# 3 Some principles of induction

Proofs of properties of programs often rely on the application of a proof method, or really a family of proof methods, called induction. The most commonly used forms of induction are mathematical induction and structural induction. These are both special cases of a powerful proof method called well-founded induction.

## 3.1 Mathematical induction

The natural numbers are built-up by starting from 0 and repeatedly adjoining successors. The natural numbers consist of no more than those elements which are obtained in this way. There is a corresponding proof principle called *mathematical induction.*

Let $P(n)$ be a a property of the natural numbers $n = 0, 1, \cdots$. The principle of mathematical induction says that in order to show $P(n)$ holds for all natural numbers $n$ it is sufficient to show

- $P(0)$ is true
- If $P(m)$ is true then so is $P(m+1)$ for any natural number $m$.

We can state it more succinctly, using some logical notation, as

$$(P(0)\ \&\ (\forall m \in \omega.\ P(m) \Rightarrow P(m+1)) \Rightarrow \forall n \in \omega.\ P(n).$$

The principle of mathematical induction is intuitively clear: If we know $P(0)$ and we have a method of showing $P(m+1)$ from the assumption $P(m)$ then from $P(0)$ we know $P(1)$, and applying the method again, $P(2)$, and then $P(3)$, and so on. The assertion $P(m)$ is called the *induction hypothesis*, $P(0)$ the *basis* of the induction and $(\forall m \in \omega.\ P(m) \Rightarrow P(m+1))$ the *induction step.*

Mathematical induction shares a feature with all other methods of proof by induction, that the first most obvious choice of induction hypothesis may not work in a proof. Imagine it is required to prove that a property $P$ holds of all the natural numbers. Certainly it is sensible to try to prove this with $P(m)$ as induction hypothesis. But quite often proving the induction step $\forall m \in \omega.\ (P(m) \Rightarrow P(m+1))$ is impossible. The rub can come in proving $P(m+1)$ from the assumption $P(m)$ because the assumption $P(m)$ is not strong enough. The way to tackle this is to strengthen the induction hypothesis to a property $P'(m)$ which implies $P(m)$. There is an art in finding $P'(m)$ however, because in proving the induction step, although we have a stronger assumption $P'(m)$, it is at the cost of having more to prove in $P'(m+1)$ which may be unnecessarily difficult, or impossible.

In showing a property $Q(m)$ holds inductively of all numbers $m$, it might be that the property's truth at $m+1$ depends not just on its truth at the predecessor $m$ but on

its truth at other numbers preceding $m$ as well. It is sensible to strengthen $Q(m)$ to an induction hypothesis $P(m)$ standing for $\forall k < m.\ Q(k)$. Taking $P(m)$ to be this property in the statement of ordinary mathematical induction we obtain

$$\forall k < 0.\ Q(k)$$

for the basis, and

$$\forall m \in \omega.((\forall k < m.\ Q(k)) \Rightarrow (\forall k < m+1.\ Q(k)))$$

for the induction step. However, the basis is vacuously true—there are no natural numbers strictly below 0, and the step is equivalent to

$$\forall m \in \omega.(\forall k < m.\ Q(k)) \Rightarrow Q(m).$$

We have obtained *course-of-values induction* as a special form of mathematical induction:

$$(\forall m \in \omega.(\forall k < m.\ Q(k)) \Rightarrow Q(m)) \Rightarrow \forall n \in \omega.\ Q(n).$$

**Exercise 3.1** Prove by mathematical induction that the following property $P$ holds for all natural numbers:

$$P(n) \iff_{def} \Sigma_{i=1}^{n}(2i-1) = n^2.$$

(The notation $\Sigma_{i=k}^{l} s_i$ abbreviates $s_k + s_{k+1} + \cdots + s_l$ when $k, l$ are integers with $k < l$.) □

**Exercise 3.2** A string is a sequence of symbols. A string $a_1 a_2 \cdots a_n$ with $n$ positions occupied by symbols is said to have *length* $n$. A string can be empty in which case it is said to have length 0. Two strings $s$ and $t$ can be concatenated to form the string $st$. Use mathematical induction to show there is no string $u$ which satisfies $au = ub$ for two distinct symbols $a$ and $b$. □

## 3.2 Structural induction

We would like a technique to prove "obvious" facts like

$$\langle a, \sigma \rangle \to m \ \&\ \langle a, \sigma \rangle \to m' \Rightarrow m = m'$$

for all arithmetic expressions $a$, states $\sigma$ and numbers $m, m'$. It says the evaluation of arithmetic expressions in **IMP** is *deterministic*. The standard tool is the principle of *structural induction*. We state it for arithmetic expressions but of course it applies more generally to all the syntactic sets of our language **IMP**.

Let $P(a)$ be a property of arithmetic expressions $a$. To show $P(a)$ holds for all arithmetic expressions $a$ it is sufficient to show:

- For all numerals $m$ it is the case that $P(m)$ holds.
- For all locations $X$ it is the case that $P(X)$ holds.
- For all arithmetic expressions $a_0$ and $a_1$, if $P(a_0)$ and $P(a_1)$ hold then so does $P(a_0 + a_1)$.
- For all arithmetic expressions $a_0$ and $a_1$, if $P(a_0)$ and $P(a_1)$ hold then so does $P(a_0 - a_1)$.
- For all arithmetic expressions $a_0$ and $a_1$, if $P(a_0)$ and $P(a_1)$ hold then so does $P(a_0 \times a_1)$.

The assertion $P(a)$ is called the *induction hypothesis*. The principle says that in order to show the induction hypothesis is true of all arithmetic expressions it suffices to show that it is true of atomic expressions and is preserved by all the methods of forming arithmetic expressions. Again this principle is intuitively obvious as arithmetic expressions are precisely those built-up according to the cases above. It can be stated more compactly using logical notation:

$$
\begin{array}{l}
(\forall m \in \mathbf{N}.\ P(m)) \ \&\ (\forall X \in \mathbf{Loc}.P(X)) \ \& \\
(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0) \ \&\ P(a_1) \Rightarrow P(a_0 + a_1)) \ \& \\
(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0) \ \&\ P(a_1) \Rightarrow P(a_0 - a_1)) \ \& \\
(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0) \ \&\ P(a_1) \Rightarrow P(a_0 \times a_1)) \\
\Rightarrow \\
\forall a \in \mathbf{Aexp}.\ P(a).
\end{array}
$$

In fact, as is clear, the conditions above not only imply $\forall a \in \mathbf{Aexp}.\ P(a)$ but also are equivalent to it.

Sometimes a degenerate form of structural induction is sufficient. An argument by cases on the structure of expressions will do when a property is true of all expressions simply by virtue of the different forms expressions can take, without having to use the fact that the property holds for subexpressions. An argument by cases on arithmetic expressions uses the fact that if

$$
\begin{array}{l}
(\forall m \in \mathbf{N}.\ P(m)) \& \\
(\forall X \in \mathbf{Loc}.P(X)) \ \& \\
(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0 + a_1)) \ \& \\
(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0 - a_1)) \ \& \\
(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0 \times a_1))
\end{array}
$$

then $\forall a \in \mathbf{Aexp}.\ P(a)$.

As an example of how to do proofs by structural induction we prove that the evaluation of arithmetic expression is deterministic.

**Proposition 3.3** *For all arithmetic expressions $a$, states $\sigma$ and numbers $m, m'$*

$$\langle a, \sigma\rangle \to m \ \&\ \langle a, \sigma\rangle \to m' \Rightarrow m = m'.$$

**Proof:** We proceed by structural induction on arithmetic expressions $a$ using the induction hypothesis $P(a)$ where

$$P(a) \text{ iff } \forall \sigma, m, m'.\ (\langle a, \sigma\rangle \to m \ \&\ \langle a, \sigma\rangle \to m' \Rightarrow m = m').$$

For brevity we shall write $\langle a, \sigma\rangle \to m, m'$ for $\langle a, \sigma\rangle \to m$ and $\langle a, \sigma\rangle \to m'$. Using structural induction the proof splits into cases according to the structure of $a$:
$a \equiv n$: If $\langle a, \sigma\rangle \to m, m'$ then there is only one rule for the evaluation of numbers so $m = m' = n$.
$a \equiv a_0 + a_1$: If $\langle a, \sigma\rangle \to m, m'$ then considering the form of the single rule for the evaluation of sums there must be $m_0, m_1$ so

$$\langle a_0, \sigma\rangle \to m_0 \text{ and } \langle a_1, \sigma\rangle \to m_1 \text{ with } m = m_0 + m_1$$

as well as $m_0', m_1'$ so

$$\langle a_0, \sigma\rangle \to m_0' \text{ and } \langle a_1, \sigma\rangle \to m_1' \text{ with } m' = m_0' + m_1'$$

By the induction hypothesis applied to $a_0$ and $a_1$ we obtain $m_0 = m_0'$ and $m_1 = m_1'$. Thus $m = m_0 + m_1 = m_0' + m_1' = m'$.

