **5**. □

We shall introduce the following terminology. The execution of a statement $S$ on a state $s$

- *terminates* if and only if there is a state $s'$ such that $\langle S, s\rangle \rightarrow s'$ and
- *loops* if and only if there is *no* state $s'$ such that $\langle S, s\rangle \rightarrow s'$.

(For the latter definition, note that no run-time errors are possible.) We shall say that a statement $S$ *always terminates* if its execution on a state $s$ terminates for all choices of $s$, and *always loops* if its execution on a state $s$ loops for all choices of $s$.

### Exercise 2.4

Consider the following statements

- `while` $\neg$`(x=1) do (y:=y*x; x:=x-1)`
- `while 1`$\leq$`x do (y:=y*x; x:=x-1)`
- `while true do skip`

For each statement determine whether or not it always terminates and whether or not it always loops. Try to argue for your answers using the axioms and rules of Table 2.1. □

## Properties of the Semantics

The transition system gives us a way of arguing about statements and their properties. As an example, we may be interested in whether two statements $S_1$ and $S_2$ are *semantically equivalent*; this means that for all states $s$ and $s'$

$$\langle S_1, s\rangle \to s' \quad \text{if and only if} \quad \langle S_2, s\rangle \to s'$$

### Lemma 2.5

The statement

$$\texttt{while } b \texttt{ do } S$$

is semantically equivalent to

$$\texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip}$$

Proof: The proof is in two parts. We shall first prove that if

$$\langle \texttt{while } b \texttt{ do } S, s\rangle \to s'' \qquad (*)$$

then

$$\langle \texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip}, s\rangle \to s'' \qquad (**)$$

Thus, if the execution of the loop terminates, then so does its one-level unfolding. Later we shall show that if the unfolded loop terminates, then so will the loop itself; the conjunction of these results then proves the lemma.

Because (*) holds, we know that we have a derivation tree $T$ for it. It can have one of two forms depending on whether it has been constructed using the rule $[\text{while}^{\text{tt}}_{\text{ns}}]$ or the axiom $[\text{while}^{\text{ff}}_{\text{ns}}]$. In the first case, the derivation tree $T$ has the form

$$\frac{T_1 \qquad T_2}{\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s''}$$

where $T_1$ is a derivation tree with root $\langle S,\ s\rangle \to s'$ and $T_2$ is a derivation tree with root $\langle \texttt{while}\ b\ \texttt{do}\ S,\ s'\rangle \to s''$. Furthermore, $\mathcal{B}[\![b]\!]s = \mathbf{tt}$. Using the derivation trees $T_1$ and $T_2$ as the premises for the rules $[\text{comp}_{\text{ns}}]$, we can construct the derivation tree

$$\frac{T_1 \qquad T_2}{\langle S;\ \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s''}$$

Using that $\mathcal{B}[\![b]\!]s = \mathbf{tt}$, we can use the rule $[\text{if}^{\text{tt}}_{\text{ns}}]$ to construct the derivation tree

$$\frac{\dfrac{T_1 \qquad\qquad T_2}{\langle S;\ \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s''}}{\langle \texttt{if}\ b\ \texttt{then}\ (S;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else skip},\ s\rangle \to s''}$$

thereby showing that (**) holds.

Alternatively, the derivation tree $T$ is an instance of $[\text{while}^{\text{ff}}_{\text{ns}}]$. Then $\mathcal{B}[\![b]\!]s = \mathbf{ff}$ and we must have that $s''{=}s$. So $T$ simply is

$$\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s$$

Using the axiom $[\text{skip}_{\text{ns}}]$, we get a derivation tree

$$\langle \texttt{skip},\ s\rangle \to s''$$

and we can now apply the rule $[\text{if}^{\text{ff}}_{\text{ns}}]$ to construct a derivation tree for (**):

$$\frac{\langle \texttt{skip},\ s\rangle \to s''}{\langle \texttt{if}\ b\ \texttt{then}\ (S;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else skip},\ s\rangle \to s''}$$

This completes the first part of the proof.

For the second part of the proof, we assume that (**) holds and shall prove that (*) holds. So we have a derivation tree $T$ for (**) and must construct one for (*). Only two rules could give rise to the derivation tree $T$ for (**), namely $[\text{if}^{\text{tt}}_{\text{ns}}]$ or $[\text{if}^{\text{ff}}_{\text{ns}}]$. In the first case, $\mathcal{B}[\![b]\!]s = \mathbf{tt}$ and we have a derivation tree $T_1$ with root

$$\langle S;\ \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \rightarrow s''$$

The statement has the general form $S_1;\ S_2$, and the only rule that could give this is $[\text{comp}_{\text{ns}}]$. Therefore there are derivation trees $T_2$ and $T_3$ for

$$\langle S,\ s\rangle \rightarrow s'$$

and

$$\langle \texttt{while}\ b\ \texttt{do}\ S,\ s'\rangle \rightarrow s''$$

for some state $s'$. It is now straightforward to use the rule $[\text{while}^{\text{tt}}_{\text{ns}}]$ to combine $T_2$ and $T_3$ into a derivation tree for (*).

