SIGPL Summer School — Supplementary Notes

Sungwoo Park

August 18 – 20, 2009
http://sigpl.or.kr/school/2009s/

Preface

This is the supplementary material to be distributed to the students at SIGPL Summer School 2009. It is based on the course notes for CSE-433 *Logic in Computer Science* taught at POSTECH, and aims to introduce the basic proof theory (natural deduction) that underlies the design of the proof assistant Coq.

Chapter 1

Propositional Logic

This chapter develops *propositional logic*, *i.e.*, logic without universal or existential quantifications.

In developing a formal system of propositional logic, we use two judgments: $A\ \mathit{prop}$ and $A\ \mathit{true}$.

$$\begin{array}{rcl} A\ \mathit{prop} & \Leftrightarrow & \textit{A is a proposition} \\ A\ \mathit{true} & \Leftrightarrow & \textit{A is true} \end{array}$$

$A\ \mathit{prop}$ becomes evident by the presence of an inference rule deducing $A\ \mathit{true}$. We will inductively define the set of propositions using binary connectives (*e.g.*, implication $\supset$, conjunction $\wedge$, disjunction $\vee$) and unary connectives (*e.g.*, negation $\neg$). The inference rules will be designed in such a way that the definition of a connective does not involve another connective. We say that the resultant system is *orthogonal* in the sense that all connectives can be developed independently of each other.

1.1 Natural deduction system for propositional logic

Natural deduction [**?**] is a principle for building a system of logic whose main concepts are *introduction* and *elimination rules*. An introduction rule explains how to deduce a truth judgment involving a particular connective, exploiting those judgments in the premise. That is, it explains how to "introduce" the connective in a derivation (when read in the top-down way). For example, an introduction rule for the conjunction connective would look like:

$$\frac{\cdots}{A \wedge B\ \mathit{true}}\ \wedge\mathsf{I}$$

A dual concept is an elimination rule which explains how to exploit a truth judgment involving a particular connective to deduce another judgment in the conclusion. That is, it explains how to "eliminate" the connective in a derivation (when read in the top-down way). For example, an elimination rule for the conjunction connective would look like:

$$\frac{A \wedge B\ \mathit{true}}{\cdots}\ \wedge\mathsf{E}$$

An introduction rule usually conveys the intuition behind a connective and is thus relatively easy to design. In contrast, an elimination rule extracts the knowledge represented by a judgment and careful design is required to ensure that the resultant system is sound and complete in some sense. For example, an ill-designed elimination rule may be so strong as to extract false knowledge that cannot be justified by its corresponding introduction rule. Or it may be too weak to deduce any interesting judgment. Note that an introduction rule takes precedence over its corresponding elimination rule because without an introduction rule, there is no use in designing an elimination rule. That is, an elimination rule cannot be considered separately from its corresponding introduction rule whereas the design of an introduction rule can be an isolated task.

Below we develop a natural deduction system for propositional logic, beginning with the conjunction connective $\wedge$ (which is the easiest case).

Conjunction

Before we investigate inference rules for $\wedge$, we need to know how to build valid propositions involving $\wedge$. Hence we need a *formation rule* to state that $A \wedge B$, read as *"A and B"* or *"A conjunction B,"* is a proposition if both A and B are propositions:

$$\frac{A\ \mathit{prop} \quad B\ \mathit{prop}}{A \wedge B\ \mathit{prop}}\ \wedge\mathsf{F}$$

In order to justify the rule $\wedge\mathsf{F}$, we need an inference rule for proving the truth of $A \wedge B$ on the assumption that there are inference rules for proving the truth of A and B. Since $A \wedge B$ is intended to be true whenever both A and B are true, we use the following introduction rule to admit $A \wedge B$ as a proposition:

$$\frac{A\ \mathit{true} \quad B\ \mathit{true}}{A \wedge B\ \mathit{true}}\ \wedge\mathsf{I}$$