The remaining cases follow in a similar way. We can conclude, by the principle of structural induction, that $P(a)$ holds for all $a \in \mathbf{Aexp}$. □

One can prove the evaluation of expressions always terminates by structural induction, and corresponding facts about boolean expressions.

**Exercise 3.4** Prove by structural induction that the evaluation of arithmetic expressions always terminates, *i.e.*, for all arithmetic expression $a$ and states $\sigma$ there is some $m$ such that $\langle a, \sigma\rangle \to m$. □

**Exercise 3.5** Using these facts about arithmetic expressions, by structural induction, prove the evaluation of boolean expressions is firstly deterministic, and secondly total. □

**Exercise 3.6** What goes wrong when you try to prove the execution of commands is deterministic by using structural induction on commands? (Later, in Section 3.4, we shall give a proof using "structural induction" on derivations.) □

## 3.3 Well-founded induction

Mathematical and structural induction are special cases of a general and powerful proof principle called well-founded induction. In essence structural induction works because breaking down an expression into subexpressions can not go on forever, eventually it must lead to atomic expressions which can not be broken down any further. If a property fails to hold of any expression then it must fail on some minimal expression which when it is broken down yields subexpressions, all of which satisfy the property. This observation justifies the principle of structural induction: to show a property holds of all expressions it is sufficient to show that a property holds of an arbitrary expression if it holds of all its subexpressions. Similarly with the natural numbers, if a property fails to hold of all natural numbers then there has to be a smallest natural number at which it fails. The essential feature shared by both the subexpression relation and the predecessor relation on natural numbers is that do not give rise to infinite descending chains. This is the feature required of a relation if it is to support well-founded induction.

**Definition:** A *well-founded relation* is a binary relation $\prec$ on a set $A$ such that there are no infinite descending chains $\cdots \prec a_i \prec \cdots \prec a_1 \prec a_0$. When $a \prec b$ we say $a$ is a *predecessor* of $b$.

Note a well-founded relation is necessarily *irreflexive i.e.*, for no $a$ do we have $a \prec a$, as otherwise there would be the infinite decending chain $\cdots \prec a \prec \cdots \prec a \prec a$. We shall generally write $\preceq$ for the reflexive closure of the relation $\prec$, *i.e.*

$$a \preceq b \iff a = b \text{ or } a \prec b.$$

Sometimes one sees an alternative definition of well-founded relation, in terms of minimal elements.

**Proposition 3.7** *Let $\prec$ be a binary relation on a set $A$. The relation $\prec$ is well-founded iff any nonempty subset $Q$ of $A$ has a minimal element, i.e. an element $m$ such that*

$$m \in Q \ \& \ \forall b \prec m.\ b \notin Q.$$

**Proof:**
"if": Suppose every nonempty subset of $A$ has a minimal element. If $\cdots \prec a_i \prec \cdots \prec a_1 \prec a_0$ were an infinite descending chain then the set $Q = \{a_i \mid i \in \omega\}$ would be nonempty without a minimal element, a contradiction. Hence $\prec$ is well-founded.
"only if": To see this, suppose $Q$ is a nonempty subset of $A$. Construct a chain of elements as follows. Take $a_0$ to be any element of $Q$. Inductively, assume a chain of

elements $a_n \prec \cdots \prec a_0$ has been constructed inside $Q$. Either there is some $b \prec a_n$ such that $b \in Q$ or there is not. If not stop the construction. Otherwise take $a_{n+1} = b$. As $\prec$ is well-founded the chain $\cdots \prec a_i \prec \cdots \prec a_1 \prec a_0$ cannot be infinite. Hence it is finite, of the form $a_n \prec \cdots \prec a_0$ with $\forall b \prec a_n.\ b \notin Q$. Take the required minimal element $m$ to be $a_n$. □

**Exercise 3.8** Let $\prec$ be a well-founded relation on a set $B$. Prove

1. its transitive closure $\prec^+$ is also well-founded,
2. its reflexive, transitive closure $\prec^*$ is a partial order.

□

**The principle of well-founded induction.**

Let $\prec$ be a well founded relation on a set $A$. Let $P$ be a property. Then $\forall a \in A.\ P(a)$ iff

$$\forall a \in A.\ ([\forall b \prec a.\ P(b)] \Rightarrow P(a)).$$

The principle says that to prove a property holds of all elements of a well-founded set it suffices to show that if the property holds of all predecessors of an arbitrary element $a$ then the property holds of $a$.

We now prove the principle. The proof rests on the observation that any nonempty subset $Q$ of a set $A$ with a well-founded relation $\prec$ has a minimal element. Clearly if $P(a)$ holds for all elements of $A$ then $\forall a \in A.\ ([\forall b \prec a.\ P(b)] \Rightarrow P(a))$. To show the converse, we assume $\forall a \in A.\ ([\forall b \prec a.\ P(b)] \Rightarrow P(a))$ and produce a contradiction by supposing $\neg P(a)$ for some $a \in A$. Then, as we have observed, there must be a minimal element $m$ of the set $\{a \in A \mid \neg P(a)\}$. But then $\neg P(m)$ and yet $\forall b \prec m.\ P(b)$, which contradicts the assumption.

In mathematics this principle is sometimes called *Noetherian induction* after the algebraist Emmy Noether. Unfortunately, in some computer science texts (*e.g.* [59]) it is misleadingly called "structural induction".

**Example:** If we take the relation $\prec$ to be the successor relation

$$n \prec m \text{ iff } m = n + 1$$

on the non-negative integers the principle of well-founded induction specialises to mathematical induction. □

**Example:** If we take $\prec$ to be the "strictly less than" relation $<$ on the non-negative integers, the principle specialises to course-of-values induction. □

**Example:** If we take $\prec$ to be the relation between expressions such that $a \prec b$ holds iff $a$ is an immediate subexpression of $b$ we obtain the principle of structural induction as a special case of well-founded induction. □

Proposition 3.7 provides an alternative to proofs by well-founded induction. Suppose $A$ is a well-founded set. Instead of using well-founded induction to show every element of $A$ satisfies a property $P$, we can consider the subset of $A$ for which the property $P$ fails, *i.e.* the subset $F$ of counterexamples. By Proposition 3.7, to show $F$ is $\emptyset$ it is sufficient to show that $F$ cannot have a minimal element. This is done by obtaining a contradiction from the assumption that there is a minimal element in $F$. (See the proof of Proposition 3.12 for an example of this approach.) Whether to use this approach or the principle of well-founded induction is largely a matter of taste, though sometimes, depending on the problem, one approach can be more direct than the other.

**Exercise 3.9** For suitable well-founded relation on strings, use the "no counterexample" approach described above to show there is no string $u$ which satisfies $au = ub$ for two distinct symbols $a$ and $b$. Compare your proof with another by well-founded induction (and with the proof by mathematical induction asked for in Section 3.1). □

Proofs can often depend on a judicious choice of well-founded relation. In Chapter 10 we shall give some useful ways of constructing well-founded relations.

As an example of how the operational semantics supports proofs we show that Euclid's algorithm for the gcd (greatest common divisor) of two non-negative numbers terminates. Though such proofs are often less clumsy when based on a denotational semantics. (Later, Exercise 6.16 will show its correctness.) Euclid's algorithm for the greatest common divisor of two positive integers can be written in **IMP** as:

$$\begin{aligned} \text{Euclid} \equiv \textbf{while } \neg(M = N) \textbf{ do} \\ \textbf{if } M \leqslant N \\ \textbf{then } N := N - M \\ \textbf{else } M := M - N \end{aligned}$$

**Theorem 3.10** *For all states* $\sigma$

$$\sigma(M) \geq 1 \ \& \ \sigma(N) \geq 1 \Rightarrow \exists \sigma'.\ \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'.$$

**Proof:** We wish to show the property

$$P(\sigma) \iff \exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'.$$

holds for all states $\sigma$ in $S = \{\sigma \in \Sigma \mid \sigma(M) \geq 1 \ \& \ \sigma(N) \geq 1\}$.

We do this by well-founded induction on the relation $\prec$ on $S$ where

$$\sigma' \prec \sigma \text{ iff } (\sigma'(M) \leq \sigma(M) \ \& \ \sigma'(N) \leq \sigma(N)) \ \& \\ (\sigma'(M) \neq \sigma(M) \text{ or } \sigma'(N) \neq \sigma(N))$$

for states $\sigma', \sigma$ in $S$. Clearly $\prec$ is well-founded as the values in $M$ and $N$ cannot be decreased indefinitely and remain positive.

Let $\sigma \in S$. Suppose $\forall \sigma' \prec \sigma.\ P(\sigma')$. Abbreviate $\sigma(M) = m$ and $\sigma(N) = n$.

If $m = n$ then $\langle \neg(M = N), \sigma \rangle \rightarrow \textbf{false}$. Using its derivation we construct the derivation

$$\dfrac{\dfrac{\vdots}{\langle \neg(M = N), \sigma \rangle \rightarrow \textbf{false}}}{\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma}$$

using the rule for while-loops which applies when the boolean condition evaluates to false. In the case where $m = n$, $\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma$.