In the second case, $\mathcal{B}[\![b]\!]s = \text{ff}$ and $T$ is constructed using the rule $[\text{if}^{\text{ff}}_{\text{ns}}]$. This means that we have a derivation tree for

$$\langle \texttt{skip},\ s\rangle \rightarrow s''$$

and according to axiom $[\text{skip}_{\text{ns}}]$ it must be the case that $s{=}s''$. But then we can use the axiom $[\text{while}^{\text{ff}}_{\text{ns}}]$ to construct a derivation tree for (*). This completes the proof. □

## Exercise 2.6

Prove that the two statements $S_1;(S_2;S_3)$ and $(S_1;S_2);S_3$ are semantically equivalent. Construct a statement showing that $S_1;S_2$ is not, in general, semantically equivalent to $S_2;S_1$. □

## Exercise 2.7

Extend the language **While** with the statement

$$\texttt{repeat}\ S\ \texttt{until}\ b$$

and define the relation $\rightarrow$ for it. (The semantics of the `repeat`-construct is not allowed to rely on the existence of a `while`-construct in the language.) Prove that `repeat` $S$ `until` $b$ and $S$; `if` $b$ `then skip else` (`repeat` $S$ `until` $b$) are semantically equivalent. □

## Exercise 2.8

Another iterative construct is

$$\texttt{for}\ x := a_1\ \texttt{to}\ a_2\ \texttt{do}\ S$$

Extend the language **While** with this statement and define the relation $\rightarrow$ for it. Evaluate the statement

$$\texttt{y:=1; for z:=1 to x do (y:=y} \star \texttt{x; x:=x-1)}$$

from a state where x has the value 5. Hint: You may need to assume that you have an "inverse" to $\mathcal{N}$, so that there is a numeral for each number that may arise during the computation. (The semantics of the `for`-construct is not allowed to rely on the existence of a `while`-construct in the language.) □

In the proof above Table 2.1 was used to inspect the structure of the derivation tree for a certain transition known to hold. In the proof of the next result, we shall combine this with an *induction on the shape of the derivation tree*. The idea can be summarized as follows:

**Induction on the Shape of Derivation Trees**

1: Prove that the property holds for all the simple derivation trees by showing that it holds for the *axioms* of the transition system.

2: Prove that the property holds for all composite derivation trees: For each *rule* assume that the property holds for its premises (this is called the *induction hypothesis*) and prove that it also holds for the conclusion of the rule provided that the conditions of the rule are satisfied.

We shall say that the semantics of Table 2.1 is *deterministic* if for all choices of $S$, $s$, $s'$, and $s''$ we have that

$$\langle S, s\rangle \rightarrow s' \text{ and } \langle S, s\rangle \rightarrow s'' \quad \text{imply} \quad s' = s''$$

This means that for every statement $S$ and initial state $s$ we can uniquely determine a final state $s'$ if (and only if) the execution of $S$ terminates.

## Theorem 2.9

The natural semantics of Table 2.1 is deterministic.

Proof: We assume that $\langle S, s\rangle \rightarrow s'$ and shall prove that

$$\text{if } \langle S, s\rangle \rightarrow s'' \text{ then } s' = s''.$$

We shall proceed by induction on the shape of the derivation tree for $\langle S, s\rangle \rightarrow s'$.

**The case** $[\text{ass}_{\text{ns}}]$: Then $S$ is $x{:=}a$ and $s'$ is $s[x \mapsto \mathcal{A}[\![a]\!]s]$. The only axiom or rule that could be used to give $\langle x{:=}a, s\rangle \rightarrow s''$ is $[\text{ass}_{\text{ns}}]$, so it follows that $s''$ must be $s[x \mapsto \mathcal{A}[\![a]\!]s]$ and thereby $s' = s''$.

**The case** [$\text{skip}_{\text{ns}}$]: Analogous.

**The case** [$\text{comp}_{\text{ns}}$]: Assume that

$$\langle S_1;S_2,\ s\rangle \rightarrow s'$$

holds because

$$\langle S_1,\ s\rangle \rightarrow s_0 \text{ and } \langle S_2,\ s_0\rangle \rightarrow s'$$

for some $s_0$. The only rule that could be applied to give $\langle S_1;S_2,\ s\rangle \rightarrow s''$ is [$\text{comp}_{\text{ns}}$], so there is a state $s_1$ such that

$$\langle S_1,\ s\rangle \rightarrow s_1 \text{ and } \langle S_2,\ s_1\rangle \rightarrow s''$$

The induction hypothesis can be applied to the premise $\langle S_1,\ s\rangle \rightarrow s_0$ and from $\langle S_1,\ s\rangle \rightarrow s_1$ we get $s_0 = s_1$. Similarly, the induction hypothesis can be applied to the premise $\langle S_2,\ s_0\rangle \rightarrow s'$ and from $\langle S_2,\ s_0\rangle \rightarrow s''$ we get $s' = s''$ as required.

**The case** [$\text{if}^{\text{tt}}_{\text{ns}}$]: Assume that

$$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2,\ s\rangle \rightarrow s'$$

holds because

$$\mathcal{B}[\![b]\!]s = \mathbf{tt} \text{ and } \langle S_1,\ s\rangle \rightarrow s'$$

From $\mathcal{B}[\![b]\!]s = \mathbf{tt}$ we get that the only rule that could be applied to give the alternative $\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2,\ s\rangle \rightarrow s''$ is [$\text{if}^{\text{tt}}_{\text{ns}}$]. So it must be the case that

$$\langle S_1,\ s\rangle \rightarrow s''$$

But then the induction hypothesis can be applied to the premise $\langle S_1,\ s\rangle \rightarrow s'$ and from $\langle S_1,\ s\rangle \rightarrow s''$ we get $s' = s''$.

**The case** [$\text{if}^{\text{ff}}_{\text{ns}}$]: Analogous.

**The case** [$\text{while}^{\text{tt}}_{\text{ns}}$]: Assume that

$$\langle \texttt{while } b \texttt{ do } S,\ s\rangle \rightarrow s'$$

because

$$\mathcal{B}[\![b]\!]s = \mathbf{tt},\ \langle S,\ s\rangle \rightarrow s_0 \text{ and } \langle \texttt{while } b \texttt{ do } S,\ s_0\rangle \rightarrow s'$$

The only rule that could be applied to give $\langle \texttt{while } b \texttt{ do } S,\ s\rangle \rightarrow s''$ is [$\text{while}^{\text{tt}}_{\text{ns}}$] because $\mathcal{B}[\![b]\!]s = \mathbf{tt}$, and this means that

$$\langle S,\ s\rangle \rightarrow s_1 \text{ and } \langle \texttt{while } b \texttt{ do } S,\ s_1\rangle \rightarrow s''$$

must hold for some $s_1$. Again the induction hypothesis can be applied to the premise $\langle S,\ s\rangle \rightarrow s_0$, and from $\langle S,\ s\rangle \rightarrow s_1$ we get $s_0 = s_1$. Thus we have

$$\langle \texttt{while } b \texttt{ do } S,\ s_0\rangle \rightarrow s' \text{ and } \langle \texttt{while } b \texttt{ do } S,\ s_0\rangle \rightarrow s''$$

Since $\langle \texttt{while}\ b\ \texttt{do}\ S, s_0\rangle \rightarrow s'$ is a premise of (the instance of) $[\text{while}_{\text{ns}}^{\text{tt}}]$, we can apply the induction hypothesis to it. From $\langle \texttt{while}\ b\ \texttt{do}\ S, s_0\rangle \rightarrow s''$ we therefore get $s' = s''$ as required.

