Chapter 1

Inductive Definitions

This chapter discusses *inductive definitions* which are an indispensable tool in the study of programming languages. The reason why we need inductive definitions is not difficult to guess: a programming language may be thought of a system which is inhabited by *infinitely* many elements (or programs), and we wish to give a complete specification of it with a *finite* description; hence we need a mechanism of inductive definition by which a finite description is capable of yielding an infinite number of elements in the system. Those techniques related to inductive definitions also play a key role in investigating properties of programming languages. We will study these concepts with a few simple languages.

1.1 Inductive definitions of syntactic categories

An integral part of the definition of a programming language is its *syntax*, which answers the question of which program (*i.e.*, a sequence of characters) is recognizable by the parser and which program is not. Typically the syntax is specified by a number of *syntactic categories* such as expressions, types, and patterns. Below we discuss how to define syntactic categories inductively in a few simple languages.

Our first example defines a syntactic category nat of natural numbers:

$$\mathsf{nat} \qquad n \quad ::= \quad \mathtt{zero} \mid \mathtt{succ}\, n$$

Here nat is the name of the syntactic category being defined, and n is called a *non-terminal*. We read ::= as *"is defined as"* and | as *"or."* `zero` stands for "zero" and `succ` "successor." Thus the above definition is interpreted as:

> A natural number n is either `zero` or `succ` n' where n' is another natural number.

Note that nat is defined inductively: a natural number `succ` n' uses another natural number n', and thus nat uses the same syntactic category in its definition. Now the definition of nat produces an infinite collection of natural numbers such as:

$$\mathtt{zero},\ \mathtt{succ}\,\mathtt{zero},\ \mathtt{succ}\,\mathtt{succ}\,\mathtt{zero},\ \mathtt{succ}\,\mathtt{succ}\,\mathtt{succ}\,\mathtt{zero},\ \cdots$$

Thus nat specifies a language of natural numbers.

A syntactic category may refer to another syntactic category in its definition. For example, given the above definition of nat, the syntactic category tree below uses nat in its inductive definition:

$$\mathsf{tree} \qquad t \quad ::= \quad \mathtt{leaf}\, n \mid \mathtt{node}\,(t, n, t)$$

Then tree specifies a language of regular binary trees of natural numbers.

A similar but intrinsically different example is two syntactic categories that are mutually inductively defined. For example, we simultaneously define two syntactic categories even and odd of even and odd numbers as follows:

$$\begin{array}{lrcl} \textsf{even} & e & ::= & \texttt{zero} \mid \texttt{succ}\ o \\ \textsf{odd} & o & ::= & \texttt{succ}\ e \end{array}$$

Note that even and odd are *subcategories* of nat because every even number e or odd number o is also a natural number. Thus we may think of even and odd as nat satisfying certain properties.

Let us consider another example of defining a syntactic subcategory. First we define a syntactic category paren of strings of parentheses:

$$\textsf{paren} \quad s \ ::= \ \epsilon \mid (s \mid)s$$

ϵ stands for the empty string (*i.e.*, $\epsilon s = s = s\epsilon$). paren specifies a language of strings of parentheses with no constraint on the use of parentheses. Now we define a subcategory mparen of paren for those strings of matched parentheses:

$$\textsf{mparen} \quad s \ ::= \ \epsilon \mid (s) \mid s\ s$$

mparen is ambiguous in the sense that a string belonging to mparen may not be decomposed in a unique way (according to the definition of mparen). For example, $()()()$ may be thought of as either $()()$ concatenated with $()$ or $()$ concatenated as $()()$. The culprit is the third case $s\ s$ in the definition: a string of matched parentheses may be written as a sequence of substrings of matched parentheses, in which case there can be more than one way to split it into two substrings of matched parentheses. An alternative definition of lparen below eliminates ambiguity in mparen:

$$\textsf{lparen} \quad s \ ::= \ \epsilon \mid (s)\ s$$

The idea behind lparen is that the first parenthesis in a non-empty string s is a left parenthesis "(" which is paired with a unique occurrence of a right parenthesis ")".

An inductive definition of a syntactic category is a convenient way to specify a language. Even the syntax of a full-scale programming language (such as SML) uses essentially the same machinery. It is, however, not the best choice for *investigating properties of languages*. For example, how can we formally express that n belongs to nat if succ n belongs to nat, let alone prove it? Or how can we show that a string belonging to mparen indeed consists of matched parentheses? The notion of *judgment* comes into play to address such issues arising in inductive definitions.

1.2 Inductive definitions of judgments

A judgment is an object of knowledge, or simply a statement, that may or may not be provable. Here are a few examples:

- "$1-1$ *is equal to* 0" is a judgment which is always provable.
- "$1+1$ *is equal to* 0" is also a judgment which is never provable.
- "*It is raining*" is a judgment which is sometimes provable and sometimes not.
- "succ succ zero *belongs to the syntactic category* nat" is a judgment which is provable if nat is defined as shown in the previous section.

Then how do we prove a judgment? For example, on what basis do we assert that "$1-1$ *is equal to* 0" is always provable? We implicitly use arithmetic to prove "$1-1$ *is equal to* 0", but strictly speaking, arithmetic rules are not given for free — we first have to reformulate them as *inference rules*.

An inference rule consists of premises and a conclusion, and is written in the following form (where J stands for a judgment):

$$\frac{J_1 \quad J_2 \quad \cdots \quad J_n}{J}\ R$$

The inference rule, whose name is R, states that if J_1 through J_n (premises) hold, then J (conclusion) also holds. As a special case, an inference rule with no premise (*i.e.*, $n = 0$) is called an *axiom*. Here are a few examples of inference rules and axioms:

$$\frac{m \textit{ is equal to } l \quad l \textit{ is equal to } n}{m \textit{ is equal to } n} \qquad \frac{m \textit{ is equal to } n}{m+1 \textit{ is equal to } n+1}$$

$$\frac{\textit{My coat is wet}}{\textit{It is raining}} \qquad \frac{}{0 \textit{ is a natural number}} \qquad \frac{}{n \textit{ is equal to } n}$$

Judgments are a general concept that covers any form of knowledge: knowledge about weather, knowledge about numbers, knowledge about programming languages, and so on. Note that judgments alone are inadequate to justify the knowledge being conveyed — we also need inference rules for proving or refuting judgments. In other words, the definition of a judgment is complete *only when there are inference rules for proving or refuting it*. Without inference rules, there can be no meaning in the judgment. For example, without arithmetic rules, the statement "$1 - 1$ *is equal to* 0" is nothing more than nonsense and thus cannot be called a judgment.