The rule $\wedge\mathsf{I}$ says that if both A and B are true, then $A \wedge B$ is true. It follows the usual interpretation of an inference rule: if the premise holds, then the conclusion holds. Now we may use the rule $\wedge\mathsf{I}$ to construct a proof of $A \wedge B\ \mathit{true}$ from a proof $\mathcal{D}_A$ of $A\ \mathit{true}$ and a proof $\mathcal{D}_B$ of $B\ \mathit{true}$; we write $\begin{array}{c}\mathcal{D}_A \\ A\ \mathit{true}\end{array}$ to mean that $\mathcal{D}_A$ is a proof of $A\ \mathit{true}$, including the last inference rule whose conclusion is $A\ \mathit{true}$:

$$\frac{\begin{array}{c}\mathcal{D}_A \\ A\ \mathit{true}\end{array} \quad \begin{array}{c}\mathcal{D}_B \\ B\ \mathit{true}\end{array}}{A \wedge B\ \mathit{true}}\ \wedge\mathsf{I}$$

The design of an elimination rule for $\wedge$ begins with $A \wedge B\ \mathit{true}$ as a premise. Since $A \wedge B\ \mathit{true}$ expresses that both A and B are true, we may conclude either $A\ \mathit{true}$ or $B\ \mathit{true}$ from $A \wedge B\ \mathit{true}$, as shown in the two elimination rules for $\wedge$:

$$\frac{A \wedge B\ \mathit{true}}{A\ \mathit{true}}\ \wedge\mathsf{E_L} \qquad \frac{A \wedge B\ \mathit{true}}{B\ \mathit{true}}\ \wedge\mathsf{E_R}$$

Implication

The implication connective $\supset$ requires the notion of a *hypothetical proof* which is a proof containing *hypotheses*. We read $A \supset B$ as *"A implies B"* or *"if A, then B,"* and use the following formation rule:

$$\frac{A\ \mathit{prop} \quad B\ \mathit{prop}}{A \supset B\ \mathit{prop}}\ \supset\mathsf{F}$$

The intuition behind $\supset$ is that $A \supset B\ \mathit{true}$ holds whenever $A\ \mathit{true}$ implies $B\ \mathit{true}$, or a hypothesis of $A\ \mathit{true}$ leads to a proof of $B\ \mathit{true}$. We write a hypothesis of $A\ \mathit{true}$ as $\overline{A\ \mathit{true}}$, and obtain the following introduction rule for $\supset$:

$$\frac{\begin{array}{c}\overline{A\ \mathit{true}}^{\,x} \\ \vdots \\ B\ \mathit{true}\end{array}}{A \supset B\ \mathit{true}}\ \supset\mathsf{I}^x$$

We may directly deduce $A\ \mathit{true}$ using the hypothesis $\overline{A\ \mathit{true}}^{\,x}$ when necessary in the proof of $B\ \mathit{true}$.

The premise of the rule $\supset\mathsf{I}^x$ is an example of a hypothetical proof because it contains a hypothesis, *i.e.*, a judgment that is assumed to hold. We say that the rule $\supset\mathsf{I}$ *internalizes* the hypothetical proof in its premise as a proposition $A \supset B$ in the sense that the truth of $A \supset B$ compactly represents the knowledge expressed by the hypothetical proof.

There are three observations to make about the rule $\supset\mathsf{I}^x$. First we annotate both the hypothesis $\overline{A\ \mathit{true}}$ and the rule name $\supset\mathsf{I}$ with the same label x. Thus a label in a hypothesis indicates from which inference rule the hypothesis originates. It is not necessary to annotate all hypotheses with different labels as long as no conflict occurs between two hypotheses with the same label. For example, the following derivation is okay even though both hypotheses are annotated with the same label x:

$$\dfrac{\dfrac{\begin{array}{c}\overline{A\ \mathit{true}}^{x}\\ \vdots \\ B\ \mathit{true}\end{array}}{A \supset B\ \mathit{true}}\supset\mathsf{I}^{x} \quad \dfrac{\begin{array}{c}\overline{A'\ \mathit{true}}^{x}\\ \vdots\\ B'\ \mathit{true}\end{array}}{A'\supset B'\ \mathit{true}}\supset\mathsf{I}^{x}}{(A\supset B)\wedge(A'\supset B')\ \mathit{true}}$$