**The case** $[\text{while}_{\text{ns}}^{\text{ff}}]$: Straightforward. □

## Exercise 2.10 (*)

Prove that `repeat` $S$ `until` $b$ (as defined in Exercise 2.7) is semantically equivalent to $S$; `while` $\neg b$ `do` $S$. Argue that this means that the extended semantics is deterministic. □

It is worth observing that we could not prove Theorem 2.9 using structural induction on the statement $S$. The reason is that the rule $[\text{while}_{\text{ns}}^{\text{tt}}]$ defines the semantics of `while` $b$ `do` $S$ in terms of itself. Structural induction works fine when the semantics is defined *compositionally* (as, e.g., $\mathcal{A}$ and $\mathcal{B}$ in Chapter 1). But the natural semantics of Table 2.1 is *not* defined compositionally because of the rule $[\text{while}_{\text{ns}}^{\text{tt}}]$.

Basically, induction on the shape of derivation trees is a kind of structural induction on the derivation trees: In the *base case*, we show that the property holds for the simple derivation trees. In the *induction step*, we assume that the property holds for the immediate constituents of a derivation tree and show that it also holds for the composite derivation tree.

## The Semantic Function $\mathcal{S}_{\text{ns}}$

The *meaning* of statements can now be summarized as a (partial) function from **State** to **State**. We define

$$\mathcal{S}_{\text{ns}}\text{: } \mathbf{Stm} \rightarrow (\mathbf{State} \hookrightarrow \mathbf{State})$$

and this means that for every statement $S$ we have a partial function

$$\mathcal{S}_{\text{ns}}[\![S]\!] \in \mathbf{State} \hookrightarrow \mathbf{State}.$$

It is given by

$$\mathcal{S}_{\text{ns}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S, s\rangle \rightarrow s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

Note that $\mathcal{S}_{\text{ns}}$ is a well-defined partial function because of Theorem 2.9. The need for partiality is demonstrated by the statement `while true do skip` that always loops (see Exercise 2.4); we then have

$$\mathcal{S}_{\text{ns}}[\![\texttt{while true do skip}]\!]\ s = \underline{\text{undef}}$$

for all states $s$.

## Exercise 2.11

The semantics of arithmetic expressions is given by the function $\mathcal{A}$. We can also use an operational approach and define a natural semantics for the arithmetic expressions. It will have two kinds of configurations:

$\langle a, s\rangle$ denoting that $a$ has to be evaluated in state $s$, and

$z$ denoting the final value (an element of $\mathbf{Z}$).

The transition relation $\rightarrow_{\text{Aexp}}$ has the form

$$\langle a, s\rangle \rightarrow_{\text{Aexp}} z$$

where the idea is that $a$ evaluates to $z$ in state $s$. Some example axioms and rules are

$$\langle n, s\rangle \rightarrow_{\text{Aexp}} \mathcal{N}[\![n]\!]$$

$$\langle x, s\rangle \rightarrow_{\text{Aexp}} s\ x$$

$$\frac{\langle a_1, s\rangle \rightarrow_{\text{Aexp}} z_1,\ \langle a_2, s\rangle \rightarrow_{\text{Aexp}} z_2}{\langle a_1 + a_2, s\rangle \rightarrow_{\text{Aexp}} z} \quad \text{where } z = z_1 + z_2$$

Complete the specification of the transition system. Use structural induction on **Aexp** to prove that the meaning of $a$ defined by this relation is the same as that defined by $\mathcal{A}$. □

## Exercise 2.12

In a similar, way we can specify a natural semantics for the boolean expressions. The transitions will have the form

$$\langle b, s\rangle \rightarrow_{\text{Bexp}} t$$

where $t \in \mathbf{T}$. Specify the transition system and prove that the meaning of $b$ defined in this way is the same as that defined by $\mathcal{B}$. □

## Exercise 2.13

Determine whether or not semantic equivalence of $S_1$ and $S_2$ amounts to $\mathcal{S}_{\text{ns}}[\![S_1]\!] = \mathcal{S}_{\text{ns}}[\![S_2]\!]$. □

| | |
|---|---|
| $[\text{ass}_{\text{sos}}]$ | $\langle x := a,\ s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$ |
| $[\text{skip}_{\text{sos}}]$ | $\langle \texttt{skip},\ s\rangle \Rightarrow s$ |
| $[\text{comp}^1_{\text{sos}}]$ | $\dfrac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_1';S_2,\ s'\rangle}$ |
| $[\text{comp}^2_{\text{sos}}]$ | $\dfrac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}$ |
| $[\text{if}^{\text{tt}}_{\text{sos}}]$ | $\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle$ if $\mathcal{B}[\![b]\!]s = \text{tt}$ |
| $[\text{if}^{\text{ff}}_{\text{sos}}]$ | $\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle$ if $\mathcal{B}[\![b]\!]s = \text{ff}$ |
| $[\text{while}_{\text{sos}}]$ | $\langle \texttt{while } b \texttt{ do } S,\ s\rangle \Rightarrow$ $\langle \texttt{if } b \texttt{ then } (S;\ \texttt{while } b \texttt{ do } S) \texttt{ else skip},\ s\rangle$ |

**Table 2.2** Structural operational semantics for **While**

## 2.2 Structural Operational Semantics

In structural operational semantics, the emphasis is on the *individual steps* of the execution; that is, the execution of assignments and tests. The transition relation has the form

$$\langle S,\ s\rangle \Rightarrow \gamma$$

where $\gamma$ either is of the form $\langle S',\ s'\rangle$ or of the form $s'$. The transition expresses the *first* step of the execution of $S$ from state $s$. There are two possible outcomes:

- If $\gamma$ is of the form $\langle S',\ s'\rangle$, then the execution of $S$ from $s$ is *not* completed and the remaining computation is expressed by the intermediate configuration $\langle S',\ s'\rangle$.
- If $\gamma$ is of the form $s'$, then the execution of $S$ from $s$ *has* terminated and the final state is $s'$.