Otherwise $m \neq n$. In this case $\langle \neg(M = N), \sigma \rangle \rightarrow \textbf{true}$. From the rules for the execution of commands we derive

$$\langle \textbf{if } M \leqslant N \textbf{ then } N := N - M \textbf{ else } M := M - N,\ \sigma \rangle \rightarrow \sigma''$$

where

$$\sigma'' = \begin{cases} \sigma[n - m/N] & \text{if } m \leqslant n \\ \sigma[m - n/M] & \text{if } n < m. \end{cases}$$

In either case $\sigma'' \prec \sigma$. Hence $P(\sigma'')$ so $\langle \text{Euclid}, \sigma'' \rangle \rightarrow \sigma'$ for some $\sigma'$. Thus applying the other rule for while-loops we obtain

$$\dfrac{\dfrac{\vdots}{\langle \neg(M = N), \sigma \rangle \rightarrow \textbf{true}} \qquad \dfrac{\vdots}{\langle \textbf{if } M \leq N \textbf{ then } N := N - M \textbf{ else } M := M - N, \sigma \rangle \rightarrow \sigma''} \qquad \dfrac{\vdots}{\langle \text{Euclid}, \sigma'' \rangle \rightarrow \sigma'}}{\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'}$$

a derivation of $\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'$. Therefore $P(\sigma)$.

By well-founded induction we conclude $\forall \sigma \in S.\ P(\sigma)$, as required. □

Well-founded induction is the most important principle in proving the termination of programs. Uncertainties about termination arise because of loops or recursions in a program. If it can be shown that execution of a loop or recursion in a program decreases the value in a well-founded set then it must eventually terminate.

## 3.4 Induction on derivations

Structural induction alone is often inadequate to prove properties of operational semantics. Often it is useful to do induction on the structure of derivations. Putting this on a firm basis involves formalising some of the ideas met in the last chapter.

Possible derivations are determined by means of rules. Instances of rules have the form

$$\frac{}{x} \quad \text{or} \quad \frac{x_1, \ldots, x_n}{x},$$

where the former is an axiom with an empty set of premises and a conclusion $x$, while the latter has $\{x_1, \ldots, x_n\}$ as its set of premises and $x$ as its conclusion. The rules specify how to construct derivations, and through these define a set. The set defined by the rules consists precisely of those elements for which there is a derivation. A derivation of an element $x$ takes the form of a tree which is either an instance of an axiom

$$\frac{}{x}$$

or of the form

$$\frac{\frac{\vdots}{x_1}, \ldots, \frac{\vdots}{x_n}}{x}$$

which includes derivations of $x_1, \ldots, x_n$, the premises of a rule instance with conclusion $x$. In such a derivation we think of $\frac{\vdots}{x_1}, \cdots, \frac{\vdots}{x_n}$ as subderivations of the larger derivation of $x$.

Rule instances are got from rules by substituting actual terms or values for metavariables in them. All the rules we are interested in are *finitary* in that their premises are finite. Consequently, all rule instances have a finite, possibly empty set of premises and a conclusion. We start a formalisation of derivations from the idea of a set of rule instances.

A *set of rule instances* $R$ consists of elements which are pairs $(X/y)$ where $X$ is a finite set and $y$ is an element. Such a pair $(X/y)$ is called a *rule instance* with *premises* $X$ and conclusion $y$.

We are more used to seeing rule instances $(X/y)$ as

$$\frac{}{y} \text{ if } X = \emptyset, \text{ and as } \frac{x_1, \cdots, x_n}{y} \text{ if } X = \{x_1, \cdots, x_n\}.$$

Assume a set of rule instances $R$. An *$R$-derivation* of $y$ is either a rule instance $(\emptyset/y)$ or a pair $(\{d_1, \cdots, d_n\}/y)$ where $(\{x_1, \cdots, x_n\}/y)$ is a rule instance and $d_1$ is an $R$-derivation

of $x_1, \ldots, d_n$ is an $R$-derivation of $x_n$. We write $d \Vdash_R y$ to mean $d$ is an $R$-derivation of $y$. Thus

$$(\emptyset/y) \Vdash_R y \text{ if } (\emptyset/y) \in R, \text{ and}$$
$$(\{d_1, \cdots, d_n\}/y) \Vdash_R y \text{ if } (\{x_1, \cdots, x_n\}/y) \in R \;\&\; d_1 \Vdash_R x_1 \;\&\; \cdots \;\&\; d_n \Vdash_R x_n.$$

We say $y$ is derived from $R$ if there is an $R$-derivation of $y$, *i.e.* $d \Vdash_R y$ for some derivation $d$. We write $\Vdash_R y$ to mean $y$ is derived from $R$. When the rules are understood we shall write just $d \Vdash y$ and $\Vdash y$.

In operational semantics the premises and conclusions are tuples. There,

$$\Vdash \langle c, \sigma \rangle \to \sigma',$$

meaning $\langle c, \sigma \rangle \to \sigma'$ is derivable from the operational semantics of commands, is customarily written as just $\langle c, \sigma \rangle \to \sigma'$. It is understood that $\langle c, \sigma \rangle \to \sigma'$ includes, as part of its meaning, that it is derivable. We shall only write $\Vdash \langle c, \sigma \rangle \to \sigma'$ when we wish to emphasise that there is a derivation.

Let $d, d'$ be derivations. Say $d'$ is an *immediate subderivation* of $d$, written $d' \prec_1 d$, iff $d$ has the form $(D/y)$ with $d' \in D$. Write $\prec$ for the transitive closure of $\prec_1$, *i.e.* $\prec = \prec_1^+$. We say $d'$ is a proper subderivation of $d$ iff $d' \prec d$.

Because derivations are finite, both relations of being an immediate subderivation $\prec_1$ and that of being a proper subderivation are well-founded. This fact can be used to show the execution of commands is deterministic.

**Theorem 3.11** *Let $c$ be a command and $\sigma_0$ a state. If $\langle c, \sigma_0 \rangle \to \sigma_1$ and $\langle c, \sigma_0 \rangle \to \sigma$, then $\sigma = \sigma_1$, for all states $\sigma, \sigma_1$.*

**Proof:** The proof proceeds by well-founded induction on the proper subderivation relation $\prec$ between derivations for the execution of commands. The property we shall show holds of all such derivations $d$ is the following:

$$P(d) \iff \forall c \in \mathbf{Com}, \sigma_0, \sigma, \sigma_1, \in \Sigma.\; d \Vdash \langle c, \sigma_0 \rangle \to \sigma \;\&\; \langle c, \sigma_0 \rangle \to \sigma_1 \Rightarrow \sigma = \sigma_1.$$

By the principle of well-founded induction, it suffices to show $\forall d' \prec d.\ P(d')$ implies $P(d)$.

Let $d$ be a derivation from the operational semantics of commands. Assume $\forall d' \prec d.\ P(d')$. Suppose

$$d \Vdash \langle c, \sigma_0 \rangle \to \sigma \text{ and } \Vdash \langle c, \sigma_0 \rangle \to \sigma_1.$$

Then $d_1 \Vdash \langle c, \sigma_0 \rangle \to \sigma_1$ for some $d_1$.

Now we show by cases on the structure of $c$ that $\sigma = \sigma_1$.

$c \equiv$ **skip**: In this case

$$d = d_1 = \frac{}{\langle \mathbf{skip}, \sigma_0 \rangle \to \sigma_0}.$$

$c \equiv X := a$: Both derivations have a similar form:

$$d = \frac{\begin{array}{c}\vdots\\ \langle a, \sigma_0 \rangle \to m\end{array}}{\langle X := a, \sigma_0 \rangle \to \sigma_0[m/X]} \qquad d_1 = \frac{\begin{array}{c}\vdots\\ \langle a, \sigma_0 \rangle \to m_1\end{array}}{\langle X := a, \sigma_0 \rangle \to \sigma_0[m_1/X]}$$

where $\sigma = \sigma_0[m/X]$ and $\sigma_1 = \sigma_0[m_1/X]$. As the evaluation of arithmetic expressions is deterministic $m = m_1$, so $\sigma = \sigma_1$.

$c \equiv c_0; c_1$: In this case

$$d = \frac{\begin{array}{c}\vdots\\ \langle c_0, \sigma_0 \rangle \to \sigma'\end{array} \quad \begin{array}{c}\vdots\\ \langle c_1, \sigma' \rangle \to \sigma\end{array}}{\langle c_0; c_1, \sigma_0 \rangle \to \sigma} \qquad d_1 = \frac{\begin{array}{c}\vdots\\ \langle c_0, \sigma_0 \rangle \to \sigma_1'\end{array} \quad \begin{array}{c}\vdots\\ \langle c_1, \sigma_1' \rangle \to \sigma_1\end{array}}{\langle c_0; c_1, \sigma_0 \rangle \to \sigma_1}.$$

Let $d^0$ be the subderivation

$$\frac{\begin{array}{c}\vdots\\ \langle c_0, \sigma_0 \rangle \to \sigma'\end{array}}{}$$

and $d^1$ the subderivation

$$\frac{\begin{array}{c}\vdots\\ \langle c_1, \sigma' \rangle \to \sigma\end{array}}{}$$

in $d$. Then $d^0 \prec d$ and $d^1 \prec d$, so $P(d^0)$ and $P(d^1)$. It follows that $\sigma' = \sigma_1'$, and $\sigma = \sigma_1$ (why?).