Needless to say, judgments are strong enough to express membership in a syntactic category. As an example, let us recast the inductive definition of nat as a system of judgments and inference rules. We first introduce a judgment n nat:

$$n \ \mathsf{nat} \qquad \Leftrightarrow \qquad n \textit{ is a natural number}$$

We use the following two inference rules to prove the judgment n nat:

$$\frac{}{\texttt{zero}\ \mathsf{nat}}\ \textit{Zero} \qquad \frac{n\ \mathsf{nat}}{\texttt{succ}\ n\ \mathsf{nat}}\ \textit{Succ}$$

n in the second inference rule is called a *metavariable* which is just a placeholder for another sequence of `zero` and `succ` and is thus *not* part of the language consisting of `zero` and `succ`. That is, n itself (before being replaced by `succ zero`, for example) is not tested for membership in nat.

The judgment n nat is now defined inductively by the inference rules: the first inference rule is a base case because it is an axiom, and the second inference rule is an inductive case because the premise contains a judgment smaller in size than the one (of the same kind) in the conclusion. Now we can prove, for example, that `succ succ zero` nat holds with the following *derivation tree*, in which `succ succ zero` nat is the root and `zero` nat is the only leaf:

$$\cfrac{\cfrac{\cfrac{}{\texttt{zero}\ \mathsf{nat}}\ \textit{Zero}}{\texttt{succ}\ \texttt{zero}\ \mathsf{nat}}\ \textit{Succ}}{\texttt{succ}\ \texttt{succ}\ \texttt{zero}\ \mathsf{nat}}\ \textit{Succ}$$

Similarly we can rewrite the definition of the syntactic category tree in terms of judgments and inference rules:

$$t\ \mathsf{tree} \qquad \Leftrightarrow \qquad t \textit{ is a regular binary tree of natural numbers}$$

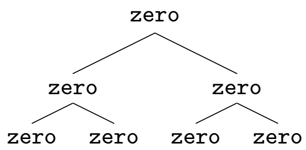
$$\frac{n\ \mathsf{nat}}{\texttt{leaf}\ n\ \mathsf{tree}}\ \textit{Leaf} \qquad \frac{t_1\ \mathsf{tree} \quad n\ \mathsf{nat} \quad t_2\ \mathsf{tree}}{\texttt{node}\ (t_1, n, t_2)\ \mathsf{tree}}\ \textit{Node}$$

A slightly more complicated example is a judgment that isolates full regular binary trees of natural numbers, as shown below. Note that there is no restriction on the form of judgment as long as its meaning is clarified by inference rules. We may even use English sentences as a valid form of judgment!

$$t\ \mathsf{ctree}\langle d \rangle \qquad \Leftrightarrow \qquad t \textit{ is a full regular binary tree of natural numbers of depth } d$$

$$\frac{n\ \mathsf{nat}}{\texttt{leaf}\ n\ \mathsf{ctree}\langle \texttt{zero} \rangle}\ \textit{Cleaf} \qquad \frac{t_1\ \mathsf{ctree}\langle d \rangle \quad n\ \mathsf{nat} \quad t_2\ \mathsf{ctree}\langle d \rangle}{\texttt{node}\ (t_1, n, t_2)\ \mathsf{ctree}\langle \texttt{succ}\ d \rangle}\ \textit{Cnode}$$

The following derivation tree (with inference rule names omitted) proves that



is a full regular binary tree of depth 2 (*i.e.*, of depth $\mathtt{succ}\ \mathtt{succ}\ \mathtt{zero}$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{zero\ nat}}}{\mathtt{leaf}\ \mathtt{zero}\ \mathsf{ctree}\langle\mathtt{zero}\rangle} \quad \dfrac{}{\mathsf{zero\ nat}} \quad \dfrac{\dfrac{}{\mathsf{zero\ nat}}}{\mathtt{leaf}\ \mathtt{zero}\ \mathsf{ctree}\langle\mathtt{zero}\rangle}}{\mathtt{node}\ (\mathtt{leaf}\ \mathtt{zero}, \mathtt{zero}, \mathtt{leaf}\ \mathtt{zero})\ \mathsf{ctree}\langle\mathtt{succ}\ \mathtt{zero}\rangle} \quad \dfrac{}{\mathsf{zero\ nat}} \quad \textit{(omitted)}}{\mathtt{node}\ (\mathtt{node}\ (\mathtt{leaf}\ \mathtt{zero}, \mathtt{zero}, \mathtt{leaf}\ \mathtt{zero}), \mathtt{zero}, \mathtt{node}\ (\mathtt{leaf}\ \mathtt{zero}, \mathtt{zero}, \mathtt{leaf}\ \mathtt{zero}))\ \mathsf{ctree}\langle\mathtt{succ}\ \mathtt{succ}\ \mathtt{zero}\rangle}$$

We can also show that $t = \mathtt{node}\ (\mathtt{leaf}\ \mathtt{zero}, \mathtt{zero}, \mathtt{node}\ (\mathtt{leaf}\ \mathtt{zero}, \mathtt{zero}, \mathtt{leaf}\ \mathtt{zero}))$ is not a full regular binary tree as we cannot prove $t\ \mathsf{ctree}\langle d\rangle$ for any natural number d:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{zero\ nat}} \quad d' = \mathtt{zero}}{\mathtt{leaf}\ \mathtt{zero}\ \mathsf{ctree}\langle d'\rangle} \quad \dfrac{}{\mathsf{zero\ nat}} \quad \dfrac{\cdots \quad d' = \mathtt{succ}\ d''}{\mathtt{node}\ (\mathtt{leaf}\ \mathtt{zero}, \mathtt{zero}, \mathtt{leaf}\ \mathtt{zero})\ \mathsf{ctree}\langle d'\rangle}}{\mathtt{node}\ (\mathtt{leaf}\ \mathtt{zero}, \mathtt{zero}, \mathtt{node}\ (\mathtt{leaf}\ \mathtt{zero}, \mathtt{zero}, \mathtt{leaf}\ \mathtt{zero}))\ \mathsf{ctree}\langle\mathtt{succ}\ d'\rangle}$$

It is easy to see why the proof fails: the left subtree of t requires $d' = \mathtt{zero}$ while the right subtree of t requires $d' = \mathtt{succ}\ d''$, and there is no way to solve two conflicting equations on d'.