We shall say that $\langle S,\ s\rangle$ is *stuck* if there is no $\gamma$ such that $\langle S,\ s\rangle \Rightarrow \gamma$.

The definition of $\Rightarrow$ is given by the axioms and rules of Table 2.2, and the general form of these is as in the previous section. Axioms $[\text{ass}_{\text{sos}}]$ and $[\text{skip}_{\text{sos}}]$ have not changed at all because the assignment and skip statements are fully executed in one step.

The rules $[\text{comp}^1_{\text{sos}}]$ and $[\text{comp}^2_{\text{sos}}]$ express that to execute $S_1;S_2$ in state $s$ we first execute $S_1$ one step from $s$. Then there are two possible outcomes:

- If the execution of $S_1$ has not been completed, we have to complete it before embarking on the execution of $S_2$.
- If the execution of $S_1$ has been completed, we can start on the execution of $S_2$.

The first case is captured by the rule $[\text{comp}^1_{\text{sos}}]$: if the result of executing the first step of $\langle S,\ s\rangle$ is an intermediate configuration $\langle S_1',\ s'\rangle$, then the next configuration is $\langle S_1';S_2,\ s'\rangle$, showing that we have to complete the execution of $S_1$ before we can start on $S_2$. The second case above is captured by the rule $[\text{comp}^2_{\text{sos}}]$: if the result of executing $S_1$ from $s$ is a final state $s'$, then the next configuration is $\langle S_2,\ s'\rangle$, so that we can now start on $S_2$.

From the axioms $[\text{if}^{\text{tt}}_{\text{sos}}]$ and $[\text{if}^{\text{ff}}_{\text{sos}}]$ we see that the first step in executing a conditional is to perform the test and to select the appropriate branch. Finally, the axiom $[\text{while}_{\text{sos}}]$ shows that the first step in the execution of the `while`-construct is to unfold it one level; that is, to rewrite it as a conditional. The test will therefore be performed in the second step of the execution (where one of the axioms for the `if`-construct is applied). We shall see an example of this shortly.

A *derivation sequence* of a statement $S$ starting in state $s$ is either

1. a *finite* sequence

$$\gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_k$$

which is sometimes written

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow \gamma_k$$

consisting of configurations satisfying $\gamma_0 = \langle S,\ s\rangle$, $\gamma_i \Rightarrow \gamma_{i+1}$ for $0 \leq i < k$, where $k \geq 0$ and where $\gamma_k$ is either a terminal configuration or a stuck configuration, or it is

2. an *infinite* sequence

$$\gamma_0, \gamma_1, \gamma_2, \cdots$$

which is sometimes written

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots$$

consisting of configurations satisfying $\gamma_0 = \langle S,\ s\rangle$ and $\gamma_i \Rightarrow \gamma_{i+1}$ for $0 \leq i$.

We shall write $\gamma_0 \Rightarrow^i \gamma_i$ to indicate that there are i steps in the execution from $\gamma_0$ to $\gamma_i$, and we write $\gamma_0 \Rightarrow^* \gamma_i$ to indicate that there are a finite number

of steps. Note that $\gamma_0 \Rightarrow^i \gamma_i$ and $\gamma_0 \Rightarrow^* \gamma_i$ need *not* be derivation sequences: they will be so if and only if $\gamma_i$ is either a terminal configuration or a stuck configuration.

## Example 2.14

Consider the statement

$$(z := x;\ x := y);\ y := z$$

of Chapter 1, and let $s_0$ be the state that maps all variables except x and y to **0** and that has $s_0$ x = **5** and $s_0$ y = **7**. We then have the derivation sequence

$$\begin{aligned}
&\langle (z := x;\ x := y);\ y := z,\ s_0\rangle \\
&\Rightarrow \langle x := y;\ y := z,\ s_0[z\mapsto 5]\rangle \\
&\Rightarrow \langle y := z,\ (s_0[z\mapsto 5])[x\mapsto 7]\rangle \\
&\Rightarrow ((s_0[z\mapsto 5])[x\mapsto 7])[y\mapsto 5]
\end{aligned}$$

Corresponding to *each* of these steps, we have *derivation trees* explaining why they take place. For the first step

$$\langle (z := x;\ x := y);\ y := z,\ s_0\rangle \Rightarrow \langle x := y;\ y := z,\ s_0[z\mapsto 5]\rangle$$

the derivation tree is

$$\dfrac{\dfrac{\langle z := x,\ s_0\rangle \Rightarrow s_0[z\mapsto 5]}{\langle z := x;\ x := y,\ s_0\rangle \Rightarrow \langle x := y,\ s_0[z\mapsto 5]\rangle}}{\langle (z := x;\ x := y);\ y := z,\ s_0\rangle \Rightarrow \langle x := y;\ y := z,\ s_0[z\mapsto 5]\rangle}$$

and it has been constructed from the axiom $[ass_{sos}]$ and the rules $[comp^1_{sos}]$ and $[comp^2_{sos}]$. The derivation tree for the second step is constructed in a similar way using only $[ass_{sos}]$ and $[comp^2_{sos}]$, and for the third step it simply is an instance of $[ass_{sos}]$. □