Second the hypothesis $\overline{A\ \mathit{true}}^{x}$ remains in effect only within the premise of the rule $\supset\mathsf{I}^{x}$. In other words, its scope is restricted to the premise of the rule $\supset\mathsf{I}^{x}$. After the rule $\supset\mathsf{I}^{x}$ is applied to deduce $A \supset B\ \mathit{true}$, $\overline{A\ \mathit{true}}^{x}$ may no longer be used as a valid hypothesis. For example, the proof below may not use the hypothesis $\overline{A\ \mathit{true}}^{x}$ in the proof of in the proof $\mathcal{D}_A$ of $A\ \mathit{true}$ which lies outside the scope of $\overline{A\ \mathit{true}}^{x}$:

$$\dfrac{\begin{array}{c}\mathcal{D}_A\\ A\ \mathit{true}\end{array} \quad \dfrac{\begin{array}{c}\overline{A\ \mathit{true}}^{x}\\ \vdots\\ B\ \mathit{true}\end{array}}{A\supset B\ \mathit{true}}\supset\mathsf{I}^{x}}{A\wedge(A\supset B)\ \mathit{true}}\wedge\mathsf{I}$$

We say that a hypothesis is *discharged* when its corresponding inference rule is applied and its scope is exited.

Note that while the premise of the rule $\supset\mathsf{I}^{x}$ is a hypothetical proof, the whole proof itself is *not* a hypothetical proof. Specifically the proof $\mathcal{D}$ below is a hypothetical proof, but the proof $\mathcal{E}$ is not:

$$\mathcal{E}\left\{\begin{array}{c}\mathcal{D}\left\{\begin{array}{c}\overline{A\ \mathit{true}}^{x}\\ \vdots\\ B\ \mathit{true}\end{array}\right.\\ \hline A\supset B\ \mathit{true}\end{array}\right.\supset\mathsf{I}^{x}$$

The reason why $\mathcal{E}$ is not a hypothetical proof is that the hypothesis $\overline{A\ \mathit{true}}^{x}$ is discharged when the rule $\supset\mathsf{I}^{x}$ is applied, and thus is not visible to the outside. That is, we are free to use any hypothesis without turning the whole proof into a hypothetical proof as long as it is eventually discharged.

Third the hypothesis $\overline{A\ \mathit{true}}^{x}$ may be used not just once but as many times as necessary. In fact, we may even ignore it in the proof without using it at all. Here are examples of proofs that ignore $\overline{A\ \mathit{true}}^{x}$, use it once, and use it twice:

$$\dfrac{\dfrac{\overline{B\ \mathit{true}}^{y} \quad \begin{array}{c}\overline{A\ \mathit{true}}^{x}\\ \textit{(not used in the proof)}\end{array}}{A\supset B\ \mathit{true}}\supset\mathsf{I}^{x}}{B\supset(A\supset B)\ \mathit{true}}\supset\mathsf{I}^{y} \qquad \dfrac{\overline{A\ \mathit{true}}^{x}}{A\supset A\ \mathit{true}}\supset\mathsf{I}^{x} \qquad \dfrac{\dfrac{\overline{A\ \mathit{true}}^{x} \quad \overline{A\ \mathit{true}}^{x}}{A\wedge A\ \mathit{true}}\wedge\mathsf{I}}{A\supset(A\wedge A)\ \mathit{true}}\supset\mathsf{I}^{x}$$

As with the elimination rules for $\wedge$, the design of the elimination rule for $\supset$ begins with a premise $A\supset B\ \mathit{true}$. Since $A\supset B\ \mathit{true}$ expresses that $A\ \mathit{true}$ implies $B\ \mathit{true}$, the only way to exploit it is by supplying a proof of $A\ \mathit{true}$ to conclude $B\ \mathit{true}$. Hence the elimination rule for $\supset$ uses both $A\supset B\ \mathit{true}$ and $A\ \mathit{true}$ as its premises:

$$\dfrac{A\supset B\ \mathit{true} \quad A\ \mathit{true}}{B\ \mathit{true}}\supset\mathsf{E}$$

The following example proves $(A\supset B)\supset(A\supset B)\ \mathit{true}$ using the rule $\supset\mathsf{E}$:

$$\dfrac{\dfrac{\dfrac{\overline{A\supset B\ \mathit{true}}^{x} \quad \overline{A\ \mathit{true}}^{y}}{B\ \mathit{true}}\supset\mathsf{E}}{A\supset B\ \mathit{true}}\supset\mathsf{I}^{y}}{(A\supset B)\supset(A\supset B)\ \mathit{true}}\supset\mathsf{I}^{x}$$

(We can also prove $(A \supset B) \supset (A \supset B)$ *true* by directly using the hypothesis $\overline{A \supset B\ \mathit{true}}^{x}$.)