$c \equiv$ **if** $b$ **then** $c_0$ **else** $c_1$: The rule for conditionals which applies in this case is determined by how the boolean $b$ evaluates. By the exercises of Section 3.2, its evaluation is deterministic so either $\langle b, \sigma_0 \rangle \to$ **true** or $\langle b, \sigma_0 \rangle \to$ **false**, but not both.

When $\langle b, \sigma_0 \rangle \to$ true we have:

$$d = \frac{\begin{array}{c}\vdots\\ \langle b, \sigma_0 \rangle \to \mathbf{true}\end{array} \quad \begin{array}{c}\vdots\\ \langle c_0, \sigma_0 \rangle \to \sigma\end{array}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma_0 \rangle \to \sigma} \qquad d_1 = \frac{\begin{array}{c}\vdots\\ \langle b, \sigma_0 \rangle \to \mathbf{true}\end{array} \quad \begin{array}{c}\vdots\\ \langle c_0, \sigma_0 \rangle \to \sigma_1\end{array}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma_0 \rangle \to \sigma_1}.$$

Let $d'$ be the subderivation of $\langle c_0, \sigma_0\rangle \to \sigma$ in $d$. Then $d' \prec d$. Hence $P(d')$. Thus $\sigma = \sigma_1$. When $\langle b, \sigma_0\rangle \to \mathbf{false}$ the argument is similar.

$c \equiv \mathbf{while}\ b\ \mathbf{do}\ c$: The rule for while-loops which applies is again determined by how $b$ evaluates. Either $\langle b, \sigma_0\rangle \to \mathbf{true}$ or $\langle b, \sigma\rangle \to \mathbf{false}$, but not both.

When $\langle b, \sigma_0\rangle \to \mathbf{false}$ we have :

$$d = \frac{\dfrac{\vdots}{\langle b, \sigma_0\rangle \to \mathbf{false}}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma_0\rangle \to \sigma_0} \qquad d_1 = \frac{\dfrac{\vdots}{\langle b, \sigma_0\rangle \to \mathbf{false}}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma_0\rangle \to \sigma_0}$$

so certainly $\sigma = \sigma_0 = \sigma_1$.

When $\langle b, \sigma_0\rangle \to \mathbf{true}$ we have:

$$d = \frac{\dfrac{\vdots}{\langle b, \sigma_0\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c, \sigma_0\rangle \to \sigma'} \quad \dfrac{\vdots}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'\rangle \to \sigma}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma_0\rangle \to \sigma}$$

$$d_1 = \frac{\dfrac{\vdots}{\langle b, \sigma_0\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c, \sigma_0\rangle \to \sigma_1'} \quad \dfrac{\vdots}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma_1'\rangle \to \sigma_1}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma_0\rangle \to \sigma_1}$$

Let $d'$ be the subderivation of $\langle c, \sigma_0\rangle \to \sigma'$ and $d''$ the subderivation of $\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'\rangle \to \sigma$ in $d$. Then $d' \prec d$ and $d'' \prec d$ so $P(d')$ and $P(d'')$. It follows that $\sigma' = \sigma_1'$, and subsequently that $\sigma = \sigma_1$.

In all cases of $c$ we have shown $d \Vdash \langle c, \sigma_0\rangle \to \sigma$ and $\langle c, \sigma_0\rangle \to \sigma_1$ implies $\sigma = \sigma_1$.

By the principle of well-founded induction we conclude that $P(d)$ holds for all derivations $d$ for the execution of commands. This is equivalent to

$$\forall c \in \mathbf{Com}, \sigma_0, \sigma, \sigma_1, \in \Sigma.\ \langle c, \sigma_0\rangle \to \sigma\ \&\ \langle c, \sigma_0\rangle \to \sigma_1 \Rightarrow \sigma = \sigma_1,$$

which proves the theorem. □

As was remarked, Proposition 3.7 provides an alternative to proofs by well-founded induction. Induction on derivations is a special kind of well-founded induction used to prove a property holds of all derivations. Instead, we can attempt to produce a contradiction from the assumption that there is a minimal derivation for which the property is false. The approach is illustrated below:

**Proposition 3.12** *For all states* $\sigma, \sigma'$,

$$\langle \textbf{while true do skip}, \sigma \rangle \not\to \sigma'.$$

**Proof:** Abbreviate $w \equiv \textbf{while true do skip}$. Suppose $\langle w, \sigma \rangle \to \sigma'$ for some states $\sigma, \sigma'$. Then there is a minimal derivation $d$ such that $\exists \sigma, \sigma' \in \Sigma.\ d \Vdash \langle w, \sigma \rangle \to \sigma'$. Only one rule can be the final rule of $d$, making $d$ of the form:

$$d = \frac{\begin{array}{ccc} \vdots & \vdots & \vdots \\ \langle \textbf{true}, \sigma \rangle \to \textbf{true} & \langle c, \sigma \rangle \to \sigma'' & \langle \textbf{while true do } c, \sigma'' \rangle \to \sigma' \end{array}}{\langle \textbf{while true do } c, \sigma \rangle \to \sigma'}$$

But this contains a proper subderivation $d' \Vdash \langle w, \sigma \rangle \to \sigma'$, contradicting the minimality of $d$. □

## 3.5 Definitions by induction

Techniques like structural induction are often used to define operations on the set defined. Integers and arithmetic expressions share a common property, that of being built-up in a unique way. An integer is either zero or the successor of a unique integer, while an arithmetic expression is either atomic or a sum, or product *etc.* of a unique pair of expressions. It is by virtue of their being built up in a unique way that we can can make definitions by induction on integers and expressions. For example to define the length of an expression it is natural to define it in terms of the lengths of its components. For arithmetic expressions we can define

$$\begin{aligned} &\text{length}(n) = \text{length}(X) = 1, \\ &\text{length}(a_0 + a_1) = 1 + \text{length}(a_0) + \text{length}(a_1), \\ &\ldots \end{aligned}$$

For future reference we define $\text{loc}_L(c)$, the set of those locations which appear on the left of an assignment in a command. For a command $c$, the function $\text{loc}_L(c)$ is defined by structural induction by taking

$$\begin{aligned} \text{loc}_L(\textbf{skip}) &= \emptyset, & \text{loc}_L(X := a) &= \{X\}, \\ \text{loc}_L(c_0; c_1) &= \text{loc}_L(c_0) \cup \text{loc}_L(c_1), & \text{loc}_L(\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1) &= \text{loc}_L(c_0) \cup \text{loc}_L(c_1), \\ \text{loc}_L(\textbf{while } b \textbf{ do } c) &= \text{loc}_L(c). \end{aligned}$$

In a similar way one defines operations on the natural numbers by mathematical induction and operations defined on sets given by rules. In fact the proof of Proposition 3.7,

that every nonempty subset of a well-founded set has a minimal element, contains an implicit use of definition by induction on the natural numbers to construct a chain with a minimal element in the nonempty set.

Both definition by structural induction and definition by mathematical induction are special cases of definition by well-founded induction, also called *well-founded recursion.* To understand this name, notice that both definition by induction and structural induction allow a form of recursive definition. For example, the length of an arithmetic expression could have been defined in this manner:

$$\text{length}(a) = \begin{cases} 1 & \text{if } a \equiv n, \text{ a number} \\ \text{length}(a_0) + \text{length}(a_1) & \text{if } a \equiv (a_0 + a_1), \\ \vdots & \end{cases}$$

How the length function acts on a particular argument, like $(a_0 + a_1)$ is specified in terms of how the length function acts on other arguments, like $a_0$ and $a_1$. In this sense the definition of the length function is defined recursively in terms of itself. However this recursion is done in such a way that the value on a particular argument is only specified in terms of strictly smaller arguments. In a similar way we are entitled to define functions on an arbitrary well-founded set. The general principle is more difficult to understand, resting as it does on some relatively sophisticated constructions on sets, and for this reason its full treatment is postponed to Section 10.4. (Although the material won't be needed until then, the curious or impatient reader might care to glance ahead. Despite its late appearance that section does not depend on any additional concepts.)

**Exercise 3.13** Give definitions by structural induction of $\text{loc}(a)$, $\text{loc}(b)$ and $\text{loc}_R(c)$, the sets of locations which appear in arithmetic expressions $a$, boolean expressions $b$ and the right-hand sides of assignments in commands $c$. □

## 3.6 Further reading

The techniques and ideas discussed in this chapter are well-known, basic techniques within mathematical logic. As operational semantics follows the lines of natural deduction, it is not surprising that it shares basic techniques with proof theory, as presented in [84] for example—derivations are really a simple kind of proof. For a fairly advanced, though accessible, account of proof theory with a computer science slant see [51, 40], which contains much more on notations for proofs (and so derivations). Further explanation and uses of well-founded induction can be found in [59] and [21], where it is called "structural induction", in [58] and [73]), and here, especially in Chapter 10.