## Example 2.15

Assume that $s$ x = **3**. The first step of execution from the configuration

$$\langle y{:=}1;\ \texttt{while}\ \neg(x{=}1)\ \texttt{do}\ (y{:=}y \star x;\ x{:=}x{-}1),\ s\rangle$$

will give the configuration

$$\langle \texttt{while}\ \neg(x{=}1)\ \texttt{do}\ (y{:=}y \star x;\ x{:=}x{-}1),\ s[y\mapsto 1]\rangle$$

This is achieved using the axiom $[\text{ass}_{\text{sos}}]$ and the rule $[\text{comp}^2_{\text{sos}}]$ as shown by the derivation tree

$$\frac{\langle \texttt{y:=1},\ s\rangle \Rightarrow s[\texttt{y}\mapsto\mathbf{1}]}{\begin{array}{c}\langle \texttt{y:=1; while } \neg\texttt{(x=1) do (y:=y}\star\texttt{x; x:=x}-\texttt{1)},\ s\rangle \Rightarrow \\ \langle \texttt{while } \neg\texttt{(x=1) do (y:=y}\star\texttt{x; x:=x}-\texttt{1)},\ s[\texttt{y}\mapsto\mathbf{1}]\rangle\end{array}}$$

The next step of the execution will rewrite the loop as a conditional using the axiom $[\text{while}_{\text{sos}}]$ so we get the configuration

$$\begin{array}{l}\langle \texttt{if } \neg\texttt{(x=1) then } \texttt{((y:=y}\star\texttt{x; x:=x}-\texttt{1);} \\ \qquad\qquad\qquad\quad \texttt{while } \neg\texttt{(x=1) do (y:=y}\star\texttt{x; x:=x}-\texttt{1))} \\ \qquad\quad \texttt{else skip},\ s[\texttt{y}\mapsto\mathbf{1}]\rangle\end{array}$$

The following step will perform the test and yields (according to $[\text{if}^{\text{tt}}_{\text{sos}}]$) the configuration

$$\langle \texttt{(y:=y}\star\texttt{x; x:=x}-\texttt{1); while } \neg\texttt{(x=1) do (y:=y} \star \texttt{x; x:=x}-\texttt{1)},\ s[\texttt{y}\mapsto\mathbf{1}]\rangle$$

We can then use $[\text{ass}_{\text{sos}}]$, $[\text{comp}^2_{\text{sos}}]$, and $[\text{comp}^1_{\text{sos}}]$ to obtain the configuration

$$\langle \texttt{x:=x}-\texttt{1; while } \neg\texttt{(x=1) do (y:=y} \star \texttt{x; x:=x}-\texttt{1)},\ s[\texttt{y}\mapsto\mathbf{3}]\rangle$$

as is verified by the derivation tree

$$\frac{\dfrac{\langle \texttt{y:=y}\star\texttt{x},\ s[\texttt{y}\mapsto\mathbf{1}]\rangle \Rightarrow s[\texttt{y}\mapsto\mathbf{3}]}{\langle \texttt{y:=y}\star\texttt{x; x:=x}-\texttt{1},\ s[\texttt{y}\mapsto\mathbf{1}]\rangle \Rightarrow \langle \texttt{x:=x}-\texttt{1},\ s[\texttt{y}\mapsto\mathbf{3}]\rangle}}{\begin{array}{c}\langle \texttt{(y:=y}\star\texttt{x; x:=x}-\texttt{1); while } \neg\texttt{(x=1) do (y:=y}\star\texttt{x; x:=x}-\texttt{1)},\ s[\texttt{y}\mapsto\mathbf{1}]\rangle \Rightarrow \\ \langle \texttt{x:=x}-\texttt{1; while } \neg\texttt{(x=1) do (y:=y} \star \texttt{x; x:=x}-\texttt{1)},\ s[\texttt{y}\mapsto\mathbf{3}]\rangle\end{array}}$$

Using $[\text{ass}_{\text{sos}}]$ and $[\text{comp}^2_{\text{sos}}]$, the next configuration will then be

$$\langle \texttt{while } \neg\texttt{(x=1) do (y:=y} \star \texttt{x; x:=x}-\texttt{1)},\ s[\texttt{y}\mapsto\mathbf{3}][\texttt{x}\mapsto\mathbf{2}]\rangle$$

Continuing in this way, we eventually reach the final state $s[\texttt{y}\mapsto\mathbf{6}][\texttt{x}\mapsto\mathbf{1}]$. □

## Exercise 2.16

Construct a derivation sequence for the statement

$$\texttt{z:=0; while y}\leq\texttt{x do (z:=z+1; x:=x}-\texttt{y)}$$

when executed in a state where x has the value **17** and y has the value **5**. Determine a state $s$ such that the derivation sequence obtained for the statement above and $s$ is infinite. □

Given a statement $S$ in the language **While** and a state $s$, it is always possible to find *at least one* derivation sequence that starts in the configuration $\langle S, s\rangle$: simply apply axioms and rules forever or until a terminal or stuck configuration is reached. Inspection of Table 2.2 shows that there are no stuck configurations in **While**, and Exercise 2.22 below will show that there is in fact only one derivation sequence that starts with $\langle S, s\rangle$. However, some of the constructs considered in Chapter 3 that extend **While** will have configurations that are stuck or more than one derivation sequence that starts in a given configuration.

In analogy with the terminology of the previous section, we shall say that the execution of a statement $S$ on a state $s$

- *terminates* if and only if there is a finite derivation sequence starting with $\langle S, s\rangle$ and
- *loops* if and only if there is an infinite derivation sequence starting with $\langle S, s\rangle$.

We shall say that the execution of $S$ on $s$ *terminates successfully* if $\langle S, s\rangle \Rightarrow^* s'$ for some state $s'$; in **While** an execution terminates successfully if and only if it terminates because there are no stuck configurations. Finally, we shall say that a statement $S$ *always terminates* if it terminates on all states, and *always loops* if it loops on all states.