Here are two examples involving both $\wedge$ and $\supset$. The two proofs show that $A \supset (B \supset C)$ and $(A \wedge B) \supset C$ are logically equivalent because each one implies the other. (See Section 1.2 for further details.)

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{A \supset (B \supset C)\ \mathit{true}}^{x} \quad \dfrac{\overline{A \wedge B\ \mathit{true}}^{y}}{A\ \mathit{true}}\wedge\mathsf{E_L}}{B \supset C\ \mathit{true}}\supset\mathsf{E} \quad \dfrac{\overline{A \wedge B\ \mathit{true}}^{y}}{B\ \mathit{true}}\wedge\mathsf{E_R}}{C\ \mathit{true}}\supset\mathsf{E}}{(A \wedge B) \supset C\ \mathit{true}}\supset\mathsf{I}^{y}}{(A \supset (B \supset C)) \supset ((A \wedge B) \supset C)\ \mathit{true}}\supset\mathsf{I}^{x} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\overline{(A \wedge B) \supset C\ \mathit{true}}^{x} \quad \dfrac{\overline{A\ \mathit{true}}^{y} \quad \overline{B\ \mathit{true}}^{z}}{A \wedge B\ \mathit{true}}\wedge\mathsf{I}}{C\ \mathit{true}}\supset\mathsf{E}}{B \supset C\ \mathit{true}}\supset\mathsf{I}^{z}}{A \supset (B \supset C)\ \mathit{true}}\supset\mathsf{I}^{y}}{((A \wedge B) \supset C) \supset (A \supset (B \supset C))\ \mathit{true}}\supset\mathsf{I}^{x}$$

Disjunction

Like $\wedge$ and $\supset$, the disjunction connective $\vee$ is binary:

$$\frac{A\ \mathit{prop} \quad B\ \mathit{prop}}{A \vee B\ \mathit{prop}}\vee\mathsf{F}$$

$A \vee B$, read as *"A or B"* or *"A disjunction B,"* is intended to be true when either A or B is true, but we do not necessarily know which alternative is true. In our formulation of propositional logic, an introduction rule for $\vee$ concludes $A \vee B$ *true* from a proof of either A *true* or B *true*:

$$\frac{A\ \mathit{true}}{A \vee B\ \mathit{true}}\vee\mathsf{I_L} \qquad \frac{B\ \mathit{true}}{A \vee B\ \mathit{true}}\vee\mathsf{I_R}$$

The design of an elimination rule for $\vee$ is not obvious. A naive attempt would be to conclude one of A *true* and B *true* from $A \vee B$ *true*:

$$\frac{A \vee B\ \mathit{true}}{A\ \mathit{true}}\vee\mathsf{E_L}? \qquad \frac{A \vee B\ \mathit{true}}{B\ \mathit{true}}\vee\mathsf{E_R}?$$

In a certain sense, both rules are too strong (or too powerful) because they conclude a judgment that cannot be justified by $A \vee B$ *true*, which does not specify exactly which of A *true* and B *true* holds. In fact, each rule allows us to prove A *true* for any proposition A:

$$\dfrac{\dfrac{\dfrac{\overline{B\ \mathit{true}}^{x}}{B \supset B\ \mathit{true}}\supset\mathsf{I}^{x}}{A \vee (B \supset B)\ \mathit{true}}\vee\mathsf{I_R}}{A\ \mathit{true}}\vee\mathsf{E_L}?$$

Since it is generally unknown which of A *true* and B *true* has been supplied in a proof of $A \vee B$ *true* (*e.g.*, when $A \vee B$ *true* is a hypothesis), the only logical way to exploit $A \vee B$ *true* is by considering both possibilities simultaneously. If we can prove C *true* both from A *true* and from B *true* for a certain proposition C, then we may conclude C *true* from $A \vee B$ *true*, since C *true* holds regardless of how the proof of $A \vee B$ *true* has been built. The elimination rule for $\vee$ expresses such a way of reasoning:

$$\frac{A \vee B\ \mathit{true} \quad \begin{matrix}\overline{A\ \mathit{true}}^{x}\\ \vdots \\ C\ \mathit{true}\end{matrix} \quad \begin{matrix}\overline{B\ \mathit{true}}^{y}\\ \vdots \\ C\ \mathit{true}\end{matrix}}{C\ \mathit{true}}\vee\mathsf{E}^{x,y}$$

Note that A *true* and B *true* are introduced as new hypotheses and are annotated with different labels x and y. As in the elimination rule for $\supset$, their scope is limited to their respective premises of the rule $\vee\mathsf{E}^{x,y}$ (*i.e.*, $\overline{A\ \mathit{true}}^{x}$ to the second premise and $\overline{B\ \mathit{true}}^{y}$ to the third premise), which means that both hypotheses are discharged when C *true* is deduced in the conclusion.