# 4 Inductive definitions

This chapter is an introduction to the theory of inductively defined sets, of which presentations of syntax and operational semantics are examples. Sets inductively defined by rules are shown to be the least sets closed under the rules. As such, a principle of induction, called rule induction, accompanies the constructions. It specialises to proof rules for reasoning about the operational semantics of **IMP**.

## 4.1 Rule induction

We defined the syntactic set of arithmetic expressions **Aexp** as the set obtained from the formation rules for arithmetic expressions. We have seen there is a corresponding induction principle, that of structural induction on arithmetic expressions. We have defined the operational semantics of while-programs by defining evaluation and execution relations as relations given by rules which relate evaluation or execution of terms to the evaluation or execution of their components. For example, the evaluation relation on arithmetic expressions was defined by the rules of Section 2.2 as a ternary relation which is the set consisting of triples $(a, \sigma, n)$ of $\mathbf{Aexp} \times \Sigma \times \mathbf{N}$ such that $\langle a, \sigma \rangle \rightarrow n$. There is a corresponding induction principle which we can see as a special case of a principle we call rule induction.

We are interested in defining a set by rules. Viewed abstractly, instances of rules have the form $(\emptyset/x)$ or $(\{x_1, \ldots, x_n\}/x)$. Given a set of rule instances $R$, we write $I_R$ for the set defined by $R$ consisting of precisely of those elements $x$ for which there is a derivation. Put another way

$$I_R = \{x \mid \Vdash_R x\}.$$

The principle of rule induction is useful to show a property is true of all the elements in a set defined by some rules. It is based on the idea that if a property is preserved in moving from the premises to the conclusion of all rule instances in a derivation then the conclusion of the derivation has the property, so the property is true of all elements in the set defined by the rules.

**The general principle of rule induction**

Let $I_R$ be defined by rule instances $R$. Let $P$ be a property. Then $\forall x \in I_R.\ P(x)$ iff for all rule instances $(X/y)$ in $R$ for which $X \subseteq I_R$

$$(\forall x \in X.\ P(x)) \Rightarrow P(y).$$

Notice for rule instances of the form $(X/y)$, with $X = \emptyset$, the last condition is equivalent to $P(y)$. Certainly then $\forall x \in X.\ x \in I_R\ \&\ P(x)$ is vacuously true because any $x$ in $\emptyset$

satisfies $P$—there are none. The statement of rule induction amounts to the following. For rule instances $R$, we have $\forall y \in I_R.\ P(y)$ iff for all instances of axioms

$$\frac{}{x}$$

$P(x)$ is true, and for all rule instances

$$\frac{x_1, \ldots, x_n}{x}$$

if $x_k \in I_R$ & $P(x_k)$ is true for all the premises, when $k$ ranges from 1 to $n$, then $P(x)$ is true of the conclusion.

The principle of rule induction is fairly intuitive. It corresponds to a superficially different, but equivalent method more commonly employed in mathematics. (This observation will also lead to a proof of the validity of rule induction.) We say a set $Q$ is *closed* under rule instances $R$, or simply *$R$-closed*, iff for all rule instances $(X/y)$

$$X \subseteq Q \Rightarrow y \in Q.$$

In other words, a set is closed under the rule instances if whenever the premises of any rule instance lie in the set so does its conclusion. In particular, an $R$-closed set must contain all the instances of axioms. The set $I_R$ is the least set closed under $R$ in this sense:

**Proposition 4.1** *With respect to rule instances $R$*
(i) *$I_R$ is $R$-closed, and*
(ii) *if $Q$ is an $R$-closed set then $I_R \subseteq Q$.*

**Proof:**
(i) It is easy to see $I_R$ is closed under $R$. Suppose $(X/y)$ is an instance of a rule in $R$ and that $X \subseteq I_R$. Then from the definition of $I_R$ there are derivations of each element of $X$. If $X$ is nonempty these derivations can be combined with the rule instance $(X/y)$ to provide a derivation of $y$, and, otherwise, $(\emptyset/y)$ provides a derivation immediately. In either case we obtain a derivation of $y$ which must therefore be in $I_R$ too. Hence $I_R$ is closed under $R$.
(ii) Suppose that $Q$ is $R$-closed. We want to show $I_R \subseteq Q$. Any element of $I_R$ is the conclusion of some derivation. But any derivation is built out of rule instances $(X/y)$. If the premises $X$ are in $Q$ then so is the conclusion $y$ (in particular, the conclusion of any axiom will be in $Q$). Hence we can work our way down any derivation, starting at

axioms, to show its conclusion is in $Q$. More formally, we can do an induction on the proper subderivation relation $\prec$ to show

$$\forall y \in I_R.\ d \Vdash_R y \Rightarrow y \in Q$$

for all $R$-derivations $d$. Therefore $I_R \subseteq Q$. □

**Exercise 4.2** Do the induction on derivations mentioned in the proof above. □

Suppose we wish to show a property $P$ is true of all elements of $I_R$, the set defined by rules $R$. The conditions (i) and (ii) in the proposition above furnish a method. Defining the set

$$Q = \{x \in I_R \mid P(x)\},$$

the property $P$ is true of all elements of $I_R$ iff $I_R \subseteq Q$. By condition (ii), to show $I_R \subseteq Q$ it suffices to show that $Q$ is $R$-closed. This will follow if for all rule instances $(X/y)$

$$(\forall x \in X.\ x \in I_R\ \&\ P(x)) \Rightarrow P(y)$$

But this is precisely what is required by rule induction to prove the property $P$ holds for all elements of $I_R$. The truth of this statement is not just sufficient but also necessary to show the property $P$ of all elements of $I_R$. Suppose $P(x)$ for all $x \in I_R$. Let $(X/y)$ be a rule instance such that

$$\forall x \in X.\ x \in I_R\ \&\ P(x).$$

By (i), saying $I_R$ is $R$-closed, we get $y \in I_R$, and so that $P(y)$. And in this way we have derived the principle of rule induction from (i) and (ii), saying that $I_R$ is the least $R$-closed set.

**Exercise 4.3** For rule instances $R$, show

$$\bigcap \{Q \mid Q \text{ is } R\text{-closed}\}$$

is $R$-closed. What is this set? □

**Exercise 4.4** Let the rules consist of $(\emptyset/0)$ and $(\{n\}/(n+1))$ where $n$ is a natural number. What is the set defined by the rules and what is rule induction in this case? □

In presenting rules we have followed the same style as that used in giving operational semantics. When it comes to defining syntactic sets by rules, BNF is the traditional way though it can be done differently. For instance, what is traditionally written as

$$a ::= \cdots \mid a_0 + a_1 \mid \cdots,$$

saying that if $a_0$ and $a_1$ are well-formed expressions arithmetic expressions then so is $a_0 + a_1$, could instead be written as

$$\frac{a_0 : \mathbf{Aexp} \quad a_1 : \mathbf{Aexp}}{a_0 + a_1 : \mathbf{Aexp}}.$$

This way of presenting syntax is becoming more usual.

**Exercise 4.5** What is rule induction in the case where the rules are the formation rules for **Aexp**? What about when the rules are those for boolean expressions? (Careful! See the next section.) □

## 4.2 Special rule induction

Thinking of the syntactic sets of boolean expressions and commands it is clear that sometimes a syntactic set is given by rules which involve elements from another syntactic set. For example, the formation rules for commands say how commands can be formed from arithmetic and boolean expressions, as well as other commands. The formation rules

$$c ::= \cdots \mid X := a \mid \cdots \mid \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 \mid \cdots,$$

can, for the sake of uniformity, be written as

$$\frac{X : \mathbf{Loc} \quad a : \mathbf{Aexp}}{X := a : \mathbf{Com}} \quad \text{and} \quad \frac{b : \mathbf{Bexp} \quad c_0 : \mathbf{Com} \quad c_1 : \mathbf{Com}}{\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 : \mathbf{Com}}.$$

Rule induction works by showing properties are preserved by the rules. This means that if we are to use rule induction to prove a property of all commands we must make sure that the property covers all arithmetic and boolean expressions as well. As it stands, the principle of rule induction does not instantiate to structural induction on commands, but to a considerably more awkward proof principle, simultaneously combining structural induction on commands with that on arithmetic and boolean expressions. A modified principle of rule induction is required for establishing properties of *subsets* of the set defined by rules.