### Exercise 2.17

Extend **While** with the construct `repeat` $S$ `until` $b$ and specify a structural operational semantics for it. (The semantics for the `repeat`-construct is not allowed to rely on the existence of a `while`-construct.) □

### Exercise 2.18

Extend **While** with the construct `for` $x := a_1$ `to` $a_2$ `do` $S$ and specify the structural operational semantics for it. Hint: You may need to assume that you have an "inverse" to $\mathcal{N}$ so that there is a numeral for each number that may arise during the computation. (The semantics for the `for`-construct is not allowed to rely on the existence of a `while`-construct.) □

## Properties of the Semantics

For structural operational semantics, it is often useful to conduct proofs by induction on the *lengths* of the finite derivation sequences considered. The

proof technique may be summarized as follows:

| **Induction on the Length of Derivation Sequences** |
|---|
| 1: Prove that the property holds for all derivation sequences of length 0. |
| 2: Prove that the property holds for all finite derivation sequences: Assume that the property holds for all derivation sequences of length at most k (this is called the *induction hypothesis*) and show that it holds for derivation sequences of length k+1. |

The induction step of a proof following this principle will often be done by inspecting either

– the structure of the syntactic element or

– the derivation tree validating the first transition of the derivation sequence.

Note that the proof technique is a simple application of mathematical induction.

To illustrate the use of the proof technique, we shall prove the following lemma (to be used in the next section). Intuitively, the lemma expresses that the execution of a composite construct $S_1;S_2$ can be split into two parts, one corresponding to $S_1$ and the other corresponding to $S_2$.

## Lemma 2.19

If $\langle S_1;S_2,\ s\rangle \Rightarrow^{k} s''$, then there exists a state $s'$ and natural numbers $k_1$ and $k_2$ such that $\langle S_1,\ s\rangle \Rightarrow^{k_1} s'$ and $\langle S_2,\ s'\rangle \Rightarrow^{k_2} s''$, where $k = k_1+k_2$.

Proof: The proof is by induction on the number k; that is, by induction on the length of the derivation sequence $\langle S_1;S_2,\ s\rangle \Rightarrow^{k} s''$.

If $k = 0$, then the result holds vacuously (because $\langle S_1;S_2,\ s\rangle$ and $s''$ are different).

For the induction step, we assume that the lemma holds for $k \leq k_0$, and we shall prove it for $k_0+1$. So assume that

$$\langle S_1;S_2,\ s\rangle \Rightarrow^{k_0+1} s''$$

This means that the derivation sequence can be written as

$$\langle S_1;S_2,\ s\rangle \Rightarrow \gamma \Rightarrow^{k_0} s''$$

for some configuration $\gamma$. Now one of two cases applies depending on which of the two rules $[\mathrm{comp}^1_{\mathrm{sos}}]$ and $[\mathrm{comp}^2_{\mathrm{sos}}]$ was used to obtain $\langle S_1;S_2,\ s\rangle \Rightarrow \gamma$.

In the first case, where $[\mathrm{comp}^1_{\mathrm{sos}}]$ is used, we have $\gamma = \langle S_1';\ S_2, s'\rangle$ and

$$\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_1';S_2,\ s'\rangle$$

because

$$\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle$$

We therefore have

$$\langle S_1';S_2,\ s'\rangle \Rightarrow^{k_0} s''$$

and the induction hypothesis can be applied to this derivation sequence because it is shorter than the one with which we started. This means that there is a state $s_0$ and natural numbers $k_1$ and $k_2$ such that

$$\langle S_1',\ s'\rangle \Rightarrow^{k_1} s_0 \text{ and } \langle S_2,\ s_0\rangle \Rightarrow^{k_2} s''$$

where $k_1{+}k_2{=}k_0$. Using that $\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle$ and $\langle S_1',\ s'\rangle \Rightarrow^{k_1} s_0$, we get

$$\langle S_1,\ s\rangle \Rightarrow^{k_1+1} s_0$$

We have already seen that $\langle S_2,\ s_0\rangle \Rightarrow^{k_2} s''$, and since $(k_1{+}1){+}k_2 = k_0{+}1$, we have proved the required result.

The second possibility is that $[\mathrm{comp}^2_{\mathrm{sos}}]$ has been used to obtain the derivation $\langle S_1;S_2,\ s\rangle \Rightarrow \gamma$. Then we have

$$\langle S_1,\ s\rangle \Rightarrow s'$$

and $\gamma$ is $\langle S_2,\ s'\rangle$ so that

$$\langle S_2,\ s'\rangle \Rightarrow^{k_0} s''$$

The result now follows by choosing $k_1{=}1$ and $k_2{=}k_0$. □

## Exercise 2.20

Suppose that $\langle S_1;S_2,\ s\rangle{\Rightarrow}^*\langle S_2,\ s'\rangle$. Show that it is *not* necessarily the case that $\langle S_1,\ s\rangle{\Rightarrow}^* s'$. □

## Exercise 2.21 (Essential)

Prove that

$$\text{if } \langle S_1,\ s\rangle \Rightarrow^k s' \text{ then } \langle S_1;S_2,\ s\rangle \Rightarrow^k \langle S_2,\ s'\rangle$$

I.e., the execution of $S_1$ is not influenced by the statement following it. □

In the previous section, we defined a notion of determinism based on the natural semantics. For the structural operational semantics, we define the similar notion as follows. The semantics of Table 2.2 is *deterministic* if for all choices of $S$, $s$, $\gamma$, and $\gamma'$ we have that

$$\langle S, s\rangle \Rightarrow \gamma \text{ and } \langle S, s\rangle \Rightarrow \gamma' \text{ imply } \gamma = \gamma'$$

## Exercise 2.22 (Essential)

Show that the structural operational semantics of Table 2.2 is deterministic. Deduce that there is exactly one derivation sequence starting in a configuration $\langle S, s\rangle$. Argue that a statement $S$ of **While** cannot both terminate and loop on a state $s$ and hence cannot both be always terminating and always looping. □

In the previous section, we defined a notion of two statements $S_1$ and $S_2$ being semantically equivalent. The similar notion can be defined based on the structural operational semantics: $S_1$ and $S_2$ are *semantically equivalent* if for all states $s$

- $\langle S_1, s\rangle \Rightarrow^* \gamma$ if and only if $\langle S_2, s\rangle \Rightarrow^* \gamma$, whenever $\gamma$ is a configuration that is either stuck or terminal, and
- there is an infinite derivation sequence starting in $\langle S_1, s\rangle$ if and only if there is one starting in $\langle S_2, s\rangle$.