Unlike the elimination rules for $\wedge$ and $\supset$, the elimination rule for $\vee$ exploits $A \vee B$ *true* in an indirect way in that its conclusion contains a proposition C that is not necessarily A, B, or their combination.

That is, when applying the elimination rule to $A \lor B\ \mathit{true}$, we ourselves have to choose a proposition C (which can be completely unrelated to A and B) such that $C\ \mathit{true}$ is provable both from $A\ \mathit{true}$ and from $B\ \mathit{true}$. For this reason, the inclusion of $\lor$ in a system of logic makes it hard to investigate metalogical properties of the system, as we will see later.

As a trivial example, let us prove that $A\ \mathit{true}$ is stronger than $A \lor B\ \mathit{true}$:

$$\dfrac{\dfrac{\dfrac{}{A\ \mathit{true}}\,x}{A \lor B\ \mathit{true}}\ \lor\mathsf{I}_\mathsf{L}}{A \supset (A \lor B)\ \mathit{true}}\ \supset\mathsf{I}^x$$

The converse does not hold, *i.e.*, $A \lor B\ \mathit{true}$ is strictly weaker than $A\ \mathit{true}$, because there is no way to prove $A\ \mathit{true}$ from $B\ \mathit{true}$ for arbitrary propositions A and B:

$$\dfrac{\dfrac{\dfrac{}{A \lor B\ \mathit{true}}\,x \quad \dfrac{}{A\ \mathit{true}}\,y \quad \begin{array}{c}\dfrac{}{B\ \mathit{true}}\,z \\ \vdots \\ A\ \mathit{true}\quad \textit{(impossible)}\end{array}}{A\ \mathit{true}}\ \lor\mathsf{E}^{y,z}}{(A \lor B) \supset A\ \mathit{true}}\ \supset\mathsf{I}^x$$

As another example, let us prove that the disjunction connective is commutative:

$$(A \lor B) \supset (B \lor A)\ \mathit{true}$$

We begin by applying the rule $\supset\mathsf{I}$ so that the problem reduces to proving $B \lor A\ \mathit{true}$ from $A \lor B\ \mathit{true}$:

$$\dfrac{\begin{array}{c}\dfrac{}{A \lor B\ \mathit{true}}\,x \\ \vdots \\ B \lor A\ \mathit{true}\end{array}}{(A \lor B) \supset (B \lor A)\ \mathit{true}}\ \supset\mathsf{I}^x$$

At this point, the proof may proceed either in a bottom-up way by applying an introduction rule $\lor\mathsf{I}_\mathsf{L}$ or $\lor\mathsf{I}_\mathsf{R}$ to $B \lor A\ \mathit{true}$, or in a top-down way by applying the elimination rule $\lor\mathsf{E}$ to $A \lor B\ \mathit{true}$. In the first case, we eventually get stuck because it is impossible to prove $A\ \mathit{true}$ or $B\ \mathit{true}$ from $A \lor B\ \mathit{true}$. For example, we cannot fill the gap in the proof shown below:

$$\dfrac{\dfrac{\begin{array}{c}\dfrac{}{A \lor B\ \mathit{true}}\,x \\ \vdots \\ B\ \mathit{true}\end{array}}{B \lor A\ \mathit{true}}\ \lor\mathsf{I}_\mathsf{L}}{(A \lor B) \supset (B \lor A)\ \mathit{true}}\ \supset\mathsf{I}^x$$