**The special principle of rule induction**

Let $I_R$ be defined by rule instances $R$. Let $A \subseteq I_R$. Let $Q$ be a property. Then $\forall a \in A.\ Q(a)$ iff for all rule instances $(X/y)$ in $R$, with $X \subseteq I_R$ and $y \in A$,

$$(\forall x \in X \cap A.\ Q(x)) \Rightarrow Q(y).$$

The special principle of rule induction actually follows from the general principle. Let $R$ be a set of rule instances. Let $A$ be a subset of $I_R$, the set defined by $R$. Suppose $Q(x)$ is a property we are interested in showing is true of all elements of $A$. Define a corresponding property $P(x)$ by

$$P(x) \iff (x \in A \Rightarrow Q(x)).$$

Showing $Q(a)$ for all $a \in A$ is equivalent to showing that $P(x)$ is true for all $x \in I_R$. By the general principle of rule induction the latter is equivalent to

$$\forall (X/y) \in R. \quad X \subseteq I_R \;\&\; (\forall x \in X.(x \in A \Rightarrow Q(x))) \Rightarrow (y \in A \Rightarrow Q(y)).$$

But this is logically equivalent to

$$\forall (X/y) \in R. \quad (X \subseteq I_R \;\&\; y \in A \;\&\; (\forall x \in X.(x \in A \Rightarrow Q(x)))) \Rightarrow Q(y).$$

This is equivalent to the condition required by the special principle of rule induction.

**Exercise 4.6** Explain how structural induction for commands and booleans follows from the special principle of rule induction. □

Because the special principle follows from the general, any proof using the special principle can be replaced by one using the principle of general rule induction. But in practice use of the special principle can drastically cut down the number of rules to consider, a welcome feature when it comes to considering rule induction for operational semantics.

## 4.3 Proof rules for operational semantics

Not surprisingly, rule induction can be a useful tool for proving properties of operational semantics presented by rules, though then it generally takes a superficially different form because the sets defined by the rules are sets of tuples. This section presents the special cases of rule induction which we will use later in reasoning about the operational behaviour of **IMP** programs.

### 4.3.1 Rule induction for arithmetic expressions

The principle of rule induction for the evaluation of arithmetic expressions is got from the rules for their operational semantics. It is an example of rule induction; a property $P(a, \sigma, n)$ is true of all evaluations $\langle a, \sigma \rangle \to n$ iff it is preserved by the rules for building

up the evaluation relation.

$$
\begin{array}{l}
\forall a \in \mathbf{Aexp}, \sigma \in \Sigma, n \in \mathbf{N}. \quad \langle a, \sigma \rangle \to n \Rightarrow P(a, \sigma, n) \\
\text{iff} \\
[\forall n \in \mathbf{N}, \sigma \in \Sigma.\ P(n, \sigma, n) \\
\& \\
\forall X \in \mathbf{Loc}, \sigma \in \Sigma.\ P(X, \sigma, \sigma(X)) \\
\& \\
\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, n_0, n_1 \in \mathbf{N}. \\
\langle a_0, \sigma \rangle \to n_0 \ \&\ P(a_0, \sigma, n_0) \ \&\ \langle a_1, \sigma \rangle \to n_1 \ \&\ P(a_1, \sigma, n_1) \\
\Rightarrow P(a_0 + a_1, \sigma, n_0 + n_1) \\
\& \\
\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, n_0, n_1 \in \mathbf{N}. \\
\langle a_0, \sigma \rangle \to n_0 \ \&\ P(a_0, \sigma, n_0) \ \&\ \langle a_1, \sigma \rangle \to n_1 \ \&\ P(a_1, \sigma, n_1) \\
\Rightarrow P(a_0 - a_1, \sigma, n_0 - n_1) \\
\& \\
\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, n_0, n_1 \in \mathbf{N}. \\
\langle a_0, \sigma \rangle \to n_0 \ \&\ P(a_0, \sigma, n_0) \ \&\ \langle a_1, \sigma \rangle \to n_1 \ \&\ P(a_1, \sigma, n_1) \\
\Rightarrow P(a_0 \times a_1, \sigma, n_0 \times n_1)].
\end{array}
$$

Compare this specific principle with that for general rule induction. Notice how all possible rule instances are covered by considering one evaluation rule at a time.

### 4.3.2 Rule induction for boolean expressions

The rules for the evaluation of boolean expressions involve those for the evaluation of arithmetic expressions. Together the rules define a subset of

$$(\mathbf{Aexp} \times \Sigma \times \mathbf{N}) \cup (\mathbf{Bexp} \times \Sigma \times \mathbf{T}).$$

A principle useful for reasoning about the operational semantics of boolean expressions is got from the special principle of rule induction for properties $P(b, \sigma, t)$ on the subset $\mathbf{Bexp} \times \Sigma \times \mathbf{T}$.

$$\forall b \in \mathbf{Bexp}, \sigma \in \Sigma, t \in \mathbf{T}. \quad \langle b, \sigma \rangle \to t \Rightarrow P(b, \sigma, t)$$

iff

$$[\forall \sigma \in \Sigma.\ P(\mathbf{false}, \sigma, \mathbf{false}) \ \&\ \forall \sigma \in \Sigma.\ P(\mathbf{true}, \sigma, \mathbf{true})$$

$$\&$$

$$\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, m, n \in \mathbf{N}.$$

$$\langle a_0, \sigma \rangle \to m \ \&\ \langle a_1, \sigma \rangle \to n \ \&\ m = n \Rightarrow P(a_0 = a_1, \sigma, \mathbf{true})$$

$$\&$$

$$\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, m, n \in \mathbf{N}.$$

$$\langle a_0, \sigma \rangle \to m \ \&\ \langle a_1, \sigma \rangle \to n \ \&\ m \neq n \Rightarrow P(a_0 = a_1, \sigma, \mathbf{false})$$

$$\&$$

$$\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, m, n \in \mathbf{N}.$$

$$\langle a_0, \sigma \rangle \to m \ \&\ \langle a_1, \sigma \rangle \to n \ \&\ m \leq n \Rightarrow P(a_0 \leq a_1, \sigma, \mathbf{true})$$

$$\&$$

$$\forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, m, n \in \mathbf{N}.$$

$$\langle a_0, \sigma \rangle \to m \ \&\ \langle a_1, \sigma \rangle \to n \ \&\ m \not\leq n \Rightarrow P(a_0 \leq a_1, \sigma, \mathbf{false})$$

$$\&$$

$$\forall b \in \mathbf{Bexp}, \sigma \in \Sigma, t \in \mathbf{T}.$$

$$\langle b, \sigma \rangle \to t \ \&\ P(b, \sigma, t) \Rightarrow P(\neg b, \sigma, \neg t)$$

$$\&$$

$$\forall b_0, b_1 \in \mathbf{Bexp}, \sigma \in \Sigma, t_0, t_1 \in \mathbf{T}.$$

$$\langle b_0, \sigma \rangle \to t_0 \ \&\ P(b_0, \sigma, t_0) \ \&\ \langle b_1, \sigma \rangle \to t_1 \ \&\ P(b_1, \sigma, t_1) \Rightarrow P(b_0 \wedge b_1, \sigma, t_0 \wedge t_1)$$

$$\&$$

$$\forall b_0, b_1 \in \mathbf{Bexp}, \sigma \in \Sigma, t_0, t_1 \in \mathbf{T}.$$

$$\langle b_0, \sigma \rangle \to t_0 \ \&\ P(b_0, \sigma, t_0) \ \&\ \langle b_1, \sigma \rangle \to t_1 \ \&\ P(b_1, \sigma, t_1) \Rightarrow P(b_0 \vee b_1, \sigma, t_0 \vee t_1)].$$

### 4.3.3 Rule induction for commands

The principle of rule induction we use for reasoning about the operational semantics of commands is an instance of the special principle of rule induction. The rules for the execution of commands involve the evaluation of arithmetic and boolean expressions. The rules for the operational semantics of the different syntactic sets taken together

define a subset of

$$(\mathbf{Aexp} \times \Sigma \times \mathbf{N}) \cup (\mathbf{Bexp} \times \Sigma \times \mathbf{T}) \cup (\mathbf{Com} \times \Sigma \times \Sigma).$$

We use the special principle for properties $P(c, \sigma, \sigma')$ on the subset $\mathbf{Com} \times \Sigma \times \Sigma$. (Try to write it down and compare your result with the following.)