As with the syntactic categories even and odd in the previous section, multiple judgments can be defined simultaneously. For example, here is the translation of the definition of even and odd into judgments and inference rules:

$$\begin{array}{rcl} n\ \mathsf{even} & \Leftrightarrow & n \textit{ is an even number} \\ n\ \mathsf{odd} & \Leftrightarrow & n \textit{ is an odd number} \end{array}$$

$$\dfrac{}{\mathsf{zero\ even}}\ \mathit{ZeroE} \qquad \dfrac{n\ \mathsf{odd}}{\mathsf{succ}\ n\ \mathsf{even}}\ \mathit{SuccE} \qquad \dfrac{n\ \mathsf{even}}{\mathsf{succ}\ n\ \mathsf{odd}}\ \mathit{SuccO}$$

1.3 Inductive proofs

We have learned how to specify systems using inductive definitions of syntactic categories or judgments, or *inductive systems* of syntactic categories or judgments. While it is powerful enough to specify even full-scale programming languages (*i.e.*, their syntax and semantics), the mechanism of inductive definition alone is hardly useful unless the resultant system is shown to exhibit desired properties. That is, we cannot just specify a system using an inductive definition and then immediately use it without proving any interesting properties. For example, our intuition says that every string in the syntactic category mparen has the same number of left and right parentheses, but the definition of mparen itself does not automatically prove this property; hence we need to formally prove this property ourselves in order to use mparen as a language of strings of matched parentheses. As another example, consider the inductive definition of the judgments n even and n odd. The definition seems to make sense, but it still remains to formally prove that n in n even indeed represents an even number and n in n odd an odd number.

There is another important reason why we need to be able to prove properties of inductive systems. An inductive system is often so complex that its soundness, *i.e.* its definition being devoid of any inconsistencies, may not be obvious at all. In such a case, we usually set out to prove a property that is supposed to hold in the system. Then each flaw in the definition that destroys the property, if any, manifests itself at some point in the proof (because it is impossible to complete the proof). For example, an expression in a functional language is supposed to evaluate to a value of the same type, but this property (called *type preservation*) is usually not obvious at all. By attempting to prove type preservation, we can either locate flaws in the definition or partially ensure that the system is sound. Thus proving properties of an inductive system is the most effective aid in fixing errors in the definition.

First we will study a principle called *structural induction* for proving properties of inductive systems of syntactic categories. Next we will study another principle called *rule induction* for proving properties of inductive systems of judgments. Since an inductive system of syntactic category is a simplified presentation of a corresponding inductive system of judgments, structural induction is in fact a special case of rule induction. Nevertheless structural induction deserves separate treatment because of the role of syntactic categories in the study of programming languages.

1.3.1 Structural induction

The principle of structural induction states that a property of a syntactic category may be proven inductively by analyzing the structure of its definition: for each base case, we show that the property holds without making any assumption; for each inductive case, we first assume that the property holds for each smaller element in it and then prove the property holds for the entire case.

A couple of examples will clarify the concept. Consider the syntactic category nat of natural numbers. We wish to prove that $P(n)$ holds for every natural number n. Examples of $P(n)$ are:

- n has a successor.
- n is `zero` or has a predecessor n' (*i.e.*, $\texttt{succ}\ n' = n$).
- n is a product of prime numbers (where definitions of products and prime numbers are assumed to be given).

By structural induction, we prove the following two statements:

- $P(\texttt{zero})$ holds.
- If $P(n)$ holds, then $P(\texttt{succ}\ n)$ also holds.

The first statement is concerned with the base case in which `zero` has no smaller element in it; hence we prove $P(\texttt{zero})$ without any assumption. The second statement is concerned with the inductive case in which $\texttt{succ}\ n$ has a smaller element n in it; hence we first assume, as an *induction hypothesis*, that $P(n)$ holds and then prove that $P(\texttt{succ}\ n)$ holds. The above instance of structural induction is essentially the same as the principle of mathematical induction.

As another example, consider the syntactic category tree of regular binary trees. In order to prove that $P(t)$ holds for every regular binary tree t, we need to prove the following two statements:

- $P(\texttt{leaf}\ n)$ holds.
- If $P(t_1)$ and $P(t_2)$ hold as induction hypotheses, then $P(\texttt{node}\ (t_1, n, t_2))$ also holds.

The above instance of structural induction is usually called *tree induction*.

As a concrete example of an inductive proof by structural induction, let us prove that every string belonging to the syntactic category mparen has the same number of left and right parentheses. (Note that we are not proving that mparen specifies a language of strings of matched parentheses.) We first define two auxiliary functions *left* and *right* to count the number of left and right parentheses. For visual clarity, we write $\mathit{left}[s]$ and $\mathit{right}[s]$ instead of $\mathit{left}(s)$ and $\mathit{right}(s)$. (We do not define *left* and *right* on the syntactic category paren because the purpose of this example is to illustrate structural induction rather than to prove an interesting property of mparen.)

$$
\begin{array}{rcl}
\mathit{left}[\epsilon] & = & 0 \\
\mathit{left}[(s)] & = & 1 + \mathit{left}[s] \\
\mathit{left}[s_1\ s_2] & = & \mathit{left}[s_1] + \mathit{left}[s_2] \\
\mathit{right}[\epsilon] & = & 0 \\
\mathit{right}[(s)] & = & 1 + \mathit{right}[s] \\
\mathit{right}[s_1\ s_2] & = & \mathit{right}[s_1] + \mathit{right}[s_2]
\end{array}
$$

Now let us interpret $P(s)$ as "$\mathit{left}[s] = \mathit{right}[s]$." Then we want to prove that if s belongs to mparen, written as $s \in \mathsf{mparen}$, then $P(s)$ holds.

Theorem 1.1. *If $s \in \mathsf{mparen}$, then $\mathit{left}[s] = \mathit{right}[s]$.*

Proof. By structural induction on s.