In the second case, the problem reduces to separately proving $B \lor A\ \mathit{true}$ from $A\ \mathit{true}$ and from $B\ \mathit{true}$, which is accomplished by applying the introduction rules for $\lor$:

$$\dfrac{\dfrac{\dfrac{}{A \lor B\ \mathit{true}}\,x \quad \dfrac{\dfrac{}{A\ \mathit{true}}\,y}{B \lor A\ \mathit{true}}\ \lor\mathsf{I}_\mathsf{R} \quad \dfrac{\dfrac{}{B\ \mathit{true}}\,z}{B \lor A\ \mathit{true}}\ \lor\mathsf{I}_\mathsf{L}}{B \lor A\ \mathit{true}}\ \lor\mathsf{E}^{y,z}}{(A \lor B) \supset (B \lor A)\ \mathit{true}}\ \supset\mathsf{I}^x$$

Truth and falsehood

Truth $\top$ is a proposition that is assumed to be always true. Hence a proof of $\top\ \mathit{true}$ requires no particular evidence and is always provable, as indicated by the empty premise in its introduction rule:

$$\dfrac{}{\top\ \mathit{prop}}\ \top\mathsf{F} \qquad \dfrac{}{\top\ \mathit{true}}\ \top\mathsf{I}$$

Then how do we exploit a proof of $\top$ *true* in an elimination rule? Since we have to provide no particular evidence in a proof of $\top$ *true*, there is no logical content in it, which implies that there is no interesting way to exploit it. Therefore $\top$ has no elimination rule.

Falsehood $\bot$ is a proposition that is never true, or equivalently, whose truth is impossible to establish. The intuition is that it denotes a logical contradiction which must not be provable under any circumstance. Therefore there is no introduction rule for $\bot$. Interestingly, however, there *is* an elimination rule for $\bot$. Suppose that we have a proof of $\bot$ *true*. If we think of $\bot$ *true* as something impossible to prove, or as something that is the most difficult to prove, the existence of its proof implies that we can prove everything (which is no more difficult to prove than $\bot$ *true*)! Therefore the elimination rule for $\bot$ deduces C *true* for an arbitrary proposition C:

$$\frac{}{\bot\ \mathit{prop}}\ \bot\mathsf{F} \qquad \frac{\bot\ \mathit{true}}{C\ \mathit{true}}\ \bot\mathsf{E}$$

Then why do we need an elimination rule for $\bot$ at all, if it is impossible to prove $\bot$ *true*? While it is impossible to prove $\bot$ *true* out of nothing, it is possible to prove $\bot$ *true* using hypotheses. For example, $\bot$ *true* in the premise of the rule $\bot$E itself may be a hypothesis, as illustrated in the proof below:

$$\dfrac{\dfrac{\dfrac{}{\bot\ \mathit{true}}\,x}{C\ \mathit{true}}\ \bot\mathsf{E}}{\bot \supset C\ \mathit{true}}\ {\supset}\mathsf{I}^x$$

In essence, there is nothing wrong with reasoning from an assumption that something impossible to prove has been proven somehow.

We say that a system of logic is *inconsistent* if $\bot$ *true* is provable in it, and *consistent* if not. An inconsistent system is worthless because a judgment A *true* is provable for an arbitrary proposition A. We will later present a proof that our system of propositional logic is consistent, whose discovery was in fact a major milestone in the history of logic.

Truth $\top$ and falsehood $\bot$ can also be viewed as the nullary cases of conjunction and disjunction, respectively. Consider a general n-ary case $\bigwedge_{i=1}^n A_i$ of conjunction with a single introduction rule and n elimination rules:

$$\frac{A_i\ \mathit{true}\quad \mathit{for}\ i = 1, \cdots, n}{\bigwedge_{i=1}^n A_i\ \mathit{true}}\ \textstyle\bigwedge\mathsf{I} \qquad \frac{\bigwedge_{i=1}^n A_i\ \mathit{true}}{A_i\ \mathit{true}}\ \textstyle\bigwedge\mathsf{E}_i\ (1 \le i \le n)$$

If we let $\top = \bigwedge_{i=1}^n A_i$ with $n = 0$, the rule $\bigwedge$I turns into the rule $\top$I because it comes to have an empty premise, and each rule $\bigwedge \mathsf{E}_i$ disappears (*i.e.*, no elimination rule for $\top$). Similarly a general n-ary case $\bigvee_{i=1}^n A_i$ of disjunction has n introduction rules and a single elimination rule:

$$\frac{A_i\ \mathit{true}}{\bigvee_{i=1}^n A_i\ \mathit{true}}\ \textstyle\bigvee\mathsf{I}_i (1 \le i \le n) \qquad \dfrac{\bigvee_{i=1}^n A_i\ \mathit{true} \qquad \begin{array}{c}\dfrac{}{A_i\ \mathit{true}}\,x_i \\ \vdots \\ C\ \mathit{true}\end{array}\quad \mathit{for}\ i = 1, \cdots, n}{C\ \mathit{true}}\ \textstyle\bigvee\mathsf{E}^x$$