$$\begin{array}{l}
\forall c \in \mathbf{Com}, \sigma, \sigma' \in \Sigma. \quad \langle c, \sigma \rangle \to \sigma' \Rightarrow P(c, \sigma, \sigma') \\
\text{iff} \\
[\forall \sigma \in \Sigma.\ P(\mathbf{skip}, \sigma, \sigma) \\
\& \\
\forall X \in \mathbf{Loc}, a \in \mathbf{Aexp}, \sigma \in \Sigma, m \in \mathbf{N}.\ \langle a, \sigma \rangle \to m \Rightarrow P(X := a, \sigma, \sigma[m/X]) \\
\& \\
\forall c_0, c_1 \in \mathbf{Com}, \sigma, \sigma', \sigma'' \in \Sigma. \\
\langle c_0, \sigma \rangle \to \sigma'' \ \&\ P(c_0, \sigma, \sigma'') \ \&\ \langle c_1, \sigma'' \rangle \to \sigma' \ \&\ P(c_1, \sigma'', \sigma') \Rightarrow P(c_0; c_1, \sigma, \sigma') \\
\& \\
\forall c_0, c_1 \in \mathbf{Com}, b \in \mathbf{Bexp}, \sigma, \sigma' \in \Sigma. \\
\langle b, \sigma \rangle \to \mathbf{true}\ \&\ \langle c_0, \sigma \rangle \to \sigma'\ \&\ P(c_0, \sigma, \sigma') \Rightarrow P(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma, \sigma') \\
\& \\
\forall c_0, c_1 \in \mathbf{Com}, b \in \mathbf{Bexp}, \sigma, \sigma' \in \Sigma. \\
\langle b, \sigma \rangle \to \mathbf{false}\ \&\ \langle c_1, \sigma \rangle \to \sigma'\ \&\ P(c_1, \sigma, \sigma') \Rightarrow P(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma, \sigma') \\
\& \\
\forall c \in \mathbf{Com}, b \in \mathbf{Bexp}, \sigma \in \Sigma. \\
\langle b, \sigma \rangle \to \mathbf{false} \Rightarrow P(\mathbf{while}\ b\ \mathbf{do}\ c, \sigma, \sigma) \\
\& \\
\forall c \in \mathbf{Com}, b \in \mathbf{Bexp}, \sigma, \sigma', \sigma'' \in \Sigma. \\
\langle b, \sigma \rangle \to \mathbf{true}\ \&\ \langle c, \sigma \rangle \to \sigma''\ \&\ P(c, \sigma, \sigma'')\ \& \\
\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'' \rangle \to \sigma'\ \&\ P(\mathbf{while}\ b\ \mathbf{do}\ c, \sigma'', \sigma') \\
\Rightarrow P(\mathbf{while}\ b\ \mathbf{do}\ c, \sigma, \sigma')].
\end{array}$$

As an example, we apply rule induction to show the intuitively obvious fact that if a location $Y$ does not occur in the left hand side of an assignment in a command $c$ then execution of $c$ cannot affect its value. Recall the definition of the locations $\mathrm{loc}_L(c)$ of a command $c$ given in Section 3.5.

**Proposition 4.7** *Let $Y \in \mathbf{Loc}$. For all commands $c$ and states $\sigma, \sigma'$,*

$$Y \notin loc_L(c) \ \&\ \langle c, \sigma\rangle \to \sigma' \Rightarrow \sigma(Y) = \sigma'(Y).$$

**Proof:** Let $P$ be the property given by:

$$P(c, \sigma, \sigma') \iff (Y \notin \mathrm{loc}_L(c) \Rightarrow \sigma(Y) = \sigma'(Y)).$$

We use rule induction on commands to show that

$$\forall c \in \mathbf{Com}, \sigma, \sigma' \in \Sigma. \quad \langle c, \sigma\rangle \to \sigma' \Rightarrow P(c, \sigma, \sigma').$$

Clearly $P(\mathbf{skip}, \sigma, \sigma)$ for any $\sigma \in \Sigma$.

Let $X \in \mathbf{Loc}, a \in \mathbf{Aexp}, \sigma \in \Sigma, m \in \mathbf{N}$. Assume $\langle a, \sigma\rangle \to m$. If $Y \notin \mathrm{loc}_L(X := a)$ then $Y \not\equiv X$, so $\sigma(Y) = \sigma[m/X](Y)$. Hence $P(X := a, \sigma, \sigma[m/X])$.

Let $c_0, c_1 \in \mathbf{Com}, \sigma, \sigma' \in \Sigma$. Assume

$$\langle c_0, \sigma\rangle \to \sigma'' \ \&\ P(c_0, \sigma, \sigma'') \ \&\ \langle c_1, \sigma''\rangle \to \sigma' \ \&\ P(c_1, \sigma'', \sigma'),$$

*i.e.*, that

$$\begin{aligned}&\langle c_0, \sigma\rangle \to \sigma'' \ \&\ (Y \notin \mathrm{loc}_L(c_0) \Rightarrow \sigma(Y) = \sigma''(Y)) \ \&\\ &\langle c_1, \sigma''\rangle \to \sigma' \ \&\ (Y \notin \mathrm{loc}_L(c_1) \Rightarrow \sigma''(Y) = \sigma'(Y)).\end{aligned}$$

Suppose $Y \notin \mathrm{loc}_L(c_0; c_1)$. Then, as $\mathrm{loc}_L(c_0; c_1) = \mathrm{loc}_L(c_0) \cup \mathrm{loc}_L(c_1)$, we obtain $Y \notin \mathrm{loc}_L(c_0)$ and $Y \notin \mathrm{loc}_L(c_1)$. Thus, from the assumption, $\sigma(Y) = \sigma''(Y) = \sigma'(Y)$. Hence $P(c_0; c_1, \sigma, \sigma')$.

We shall only consider one other case of rule instances.

Let $c \in \mathbf{Com}, b \in \mathbf{Bexp}, \sigma, \sigma', \sigma'' \in \Sigma$. Let $w \equiv \mathbf{while}\ b\ \mathbf{do}\ c$. Assume

$$\begin{aligned}\langle b, \sigma\rangle \to \mathbf{true} \ \&\ \langle c, \sigma\rangle \to \sigma'' \ \&\ P(c, \sigma, \sigma'') \ \&\\ \langle w, \sigma''\rangle \to \sigma' \ \&\ P(w, \sigma'', \sigma')\end{aligned}$$

*i.e.*,

$$\begin{aligned}\langle b, \sigma\rangle \to \mathbf{true} \ \&\ \langle c, \sigma\rangle \to \sigma'' \ \&\ (Y \notin \mathrm{loc}_L(c) \Rightarrow \sigma(Y) = \sigma''(Y)) \ \&\\ \langle w, \sigma''\rangle \to \sigma' \ \&\ (Y \notin \mathrm{loc}_L(w) \Rightarrow \sigma''(Y) = \sigma'(Y)).\end{aligned}$$

Suppose $Y \notin \mathrm{loc}_L(w)$. By the assumption $\sigma''(Y) = \sigma'(Y)$. Also, as $\mathrm{loc}_L(w) = \mathrm{loc}_L(c)$, we see $Y \notin \mathrm{loc}_L(c)$, so by the assumption $\sigma(Y) = \sigma''(Y)$. Thus $\sigma(Y) = \sigma'(Y)$. Hence $P(w, \sigma, \sigma')$.

The other cases are very similar and left as an exercise. □

We shall see many more proofs by rule induction in subsequent chapters. In general they will be smooth and direct arguments. Here are some more difficult exercises on using rule induction. As the first two exercises indicate applications of rule induction can sometimes be tricky.

**Exercise 4.8** Let $w \equiv$ **while true do skip**. Prove by special rule induction that

$$\forall \sigma, \sigma'.\ \langle w, \sigma \rangle \not\to \sigma'.$$

(Hint: Apply the special principle of rule induction restricting to the set

$$\{(w, \sigma, \sigma') \mid \sigma, \sigma' \in \Sigma\}$$

and take the property $P(w, \sigma, \sigma')$ to be constantly false.
It is interesting to compare the proof for this exercise with that of Proposition 3.12 in Section 3.4—proofs by rule induction can sometimes be less intuitive than proofs in which the form of derivations is considered.) □

Although rule induction can be used in place of induction on derivations it is no panacea; exclusive use of rule induction can sometimes make proofs longer and more confusing, as will probably become clear on trying the following exercise:

**Exercise 4.9** Take a simplified syntax of arithmetic expressions:

$$a ::= n \mid X \mid a_0 + a_1.$$

The evaluation rules of the simplified expressions are as before:

$$\langle n, \sigma \rangle \to n$$

$$\langle X, \sigma \rangle \to \sigma(X)$$

$$\frac{\langle a_0, \sigma \rangle \to n_0 \quad \langle a_1, \sigma \rangle \to n_1}{\langle a_0 + a_1, \sigma \rangle \to n}$$

where $n$ is the number which is the sum of $n_0$ and $n_1$.

By considering the unique form of derivations it is easy to see that $\langle n, \sigma \rangle \to m$ implies $m \equiv n$. Can you see how this follows by special rule induction? Use rule induction on the operational semantics (and not induction on derivations) to show that the evaluation

of expressions is deterministic.
(Hint: For the latter, take

$$P(a,\sigma,m) \iff_{def} \forall m' \in \mathbf{N}.\ \langle a,\sigma\rangle \to m' \Rightarrow m = m'$$

as induction hypothesis, and be prepared for a further use of (special) rule induction.)
An alternative proof, of Proposition 3.3 in Section 3.2, uses structural induction and considers the forms that derivations could take. How does the proof compare with that of Proposition 3.3? □

The next, fairly long, exercise proves the equivalence of two operational semantics.