Each line below corresponds to a single step in the proof. It is written in the following format:

conclusion *justification*

This format makes it easy to read the proof because in most cases, we want to see the conclusion first rather than its justification.

Case $s = \epsilon$:
$\mathit{left}[\epsilon] = 0 = \mathit{right}[\epsilon]$

Case $s = (s')$:
$\mathit{left}[s'] = \mathit{right}[s']$ by induction hypothesis on s'
$\mathit{left}[s] = 1 + \mathit{left}[s'] = 1 + \mathit{right}[s'] = \mathit{right}[s]$ from $\mathit{left}[s'] = \mathit{right}[s']$

Case $s = s_1\ s_2$:
$\mathit{left}[s_1] = \mathit{right}[s_1]$ by induction hypothesis on s_1
$\mathit{left}[s_2] = \mathit{right}[s_2]$ by induction hypothesis on s_2
$\mathit{left}[s_1\ s_2] = \mathit{left}[s_1] + \mathit{left}[s_2] = \mathit{right}[s_1] + \mathit{right}[s_2] = \mathit{right}[s_1\ s_2]$
from $\mathit{left}[s_1] = \mathit{right}[s_1]$ and $\mathit{left}[s_2] = \mathit{right}[s_2]$
$\square$

In the proof above, we may also say "by induction on the structure of s" instead of "by structural induction on s."

1.3.2 Rule induction

The principle of rule induction is similar to the principle of structural induction except that it is applied to derivation trees rather than definitions of syntactic categories. Consider an inductive definition of a judgment J with two inference rules:

$$\frac{}{J_b}\ R_{\mathsf{base}} \qquad \frac{J_1 \quad J_2 \quad \cdots \quad J_n}{J_i}\ R_{\mathsf{ind}}$$

We want to show that whenever J holds, another judgment $P(J)$ holds where $P(J)$ is a new form of judgment parameterized over J. For example, when J is "$n\ \mathsf{nat}$", $P(J)$ may be "either $n\ \mathsf{even}$ or $n\ \mathsf{odd}$." To this end, we prove the following two statements:

- $P(J_b)$ holds.
- If $P(J_1)$, $P(J_2)$, $\cdots$, and $P(J_n)$ hold as induction hypotheses, then $P(J_i)$ holds.

By virtue of the first statement, the following inference rule makes sense because we can always prove $P(J_b)$:

$$\frac{}{P(J_b)}\ R'_{\mathsf{base}}$$

The following inference rule also makes sense because of the second statement: it states that if $P(J_1)$ through $P(J_n)$ hold, then $P(J_i)$ also holds, which is precisely what the second statement proves:

$$\frac{P(J_1) \quad P(J_2) \quad \cdots \quad P(J_n)}{P(J_i)}\ R'_{\mathsf{ind}}$$

Now, for any derivation tree for J using the rules R_{base} and R_{ind}, we can prove $P(J)$ using the rules R'_{base} and R'_{ind}:

$$\frac{}{J_b}\ R_{\mathsf{base}} \quad \Longrightarrow \quad \frac{}{P(J_b)}\ R'_{\mathsf{base}}$$

$$\frac{\begin{array}{cccc}\vdots & \vdots & \cdots & \vdots \\ J_1 & J_2 & & J_n\end{array}}{J_i}\ R_{\mathsf{ind}} \quad \Longrightarrow \quad \frac{\begin{array}{cccc}\vdots & \vdots & \cdots & \vdots \\ P(J_1) & P(J_2) & & P(J_n)\end{array}}{P(J_i)}\ R'_{\mathsf{ind}}$$

In other words, J always implies $P(J)$. A generalization of the above strategy is the principle of rule induction.

As a trivial example, let us prove that n nat implies either n even or n odd. We let $P(n\ \mathsf{nat})$ be "either n even or n odd" and apply the principle of rule induction. The two rules *Zero* and *Succ* require us to prove the following two statements:

- $P(\mathtt{zero}\ \mathsf{nat})$ holds. That is, for the case where the rule *Zero* is used to prove n nat, we have $n = \mathtt{zero}$ and thus prove $P(\mathtt{zero}\ \mathsf{nat})$.
- If $P(n'\ \mathsf{nat})$ holds, $P(\mathtt{succ}\ n'\ \mathsf{nat})$ holds. That is, for the case where the rule *Succ* is used to prove n nat, we have $n = \mathtt{succ}\ n'$ and thus prove $P(\mathtt{succ}\ n'\ \mathsf{nat})$ using the induction hypothesis $P(n'\ \mathsf{nat})$.

According to the definition of $P(J)$, the two statements are equivalent to:

- Either zero even or zero odd holds.
- If either n' even or n' odd holds, then either succ n' even or succ n' odd holds.

A formal inductive proof proceeds as follows:

Theorem 1.2. *If n nat, then either n even or n odd.*

Proof. By rule induction on the judgment n nat.

It is of utmost importance that we apply the principle of rule induction to the *judgment* n nat rather than the natural number n. In other words, we analyze the structure of the proof of n nat, *not the structure of n.* If we analyze the structure of n, the proof degenerates to an example of structural induction! Hence we may also say "by induction on the structure of the proof of n nat" instead of "by rule induction on the judgment n nat."

Case $\dfrac{}{\mathtt{zero}\ \mathsf{nat}}\ \mathit{Zero}$ (where n happens to be equal to zero):
(This is the case where n nat is proven by applying the rule *Zero*. It is not obtained as a case where n is equal to zero, since we are not analyzing the structure of n. Note also that we do *not* apply the induction hypothesis because the premise has no judgment.)
zero even — by the rule *ZeroE*

Case $\dfrac{n'\ \mathsf{nat}}{\mathtt{succ}\ n'\ \mathsf{nat}}\ \mathit{Succ}$ (where n happens to be equal to succ n'):
(This is the case where n nat is proven by applying the rule *Succ*.)
n' even or n' odd — by induction hypothesis
succ n' odd or succ n' even — by the rule *SuccO* or *SuccE*
□

Rule induction can also be applied simultaneously to two or more judgments. As an example, let us prove that n in n even represents an even number and n in n odd an odd number. We use the rules *ZeroE*,

$SuccE$, and $SuccO$ in Section 1.2 along with the following inference rules using a judgment n double n':

$$\frac{}{\texttt{zero}\ \mathsf{double}\ \texttt{zero}}\ Dzero \qquad \frac{n\ \mathsf{double}\ n'}{\texttt{succ}\ n\ \mathsf{double}\ \texttt{succ}\ \texttt{succ}\ n'}\ Dsucc$$

Intuitively n double n' means that n' is a double of n (*i.e.*, $n' = 2 \times n$). The properties of even and odd numbers are stated in the following theorem:

Theorem 1.3.
If n even, then there exists n' such that n' double n.
If n odd, there there exist n' and n'' such that n' double n'' and $\texttt{succ}\ n'' = n$.