If we let $\bot = \bigvee_{i=1}^n A_i$ with $n = 0$, each rule $\bigvee \mathsf{I}_i$ disappears (*i.e.*, no introduction rule for $\bot$), and the rule $\bigvee$E turns into the rule $\bot$E because all hypothetical proofs in its premise disappear.

Now it is clear that $\top$ and $\bot$ are identities for the binary connectives $\wedge$ and $\vee$, respectively. For example, we can identify $A \wedge \top$ with A: if A *true* is provable, then $A \wedge \top$ *true* is also provable because $\top$ *true* automatically holds; the converse follows by the rule $\wedge\mathsf{E_L}$. Similarly we can identify $A \vee \bot$ with A: if $A \vee \bot$ *true* is provable, A *true* must also be provable because the second alternative $\bot$ *true* cannot be taken; the converse follows by the rule $\vee\mathsf{I_L}$.

Negation

The only unary connective in propositional logic is negation $\neg$:

$$\frac{A\ \mathit{prop}}{\neg A\ \mathit{prop}}\ \neg\mathsf{F}$$

$\neg A$, read as *"not A"* or *"negation A,"* denotes the logical negation of A, and its truth means that A cannot be true. We use an approach that uses a *notational definition* by regarding $\neg A$ as a syntactic abbreviation of $A \supset \bot$. That is, $\neg$ plays no semantic role at all and $\neg A$ is simply expanded to $A \supset \bot$. The notational definition of $\neg$ justifies the following rules:

$$\dfrac{\begin{array}{c}\overline{A\ \mathit{true}}^{\,x}\\ \vdots\\ \bot\ \mathit{true}\end{array}}{\neg A\ \mathit{true}}\,\neg\mathsf{I}^{x} \qquad \dfrac{\neg A\ \mathit{true} \quad A\ \mathit{true}}{\bot\ \mathit{true}}\,\neg\mathsf{E}$$

Note that if $\neg$ was defined as an independent connective rather than a notational convenience, these rules would destroy the orthogonality of the system because the meaning of $\neg$ would depend on the meaning of $\bot$. We use the third approach in our treatment of $\neg$ (which is the most popular definition in the literature).

As an example, we prove that if A is true, then $\neg A$ cannot be true:

$$\dfrac{\dfrac{\dfrac{\overline{\neg A\ \mathit{true}}^{\,y} \quad \overline{A\ \mathit{true}}^{\,x}}{\bot\ \mathit{true}}\,\neg\mathsf{E}}{\neg\neg A\ \mathit{true}}\,\neg\mathsf{I}^{y}}{A \supset \neg\neg A\ \mathit{true}}\,\supset\!\mathsf{I}^{x}$$

The converse $\neg\neg A \supset A\ \mathit{true}$ is *not* provable, however, which implies that $A\ \mathit{true}$ is strictly stronger than $\neg\neg A\ \mathit{true}$. That is, a proof that $\neg A$ cannot be true is not enough for concluding that A is true. A failed attempt to prove $\neg\neg A \supset A\ \mathit{true}$ would look like:

$$\dfrac{\dfrac{\dfrac{\overline{\neg\neg A\ \mathit{true}}^{\,x} \qquad \begin{array}{c}\overline{\neg\neg A\ \mathit{true}}^{\,x}\\ \vdots\ ?\\ \neg A\ \mathit{true}\end{array}}{\bot\ \mathit{true}}\,\neg\mathsf{E}}{A\ \mathit{true}}\,\bot\mathsf{E}}{\neg\neg A \supset A\ \mathit{true}}\,\supset\!\mathsf{I}^{x}$$

The unprovability of $\neg\neg A \supset A\ \mathit{true}$ is a quintessential feature of the system of logic presented so far, or any system belonging to what is known as *constructive logic* or *intuitionistic logic*. In constructive logic, what $\neg A\ \mathit{true}$ proves is not exactly the direct opposite of what $A\ \mathit{true}$ proves. Rather it provides only indirect evidence that there is no proof of $A\ \mathit{true}$ by showing that the existence of such a proof leads to a logical contradiction. In contrast, *classical logic* assumes that every proposition is either true or false and has no intermediate state. Under classical logic, $\neg\neg A\ \mathit{true}$ is indistinguishable from $A\ \mathit{true}$ because A is either true or false and we have positive evidence that A cannot be false. The truth table method for proving the truth of a proposition is based on classical logic, which tries all possible combinations of truth and falsehood values for all atomic propositions.