Note that in the first case the lengths of the two derivation sequences may be different.

## Exercise 2.23

Show that the following statements of **While** are semantically equivalent in the sense above:

- $S$;`skip` and $S$
- `while` $b$ `do` $S$ and `if` $b$ `then` $(S$; `while` $b$ `do` $S)$ `else skip`
- $S_1;(S_2;S_3)$ and $(S_1;S_2);S_3$

You may use the result of Exercise 2.22. Discuss to what extent the notion of semantic equivalence introduced above is the same as that defined from the natural semantics. □

## Exercise 2.24

Prove that `repeat` $S$ `until` $b$ (as defined in Exercise 2.17) is semantically equivalent to $S$; `while` $\neg$ $b$ `do` $S$. □

### The Semantic Function $\mathcal{S}_{\text{sos}}$

As in the previous section, the *meaning* of statements can be summarized by a (partial) function from **State** to **State**:

$$\mathcal{S}_{\text{sos}}\colon \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

It is given by

$$\mathcal{S}_{\text{sos}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S, s\rangle \Rightarrow^* s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

The well-definedness of the definition follows from Exercise 2.22.

#### Exercise 2.25

Determine whether or not semantic equivalence of $S_1$ and $S_2$ amounts to $\mathcal{S}_{\text{sos}}[\![S_1]\!] = \mathcal{S}_{\text{sos}}[\![S_2]\!]$. □

## 2.3 An Equivalence Result

We have given two definitions of the semantics of **While** and we shall now address the question of their equivalence.

#### Theorem 2.26

For every statement $S$ of **While**, we have $\mathcal{S}_{\text{ns}}[\![S]\!] = \mathcal{S}_{\text{sos}}[\![S]\!]$.

This result expresses two properties:

- If the execution of $S$ from some state terminates in one of the semantics, then it also terminates in the other and the resulting states will be equal.
- If the execution of $S$ from some state loops in one of the semantics, then it will also loop in the other.

It should be fairly obvious that the first property follows from the theorem because there are no stuck configurations in the structural operational semantics of **While**. For the other property, suppose that the execution of $S$ on state $s$ loops in one of the semantics. If it terminates in the other semantics, we have a contradiction with the first property because both semantics are deterministic (Theorem 2.9 and Exercise 2.22). Hence $S$ will have to loop on state $s$ also in the other semantics.

The theorem is proved in two stages, as expressed by Lemma 2.27 and Lemma 2.28 below. We shall first prove Lemma 2.27.

## Lemma 2.27

For every statement $S$ of **While** and states $s$ and $s'$ we have

$$\langle S, s\rangle \to s' \text{ implies } \langle S, s\rangle \Rightarrow^* s'$$

So if the execution of $S$ from $s$ terminates in the natural semantics, then it will terminate in the same state in the structural operational semantics.

Proof: The proof proceeds by induction on the shape of the derivation tree for $\langle S, s\rangle \to s'$.

**The case** $[\text{ass}_{\text{ns}}]$: We assume that

$$\langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

From $[\text{ass}_{\text{sos}}]$, we get the required

$$\langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

**The case** $[\text{skip}_{\text{ns}}]$: Analogous.

**The case** $[\text{comp}_{\text{ns}}]$: Assume that

$$\langle S_1;S_2, s\rangle \to s''$$

because

$$\langle S_1, s\rangle \to s' \text{ and } \langle S_2, s'\rangle \to s''$$

The induction hypothesis can be applied to both of the premises $\langle S_1, s\rangle \to s'$ and $\langle S_2, s'\rangle \to s''$ and gives

$$\langle S_1, s\rangle \Rightarrow^* s' \text{ and } \langle S_2, s'\rangle \Rightarrow^* s''$$

From Exercise 2.21, we get

$$\langle S_1;S_2, s\rangle \Rightarrow^* \langle S_2, s'\rangle$$

and thereby $\langle S_1;S_2, s\rangle \Rightarrow^* s''$.

**The case** $[\text{if}^{\text{tt}}_{\text{ns}}]$: Assume that

$$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \to s'$$

because

$$\mathcal{B}[\![b]\!]s = \text{tt} \text{ and } \langle S_1, s\rangle \to s'$$

Since $\mathcal{B}[\![b]\!]s = \text{tt}$, we get

$$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle \Rightarrow^* s'$$

where the first relationship comes from $[\text{if}^{\text{tt}}_{\text{sos}}]$ and the second from the induction hypothesis applied to the premise $\langle S_1,\ s\rangle \rightarrow s'$.

**The case** $[\text{if}^{\text{ff}}_{\text{ns}}]$: Analogous.

**The case** $[\text{while}^{\text{tt}}_{\text{ns}}]$: Assume that

$$\langle \texttt{while } b \texttt{ do } S,\ s\rangle \rightarrow s''$$

because

$$\mathcal{B}[\![b]\!]s = \mathbf{tt},\ \langle S,\ s\rangle \rightarrow s' \text{ and } \langle \texttt{while } b \texttt{ do } S,\ s'\rangle \rightarrow s''$$

The induction hypothesis can be applied to both of the premises $\langle S,\ s\rangle \rightarrow s'$ and $\langle \texttt{while } b \texttt{ do } S,\ s'\rangle \rightarrow s''$ and gives

$$\langle S,\ s\rangle \Rightarrow^* s' \text{ and } \langle \texttt{while } b \texttt{ do } S,\ s'\rangle \Rightarrow^* s''$$

Using Exercise 2.21, we get

$$\langle S; \texttt{while } b \texttt{ do } S,\ s\rangle \Rightarrow^* s''$$

Using $[\text{while}_{\text{sos}}]$ and $[\text{if}^{\text{tt}}_{\text{sos}}]$ (with $\mathcal{B}[\![b]\!]s = \mathbf{tt}$), we get the first two steps of

$$\begin{aligned}
&\langle \texttt{while } b \texttt{ do } S,\ s\rangle \\
&\Rightarrow \langle \texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip},\ s\rangle \\
&\Rightarrow \langle S; \texttt{while } b \texttt{ do } S,\ s\rangle \\
&\Rightarrow^* s''
\end{aligned}$$

and we have already argued for the last part.