The proof of the theorem follows the same pattern of rule induction as in previous examples except that $P(J)$ distinguishes between the two cases $J = n$ even and $J = n$ odd:

- $P(n$ even$)$ is "there exists n' such that n' double n."
- $P(n$ odd$)$ is "there there exist n' and n'' such that n' double n'' and $\texttt{succ}\ n'' = n$."

An inductive proof of the theorem proceeds as follows:

Proof of Theorem 1.3. By simultaneous rule induction on the judgments n even and n odd.

Case $\frac{}{\texttt{zero}\ \mathsf{even}}\ ZeroE$ where $n = \texttt{zero}$:
$\texttt{zero}$ double $\texttt{zero}$ — by the rule $Dzero$
We let $n' = \texttt{zero}$.

Case $\frac{n_p\ \mathsf{odd}}{\texttt{succ}\ n_p\ \mathsf{even}}\ SuccE$ where $n = \texttt{succ}\ n_p$:
n'_p double n''_p and $\texttt{succ}\ n''_p = n_p$ — by induction hypothesis
$\texttt{succ}\ n'_p$ double $\texttt{succ}\ \texttt{succ}\ n''_p$ — by the rule $Dsucc$ with n'_p double n''_p
$\texttt{succ}\ n'_p$ double n — from $\texttt{succ}\ \texttt{succ}\ n''_p = \texttt{succ}\ n_p = n$
We let $n' = \texttt{succ}\ n'_p$.

Case $\frac{n_p\ \mathsf{even}}{\texttt{succ}\ n_p\ \mathsf{odd}}\ SuccO$ where $n = \texttt{succ}\ n_p$:
n'_p double n_p — by induction hypothesis
We let $n' = n'_p$ and $n'' = n_p$ — from $n = \texttt{succ}\ n_p$

□

1.4 Techniques for inductive proofs

An inductive proof is not always as straightforward as the proof of Theorem 1.2. For example, the theorem being proven may be simply false! In such a case, the proof attempt (which will eventually fail) may help us to extract a counterexample of the theorem. If the theorem is indeed provable (or is believed to be provable) but a direct proof attempt fails, we can try a common technique for inductive proofs. Below we illustrate three such techniques: introducing a lemma, generalizing the theorem, and proving by the principle of inversion.

1.4.1 Using a lemma

We recast the definition of the syntactic categories mparen and lparen as a system of judgments and inference rules:

$$\frac{}{\epsilon\ \mathsf{mparen}}\ Meps \qquad \frac{s\ \mathsf{mparen}}{(s)\ \mathsf{mparen}}\ Mpar \qquad \frac{s_1\ \mathsf{mparen} \quad s_2\ \mathsf{mparen}}{s_1\ s_2\ \mathsf{mparen}}\ Mseq$$

$$\frac{}{\epsilon\ \mathsf{lparen}}\ \mathit{Leps} \qquad \frac{s_1\ \mathsf{lparen} \quad s_2\ \mathsf{lparen}}{(s_1)\ s_2\ \mathsf{lparen}}\ \mathit{Lseq}$$

Our goal is to show that s mparen implies s lparen. It turns out that a direct proof attempt by rule induction fails and that we need a lemma. To informally explain why we need a lemma, consider the case where the rule *Mseq* is used to prove s mparen. We may write $s = s_1\ s_2$ with s_1 mparen and s_2 mparen. By induction hypothesis on s_1 mparen and s_2 mparen, we may conclude s_1 lparen and s_2 lparen. From s_1 lparen, there are two subcases to consider:

- If $s_1 = \epsilon$, then $s = s_1\ s_2 = s_2$ and s_2 lparen implies s lparen.
- If $s_1 = (s'_1)\ s''_1$ with s'_1 lparen and s''_1 lparen, then $s = (s'_1)\ s''_1\ s_2$.

In the second subcase, it is necessary to prove $s''_1\ s_2$ lparen from s''_1 lparen and s_2 lparen, which is not addressed by what is being proven (and is not obvious). Thus the following lemma needs to be proven first:

Lemma 1.4. *If s lparen and s' lparen, then $s\ s'$ lparen.*

Then how do we prove the above lemma by rule induction? The lemma does not seem to be provable by rule induction because it does not have the form *"If J holds, then $P(J)$ holds"* — the *If* part contains two judgments! It turns out, however, that rule induction can be applied exactly in the same way. The trick is to interpret the statement in the lemma as:

If s lparen, then s' lparen implies $s\ s'$ lparen.

Then we apply rule induction to the judgment s lparen with $P(s\ \mathsf{lparen})$ being "s' lparen implies $s\ s'$ lparen." An inductive proof of the lemma proceeds as follows:

Proof of Lemma 1.4. By rule induction on the judgment s lparen. Keep in mind that the induction hypothesis on s lparen yields "s' lparen implies $s\ s'$ lparen." Consequently, if s' lparen is already available as an assumption, the induction hypothesis on s lparen yields $s\ s'$ lparen.

Case $\dfrac{}{\epsilon\ \mathsf{lparen}}\ \mathit{Leps}$ where $s = \epsilon$:

s' lparen	assumption
$s\ s' = \epsilon\ s' = s'$	
$s\ s'$ lparen	from s' lparen

Case $\dfrac{s_1\ \mathsf{lparen} \quad s_2\ \mathsf{lparen}}{(s_1)\ s_2\ \mathsf{lparen}}\ \mathit{Lseq}$ where $s = (s_1)\ s_2$:

s' lparen	assumption
$s\ s' = (s_1)\ s_2\ s'$	
"s' lparen implies $s_2\ s'$ lparen"	by induction hypothesis on s_2 lparen
$s_2\ s'$ lparen	from the assumption s' lparen
$(s_1)\ s_2\ s'$ lparen	by the rule *Lseq* with s_1 lparen and $s_2\ s'$ lparen

□

Exercise 1.5. Can you prove Lemma 1.4 by rule induction on the judgment s' lparen?