**Exercise 4.10** (Long) One operational semantics is that of Chapter 2, based on the relation $\langle c,\sigma\rangle \to \sigma'$. The other is the one-step execution relation $\langle c,\sigma\rangle \to_1 \langle c',\sigma'\rangle$ mentioned previously in Section 2.6, but where, for simplicity, evaluation of expressions is treated in exactly the same way as in Chapter 2. For instance, for the sequencing of two commands there are the rules:

$$\frac{\langle c_0,\sigma\rangle \to_1 \langle c_0',\sigma'\rangle}{\langle c_0;c_1,\sigma\rangle \to_1 \langle c_0';c_1,\sigma'\rangle} \qquad \frac{\langle c_0,\sigma\rangle \to_1 \sigma'}{\langle c_0;c_1,\sigma\rangle \to_1 \langle c_1,\sigma'\rangle}$$

Start by proving the lemma

$$\langle c_0;c_1,\sigma\rangle \to_1^* \sigma' \text{ iff } \exists\sigma''.\ \langle c_0,\sigma\rangle \to_1^* \sigma'' \ \&\ \langle c_1,\sigma''\rangle \to_1^* \sigma',$$

for all commands $c_0, c_1$ and all states $\sigma, \sigma'$. Prove this in two stages. Firstly prove

$$\forall\sigma,\sigma'.\ [\langle c_0;c_1,\sigma\rangle \to_1^n \sigma' \Rightarrow \exists\sigma''.\ \langle c_0,\sigma\rangle \to_1^* \sigma'' \ \&\ \langle c_1,\sigma''\rangle \to_1^* \sigma']$$

by mathematical induction on $n$, the length of computation. Secondly prove

$$\forall\sigma,\sigma',\sigma''.\ [\langle c_0,\sigma\rangle \to_1^n \sigma'' \ \&\ \langle c_1,\sigma''\rangle \to_1^* \sigma' \Rightarrow \langle c_0;c_1,\sigma\rangle \to_1^* \sigma']$$

by mathematical induction on $n$, this time the length of the execution of $c_0$ from state $\sigma$. Conclude that the lemma holds. Now proceed to the proof of the theorem:

$$\forall\sigma,\sigma'.\ [\langle c,\sigma\rangle \to_1^* \sigma' \text{ iff } \langle c,\sigma\rangle \to \sigma'].$$

The "only if" direction of the proof can be done by structural induction on $c$, with an induction on the length of the computation in the case where $c$ is a while-loop. The "if" direction of the proof can be done by rule induction (or by induction on derivations). □

## 4.4 Operators and their least fixed points

There is another way to view a set defined by rules. A set of rule instances $R$ determines an operator $\widehat{R}$ on sets, which given a set $B$ results in a set

$$\widehat{R}(B) = \{y \mid \exists X \subseteq B.\ (X/y) \in R\}.$$

Use of the operator $\widehat{R}$ gives another way of saying a set is $R$-closed.

**Proposition 4.11** *A set $B$ is closed under $R$ iff $\widehat{R}(B) \subseteq B$.*

**Proof:** The fact follows directly from the definitions. □

The operator $\widehat{R}$ provides a way of building up the set $I_R$. The operator $\widehat{R}$ is *monotonic* in the sense that

$$A \subseteq B \Rightarrow \widehat{R}(A) \subseteq \widehat{R}(B).$$

If we repeatedly apply $\widehat{R}$ to the empty set $\emptyset$ we obtain the sequence of sets:

$$\begin{aligned} A_0 &= \widehat{R}^0(\emptyset) = \emptyset, \\ A_1 &= \widehat{R}^1(\emptyset) = \widehat{R}(\emptyset), \\ A_2 &= \widehat{R}(\widehat{R}(\emptyset)) = \widehat{R}^2(\emptyset), \\ &\vdots \\ A_n &= \widehat{R}^n(\emptyset), \\ &\vdots \end{aligned}$$

The set $A_1$ consists of all the conclusions of instances of axioms, and in general the set $A_{n+1}$ is all things which immediately follow by rule instances with premises in $A_n$. Clearly $\emptyset \subseteq \widehat{R}(\emptyset)$, *i.e.* $A_0 \subseteq A_1$. By the monotonicity of $\widehat{R}$ we obtain $\widehat{R}(A_0) \subseteq \widehat{R}(A_1)$, *i.e.* $A_1 \subseteq A_2$. Similarly we obtain $A_2 \subseteq A_3$ *etc.*. Thus the sequence forms a chain

$$A_0 \subseteq A_1 \subseteq \cdots \subseteq A_n \subseteq \cdots.$$

Taking $A = \bigcup_{n \in \omega} A_n$, we have:

**Proposition 4.12**
*(i) $A$ is $R$-closed.*
*(ii) $\widehat{R}(A) = A$.*
*(iii) $A$ is the least $R$-closed set.*

**Proof:**
(i) Suppose $(X/y) \in R$ with $X \subseteq A$. Recall $A = \bigcup_n A_n$ is the union of an increasing chain of sets. As $X$ is a finite set there is some $n$ such that $X \subseteq A_n$. (The set $X$ is either empty, whence $X \subseteq A_0$, or of the form $\{x_1, \ldots, x_k\}$. In the latter case, we have $x_1 \in A_{n_1}, \cdots, x_k \in A_{n_k}$ for some $n_1, \ldots, n_k$. Taking $n$ bigger than all of $n_1, \ldots, n_k$ we must have $X \subseteq A_n$ as the sequence $A_0, A_1, \ldots, A_n, \ldots$ is increasing.) As $X \subseteq A_n$ we obtain $y \in \widehat{R}(A_n) = A_{n+1}$. Hence $y \in \bigcup_n A_n = A$. Thus $A$ is closed under $R$.
(ii) By Proposition 4.11 the set $A$ is $R$-closed, so we already know that $\widehat{R}(A) \subseteq A$. We require the converse inclusion. Suppose $y \in A$. Then $y \in A_n$ for some $n > 0$. Thus $y \in \widehat{R}(A_{n-1})$. This means there is some $(X/y) \in R$ with $X \subseteq A_{n-1}$. But $A_{n-1} \subseteq A$ so $X \subseteq A$ with $(X/y) \in R$, giving $y \in \widehat{R}(A)$. We have established the required converse inclusion, $A \subseteq \widehat{R}(A)$. Hence $\widehat{R}(A) = A$.
(iii) We need to show that if $B$ is another $R$-closed set then $A \subseteq B$. Suppose $B$ is closed under $R$. Then $\widehat{R}(B) \subseteq B$. We show by mathematical induction that for all natural numbers $n \in \omega$

$$A_n \subseteq B.$$

The basis of the induction $A_0 \subseteq B$ is obviously true as $A_0 = \emptyset$. To show the induction step, assume $A_n \subseteq B$. Then

$$A_{n+1} = \widehat{R}(A_n) \subseteq \widehat{R}(B) \subseteq B,$$

using the facts that $\widehat{R}$ is monotonic and that $B$ is $R$-closed. □

Notice the essential part played in the proof of (i) by the fact that rule instances are finitary, *i.e.* in a rule instance $(X/y)$, the set of premises $X$ is finite.

It follows from (i) and (iii) that $A = I_R$, the set of elements for which there are $R$-derivations. Now (ii) says precisely that $I_R$ is a fixed point of $\widehat{R}$. Moreover, (iii) implies that $I_R$ is the *least fixed point* of $\widehat{R}$, *i.e.*

$$\widehat{R}(B) = B \Rightarrow I_R \subseteq B,$$

because if any other set $B$ is a fixed point it is closed under $R$, so $I_R \subseteq B$ by Proposition 4.1. The set $I_R$, defined by the rule instances $R$, is the least fixed point, $fix(\widehat{R})$, obtained by the construction

$$fix(\widehat{R}) =_{def} \bigcup_{n \in \omega} \widehat{R}^n(\emptyset).$$

Least fixed points will play a central role in the next chapter.

**Exercise 4.13** Given a set of rules $R$ define a different operator $\overline{R}$ by

$$\overline{R}(A) = A \cup \{y \mid \exists X \subseteq A.\ (X/y) \in R\}.$$

Clearly $\overline{R}$ is monotonic and in addition satisfies the property

$$A \subseteq \overline{R}(A).$$

An operator satisfying such a property is called *increasing*. Exhibit a monotonic operator which is not increasing. Show that given any set $A$ there is a least fixed point of $\overline{R}$ which includes $A$, and that this property can fail for monotonic operations. □

**Exercise 4.14** Let $R$ be a set of rule instances. Show that $\widehat{R}$ is *continuous* in the sense that

$$\bigcup_{n\in\omega} \widehat{R}(B_n) = \widehat{R}(\bigcup_{n\in\omega} B_n)$$

for any increasing chain of sets $B_0 \subseteq \cdots \subseteq B_n \subseteq \cdots$.
(The solution to this exercise is contained in the next chapter.) □

## 4.5 Further reading

This chapter has provided an elementary introduction to the mathematical theory of *inductive definitions*. A detailed, though much harder, account can be found in Peter Aczel's handbook chapter [4]—our treatment, with just finitary rules, avoids the use of ordinals. The term "rule induction" originates with the author's Cambridge lecture notes of 1984, and seems be catching on (the principle is well-known and, for instance, is called simply $R$-induction, for rules $R$, in [4]). This chapter has refrained from any recommendations about which style of argument to use in reasoning about operational semantics; whether to use rule induction or the often clumsier, but conceptually more straightforward, induction on derivations. In many cases it is a matter of taste.