**The case** $[\text{while}^{\text{ff}}_{\text{ns}}]$: Straightforward. □

This completes the proof of Lemma 2.27. The second part of the theorem follows from Lemma 2.28.

## Lemma 2.28

For every statement $S$ of **While**, states $s$ and $s'$ and natural number k, we have that

$$\langle S,\ s\rangle \Rightarrow^{\text{k}} s' \text{ implies } \langle S,\ s\rangle \rightarrow s'.$$

So if the execution of $S$ from $s$ terminates in the structural operational semantics, then it will terminate in the same state in the natural semantics.

Proof: The proof proceeds by induction on the length of the derivation sequence $\langle S, s\rangle \Rightarrow^{k} s'$; that is, by induction on k.

If k=0, then the result holds vacuously.

To prove the induction step we assume that the lemma holds for $k \leq k_0$, and we shall then prove that it holds for $k_0$+1. We proceed by cases on how the first step of $\langle S, s\rangle \Rightarrow^{k_0+1} s'$ is obtained; that is, by inspecting the derivation tree for the first step of computation in the structural operational semantics.

**The case** $[ass_{sos}]$: Straightforward (and $k_0 = 0$).

**The case** $[skip_{sos}]$: Straightforward (and $k_0 = 0$).

**The cases** $[comp^1_{sos}]$ and $[comp^2_{sos}]$: In both cases, we assume that

$$\langle S_1;S_2, s\rangle \Rightarrow^{k_0+1} s''$$

We can now apply Lemma 2.19 and get that there exists a state $s'$ and natural numbers $k_1$ and $k_2$ such that

$$\langle S_1, s\rangle \Rightarrow^{k_1} s' \text{ and } \langle S_2, s'\rangle \Rightarrow^{k_2} s''$$

where $k_1$+$k_2$=$k_0$+1. The induction hypothesis can now be applied to each of these derivation sequences because $k_1 \leq k_0$ and $k_2 \leq k_0$. So we get

$$\langle S_1, s\rangle \rightarrow s' \text{ and } \langle S_2, s'\rangle \rightarrow s''$$

Using $[comp_{ns}]$, we now get the required $\langle S_1;S_2, s\rangle \rightarrow s''$.

**The case** $[if^{tt}_{sos}]$: Assume that $\mathcal{B}[\![b]\!]s = \mathbf{tt}$ and that

$$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle \Rightarrow^{k_0} s'$$

The induction hypothesis can be applied to the derivation $\langle S_1, s\rangle \Rightarrow^{k_0} s'$ and gives

$$\langle S_1, s\rangle \rightarrow s'$$

The result now follows using $[if^{tt}_{ns}]$.

**The case** $[if^{ff}_{sos}]$: Analogous.

**The case** $[while_{sos}]$: We have

$$\begin{aligned}
&\langle \texttt{while } b \texttt{ do } S, s\rangle \\
&\Rightarrow \langle \texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip}, s\rangle \\
&\Rightarrow^{k_0} s''
\end{aligned}$$

The induction hypothesis can be applied to the $k_0$ last steps of the derivation sequence and gives

$$\langle \texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip}, s\rangle \rightarrow s''$$

and from Lemma 2.5 we get the required

$$\langle \texttt{while } b \texttt{ do } S,\ s\rangle \rightarrow s''$$

Proof of Theorem 2.26: For an arbitrary statement $S$ and state $s$, it follows from Lemmas 2.27 and 2.28 that if $\mathcal{S}_{\mathrm{ns}}[\![S]\!]s = s'$ then $\mathcal{S}_{\mathrm{sos}}[\![S]\!]s = s'$ and vice versa. This suffices for showing that the functions $\mathcal{S}_{\mathrm{ns}}[\![S]\!]$ and $\mathcal{S}_{\mathrm{sos}}[\![S]\!]$ must be equal: if one is defined on a state $s$, then so is the other, and therefore if one is not defined on a state $s$, then neither is the other. □

## Exercise 2.29

Consider the extension of the language **While** with the statement `repeat` $S$ `until` $b$. The natural semantics of the construct was considered in Exercise 2.7 and the structural operational semantics in Exercise 2.17. Modify the proof of Theorem 2.26 so that the theorem applies to the extended language. □

## Exercise 2.30

Consider the extension of the language **While** with the statement `for` $x := a_1$ `to` $a_2$ `do` $S$. The natural semantics of the construct was considered in Exercise 2.8 and the structural operational semantics in Exercise 2.18. Modify the proof of Theorem 2.26 so that the theorem applies to the extended language. □

The proof technique employed in the proof of Theorem 2.26 may be summarized as follows:

| **Proof Summary for While:**<br>**Equivalence of two Operational Semantics** | |
|---|---|
| 1: | Prove by *induction on the shape of derivation trees* that for each derivation tree in the natural semantics there is a corresponding finite derivation sequence in the structural operational semantics. |
| 2: | Prove by *induction on the length of derivation sequences* that for each finite derivation sequence in the structural operational semantics there is a corresponding derivation tree in the natural semantics. |

When proving the equivalence of two operational semantics for a language with additional programming constructs, one may need to amend the above technique above. One reason is that the equivalence result may have to be expressed