Now we are ready to prove that s mparen implies s lparen.

Theorem 1.6. *If s mparen, then s lparen.*

Proof. By rule induction on the the judgment s mparen.

Case $\dfrac{}{\epsilon\ \mathsf{mparen}}\ \mathit{Meps}$ where $s = \epsilon$:

ϵ lparen	by the rule *Leps*

Case $\dfrac{s' \ \mathsf{mparen}}{(s') \ \mathsf{mparen}} \ \mathit{Mpar}$ where $s = (s')$:

$s' \ \mathsf{lparen}$ by induction hypothesis

$(s') \ \mathsf{lparen}$ from $\dfrac{s' \ \mathsf{lparen} \quad \dfrac{}{\epsilon \ \mathsf{lparen}} \ \mathit{Leps}}{(s') \ \epsilon \ \mathsf{lparen}} \ \mathit{Lseq}$ and $(s') = (s') \ \epsilon$

Case $\dfrac{s_1 \ \mathsf{mparen} \quad s_2 \ \mathsf{mparen}}{s_1 \ s_2 \ \mathsf{mparen}} \ \mathit{Mseq}$ where $s = s_1 \ s_2$:

$s_1 \ \mathsf{lparen}$ by induction hypothesis on $s_1 \ \mathsf{mparen}$
$s_2 \ \mathsf{lparen}$ by induction hypothesis on $s_2 \ \mathsf{mparen}$
$s_1 \ s_2 \ \mathsf{lparen}$ by Lemma 1.4

□

1.4.2 Generalizing a theorem

We have seen in Theorem 1.1 that if a string s belongs to the syntactic category mparen, or if s mparen holds, s has the same number of left and right parentheses, *i.e.*, $\mathit{left}[s] = \mathit{right}[s]$. The result, however, does not prove that s is a string of matched parentheses because it does not take into consideration positions of matching parentheses. For example, $s =)($ satisfies $\mathit{left}[s] = \mathit{right}[s]$, but is not a string of matched parentheses because the left parenthesis appears after its corresponding right parenthesis.

In order to be able to recognize strings of matched parentheses, we introduce a new judgment $k \rhd s$ where k is a non-negative integer:

$$\begin{array}{lcl} k \rhd s & \Leftrightarrow & \textit{k left parentheses concatenated with s form a string of matched parentheses} \\ & \Leftrightarrow & \underbrace{((\cdots(}_{k} \ \textit{s is a string of matched parentheses} \end{array}$$

The idea is that we scan a given string from left to right and keep counting the number of left parentheses that have not yet been matched with corresponding right parentheses. Thus we begin with $k = 0$, increment k each time a left parenthesis is encountered, and decrement k each time a right parenthesis is encountered:

$$\frac{}{0 \rhd \epsilon} \ \mathit{Peps} \qquad \frac{k+1 \rhd s}{k \rhd (s} \ \mathit{Pleft} \qquad \frac{k-1 \rhd s \quad k > 0}{k \rhd)s} \ \mathit{Pright}$$

The second premise $k > 0$ in the rule *Pright* ensures that in any prefix of a given string, the number of right parentheses may not exceed the number of left parentheses. Now a judgment $0 \rhd s$ expresses that s is a string of matched parentheses. Here are a couple of examples:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{0 \rhd \epsilon} \ \mathit{Peps} \quad 1 > 0}{1 \rhd)} \ \mathit{Pright} \quad 2 > 0}{2 \rhd))} \ \mathit{Pright}}{1 \rhd ())} \ \mathit{Pleft}}{0 \rhd (())} \ \mathit{Pleft} \qquad \dfrac{\dfrac{\begin{array}{c}(\textit{the rule Pright is not applicable because } 0 \not> 0) \\ 0 \rhd)(\end{array}}{1 \rhd))(} \ \mathit{Pright}}{0 \rhd ())(} \ \mathit{Pleft}$$

Note that while an inference rules is usually read from the premise to the conclusion, *i.e.*, "if the premise holds, then the conclusion follows," the above rules are best read from the conclusion to the premise: "in order to prove the conclusion, we prove the premise instead." For example, the rule *Peps* may be read as "in order to prove $0 \rhd \epsilon$, we do not have to prove anything else," which implies that $0 \rhd \epsilon$ automatically holds; the rule *Pleft* may be read as "in order to prove $k \rhd (s$, we only have to prove $k+1 \rhd s$." This bottom-up reading of the rules corresponds to the left-to-right direction of scanning a string. For example, a proof of $0 \rhd (())$ would proceed as the following sequence of judgments in which the given string is scanned from left to right:

$$0 \rhd (()) \quad \longrightarrow \quad 1 \rhd ()) \quad \longrightarrow \quad 2 \rhd)) \quad \longrightarrow \quad 1 \rhd) \quad \longrightarrow \quad 0 \rhd \epsilon$$

Exercise 1.7. Rewrite the inference rules for the judgment $k \rhd s$ so that they are best read from the premise to the conclusion. Show that the top-down reading corresponds to the right-to-left direction of scanning a string.

Now we wish to prove that a string s satisfying $0 \rhd s$ indeed belongs to the syntactic category mparen:

Theorem 1.8. *If $0 \rhd s$, then s mparen.*

It is easy to see that a direct proof of Theorem 1.8 by rule induction fails. For example, when $0 \rhd (s$ follows from $1 \rhd s$ by the rule *Pleft*, we cannot apply the induction hypothesis to the premise because it does not have the form $0 \rhd s'$. What we need is, therefore, a generalization of Theorem 1.8 that covers all cases of the judgment $k \rhd s$ instead of a particular case $k = 0$:

Lemma 1.9. *If $k \rhd s$, then $\underbrace{((\cdots(}_{k} s$ mparen.*

Lemma 1.9 formally verifies the intuition behind the general form of the judgment $k \rhd s$. Then Theorem 1.8 is obtained as a corollary of Lemma 1.9.

The proof of Lemma 1.9 requires another lemma whose proof is left as an exercise (see Exercise 1.14):

Lemma 1.10. *If $\underbrace{((\cdots(}_{k} s$ mparen, then $\underbrace{((\cdots(}_{k}()s$ mparen.*

Proof of Lemma 1.9. By rule induction on the judgment $k \rhd s$.

Case $\dfrac{}{0 \rhd \epsilon}\ \mathit{Peps}$ where $k = 0$ and $s = \epsilon$:

ϵ mparen — by the rule Meps

$\underbrace{((\cdots(}_{k} s$ mparen — from $\underbrace{((\cdots(}_{k} s = \epsilon$

Case $\dfrac{k+1 \rhd s'}{k \rhd (s'}\ \mathit{Pleft}$ where $s = (s'$:

$\underbrace{((\cdots(}_{k+1} s'$ mparen — by induction hypothesis on $k+1 \rhd s'$

$\underbrace{((\cdots(}_{k} s$ mparen — from $\underbrace{((\cdots(}_{k+1} s' = \underbrace{((\cdots(}_{k}(s' = \underbrace{((\cdots(}_{k} s$

Case $\dfrac{k-1 \rhd s' \quad k > 0}{k \rhd)s'}\ \mathit{Pright}$ where $s =)s'$:

$\underbrace{((\cdots(}_{k-1} s'$ mparen — by induction hypothesis on $k-1 \rhd s'$

$\underbrace{((\cdots(}_{k-1}()s'$ mparen — by Lemma 1.10

$\underbrace{((\cdots(}_{k} s$ mparen — from $\underbrace{((\cdots(}_{k-1}()s' = \underbrace{((\cdots()}_{k}s' = \underbrace{((\cdots(}_{k} s$

□

It is important that generalizing a theorem is different from introducing a lemma. We introduce a lemma when the induction hypothesis is applicable to all premises in an inductive proof, but the conclusion to be drawn is not a direct consequence of induction hypotheses. Typically such a lemma, which fills the gap between induction hypotheses and the conclusion, requires another inductive proof and is thus proven separately. In contrast, we generalize a theorem when the induction hypothesis is not applicable to some premises and an inductive proof does not even work. Introducing a lemma is to no avail here, since the induction hypothesis is applicable only to premises of inference rules and nothing else (*e.g.*, judgments proven by a lemma). Thus we generalize the theorem so that a direct inductive proof works. (The proof of the generalized theorem may require us to introduce a lemma, of course.)

To generalize a theorem is essentially to find a theorem that is harder to prove than, but immediately implies the original theorem. (In this regard, we can also say that we "strengthen" the theorem.) There is no particular recipe for generalizing a theorem, and some problem requires a deep insight into the judgment to which the induction hypothesis is to be applied. In many cases, however, identifying an invariant on the judgment under consideration gives a clue on how to generalize the theorem. For example, Theorem 1.8 deals with a special case of the judgment $k \rhd s$, and its generalization in Lemma 1.9 precisely expresses what the judgment $k \rhd s$ means.

1.4.3 Proof by the principle of inversion

Consider an inference rule $\frac{J_1 \quad J_2 \quad \cdots \quad J_n}{J}\ R$. In order to apply the rule R, we first have to establish proofs of all the premises J_1 through J_n, from which we may judge that the conclusion J also holds. An alternative way of reading the rule R is that in order to prove J, it suffices to prove $J_1, \cdots, J_n$. In either case, it is the premises, not the conclusion, that we have to prove first.

Now assume the existence of a proof of the conclusion J. That is, we assume that J is provable, but we may not have a concrete proof of it. Since the rule R is applied in the top-down direction, the existence of a proof of J does not license us to conclude that the premises $J_1, \cdots, J_n$ are also provable. For example, there may be another rule, say $\frac{J_1' \quad J_2' \quad \cdots \quad J_m}{J}\ R'$, that deduces the same conclusion, but using different premises. In this case, we cannot be certain that the rule R has been applied at the final step of the proof of J, and the existence of proofs of $J_1, \cdots, J_n$ is not guaranteed.

If, however, the rule R is the *only* way to prove the conclusion J, we may safely "invert" the rule R and deduce the premises $J_1, \cdots, J_n$ from the existence of a proof of J. That is, since the rule R is the only way to prove J, the existence of a proof of J is subject to the existence of proofs of all the premises of the rule R. Such a use of an inference rule in the bottom-up direction is called the *principle of inversion.*

As an example, let us prove that if $\mathtt{succ}\ n$ is a natural number, so is n:

Proposition 1.11. *If* $\mathtt{succ}\ n$ nat*, then* n nat.

We begin with an assumption that $\mathtt{succ}\ n$ nat holds. Since the only way to prove $\mathtt{succ}\ n$ nat is by the rule *Succ*, $\mathtt{succ}\ n$ nat must have been derived from n nat by the principle of inversion:

$$\frac{n\ \mathsf{nat}}{\mathtt{succ}\ n\ \mathsf{nat}}\ Succ$$

Thus there must be a proof of n nat whenever there exists a proof of $\mathtt{succ}\ n$ nat, which completes the proof of Proposition 1.11.

1.5 Derivable rules and admissible rules

As shown in Section 1.2, judgments are defined with a certain (fixed) number of inference rules. When put together, these inference rules justify new inference rules which may in turn be added to the system. The new inference rules do *not* change the characteristics of the system because they can all be justified by the original inference rules, but may considerably facilitate the study of the system. For example, when multiplying two integers, we seldom employ the basic arithmetic rules, which can be thought of as original inference rules; instead we mostly use the rules of the multiplication table, which can be thought of as new inference rules.

There are two ways to introduce new inference rules: as *derivable rules* and as *admissible rules*. A derivable rule is one in which the gap between the premise and the conclusion can be bridged by a derivation tree. In other words, there always exists a derivation tree that uses the premise to prove the conclusion. As an example, consider the following inference rule which states that if n is a natural number, so is $\mathtt{succ}\ \mathtt{succ}\ n$:

$$\frac{n\ \mathsf{nat}}{\mathtt{succ}\ \mathtt{succ}\ n\ \mathsf{nat}}\ \mathit{Succ2}$$

The rule *Succ2* is derivable because we can justify it with the following derivation tree:

$$\cfrac{\cfrac{n\ \mathsf{nat}}{\mathtt{succ}\ n\ \mathsf{nat}}\ \mathit{Succ}}{\mathtt{succ}\ \mathtt{succ}\ n\ \mathsf{nat}}\ \mathit{Succ}$$

Now we may use the rule *Succ2* as if it were an original inference rule; when asked to justify its use, we can just present the above derivation tree.

An admissible rule is one in which the premise implies the conclusion. A derivable rule is certainly an admissible rule because of the derivability of the conclusion from the premise. There are, however, admissible rules that are not derivable rules. (Otherwise why would we distinguish between derivable and admissible rules?) Consider the following inference rule which states that if $\mathtt{succ}\ n$ is a natural number, so is n:

$$\frac{\mathtt{succ}\ n\ \mathsf{nat}}{n\ \mathsf{nat}}\ \mathit{Succ}^{-1}$$

First observe that the rule Succ^{-1} is not derivable: the only way to derive $n\ \mathsf{nat}$ from $\mathtt{succ}\ n\ \mathsf{nat}$ is by the rule *Succ*, but the premise of the rule *Succ* is smaller than its conclusion whereas $\mathtt{succ}\ n\ \mathsf{nat}$ is larger than $n\ \mathsf{nat}$. Now suppose that the premise $\mathtt{succ}\ n\ \mathsf{nat}$ holds. By Proposition 1.11, there must be a proof of $n\ \mathsf{nat}$, which justifies the rule Succ^{-1} as an admissible rule.

An important property of derivable rules is that they remain valid even when the system is augmented with new inference rules. For example, the rule *Succ2* remains valid no matter how many new inference rules are added to the system because the derivation of $\mathtt{succ}\ \mathtt{succ}\ n\ \mathsf{nat}$ from $n\ \mathsf{nat}$ is always possible thanks to the rule *Succ* (which is not removed from the system). In contrast, admissible rules may become invalid when new inference rules are introduced. For example, suppose that the system introduces a new (bizarre) inference rule:

$$\frac{n\ \mathsf{tree}}{\mathtt{succ}\ n\ \mathsf{nat}}\ \mathit{Bizarre}$$

The rule *Bizarre* invalidates the rule Succ^{-1} because the rule *Succ* is no longer the only way to prove $\mathtt{succ}\ n\ \mathsf{nat}$ and $\mathtt{succ}\ n\ \mathsf{nat}$ fails to guarantees $n\ \mathsf{nat}$. Therefore the validity of an admissible rule must be checked each time a new inference rule is introduced.

1.6 Exercises

Exercise 1.12. Prove the converse of Theorem 1.6: if $s\ \mathsf{lparen}$, then $s\ \mathsf{mparen}$.

Exercise 1.13. Given a judgment $t\ \mathsf{tree}$, we define two functions $\mathit{numLeaf}(t)$ and $\mathit{numNode}(t)$ for calculating the number of leaves and the number of nodes in t, respectively:

$$\begin{array}{rcl}
\mathit{numLeaf}(\mathtt{leaf}\ ) & = & 1 \\
\mathit{numLeaf}(\mathtt{node}\ (t_1, n, t_2)) & = & \mathit{numLeaf}(t_1) + \mathit{numLeaf}(t_2) \\
\mathit{numNode}(\mathtt{leaf}\ ) & = & 0 \\
\mathit{numNode}(\mathtt{node}\ (t_1, n, t_2)) & = & \mathit{numNode}(t_1) + \mathit{numNode}(t_2) + 1
\end{array}$$

Use rule induction to prove that if $t\ \mathsf{tree}$, then $\mathit{numLeaf}(t) - \mathit{numNode}(t) = 1$.

Exercise 1.14. Prove a lemma: if $\underbrace{((\cdots(}_{k}\ s\ \mathsf{lparen}$, then $\underbrace{((\cdots(}_{k}()s\ \mathsf{lparen}$. Use this lemma to prove Lemma 1.10. Your proof needs to exploit the equivalence between $s\ \mathsf{mparen}$ and $s\ \mathsf{lparen}$ as stated in Theorem 1.6 and Exercise 1.12.

Exercise 1.15. Proof the converse of Theorem 1.8: if s mparen, then $0 \rhd s$.

Exercise 1.16. Consider an SML implementation of the factorial function:

```
fun fact' 0 a = a
  | fact' n a = fact' (n - 1) (n * a)
fun fact n = fact' n 1
```

We wish to prove that `fact` $\widehat{n}$ evaluates to $\widehat{n!}$ by mathematical induction on $n \geq 0$, where $\widehat{n}$ stands for an SML constant expression for a mathematical integer n. Since `fact` $\widehat{n}$ reduces to `fact'` $\widehat{n}$ `1`, we try to prove a lemma that `fact'` $\widehat{n}$ `1` evaluates to $\widehat{n!}$. Unfortunately it is impossible to prove the lemma by mathematical induction on n. How would you generalize the lemma so that mathematical induction works on n?

Exercise 1.17. The principle of mathematical induction states that for any natural number n, a judgment $P(n)$ holds if the following two conditions are met:

1. $P(0)$ holds.
2. $P(k)$ implies $P(k+1)$ where $k \geq 0$.

There is another principle, called *complete induction*, which allows stronger assumptions in proving $P(k+1)$:

1. $P(0)$ holds.
2. $P(0), P(1), \cdots, P(k)$ imply $P(k+1)$ where $k \geq 0$.

It turns out that complete induction is not a new principle; rather it is a derived principle which can be justified by the principle of mathematical induction. Use mathematical induction to show that if the two conditions for complete induction are met, $P(n)$ holds for any natural number n.

Exercise 1.18. Consider the following inference rules for comparing two natural number for equality:

$$\frac{}{\mathtt{zero} \doteq \mathtt{zero}}\ \mathit{EqZero} \qquad \frac{n \doteq m}{\mathtt{succ}\ n \doteq \mathtt{succ}\ m}\ \mathit{EqSucc}$$

Show that the following inference rule is admissible:

$$\frac{n \doteq m \quad n\ \mathsf{double}\ n' \quad m\ \mathsf{double}\ m'}{n' \doteq m'}\ \mathit{EqDouble}$$