Figure 1.1 shows all inference rules of propositional logic where the set of propositions is inductively defined as follows:

$$\text{proposition} \quad A \quad ::= \quad P \mid A \wedge A \mid A \supset A \mid A \vee A \mid \top \mid \bot \mid \neg A$$

P is called a *propositional constant* and denotes an atomic proposition (*e.g. '1 + 1 is equal to 0,' '1 + 1 is equal to 2' is true,' 'the moon is made of cheese,' etc*). The rules $\neg\mathsf{I}$ and $\neg\mathsf{E}$ are derived rules under the notational definition $\neg A = A \supset \bot$. From now on, we use the following operator precedence

$$\neg > \wedge > \vee > \supset$$

where $\wedge$, $\vee$, $\supset$ are all right-associative. Examples are:

$$\begin{array}{rcl}\neg A \wedge B & = & (\neg A) \wedge B\\ A \wedge B \vee C & = & (A \wedge B) \vee C\\ A \vee B \supset C & = & (A \vee B) \supset C\\ \neg A \wedge B \vee C \supset D & = & (((\neg A) \wedge B) \vee C) \supset D\end{array} \qquad \begin{array}{rcl}A \wedge B \wedge C & = & A \wedge (B \wedge C)\\ A \vee B \vee C & = & A \vee (B \vee C)\\ A \supset B \supset C & = & A \supset (B \supset C)\end{array}$$

$$\frac{A\ \mathit{true} \quad B\ \mathit{true}}{A \wedge B\ \mathit{true}}\wedge\mathsf{I} \qquad \frac{A \wedge B\ \mathit{true}}{A\ \mathit{true}}\wedge\mathsf{E_L} \qquad \frac{A \wedge B\ \mathit{true}}{B\ \mathit{true}}\wedge\mathsf{E_R}$$

$$\frac{\begin{array}{c}\overline{A\ \mathit{true}}^{\,x}\\ \vdots \\ B\ \mathit{true}\end{array}}{A \supset B\ \mathit{true}}\supset\mathsf{I}^{x} \qquad \frac{A \supset B\ \mathit{true} \quad A\ \mathit{true}}{B\ \mathit{true}}\supset\mathsf{E}$$

$$\frac{A\ \mathit{true}}{A \vee B\ \mathit{true}}\vee\mathsf{I_L} \qquad \frac{B\ \mathit{true}}{A \vee B\ \mathit{true}}\vee\mathsf{I_R} \qquad \frac{A \vee B\ \mathit{true} \quad \begin{array}{c}\overline{A\ \mathit{true}}^{\,x}\\ \vdots \\ C\ \mathit{true}\end{array} \quad \begin{array}{c}\overline{B\ \mathit{true}}^{\,y}\\ \vdots \\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\vee\mathsf{E}^{x,y}$$

$$\frac{}{\top\ \mathit{true}}\top\mathsf{I} \qquad \frac{\bot\ \mathit{true}}{C\ \mathit{true}}\bot\mathsf{E} \qquad \frac{\begin{array}{c}\overline{A\ \mathit{true}}^{\,x}\\ \vdots \\ \bot\ \mathit{true}\end{array}}{\neg A\ \mathit{true}}\neg\mathsf{I}^{x} \qquad \frac{\neg A\ \mathit{true} \quad A\ \mathit{true}}{\bot\ \mathit{true}}\neg\mathsf{E}$$

Figure 1.1: Natural deduction system for propositional logic

1.2 Logical equivalence

We say that a proposition A is logically equivalent to another proposition B, written $A \equiv B$, if $A\ \mathit{true}$ implies $B\ \mathit{true}$ and vice versa. A notational definition of logical equivalence $A \equiv B$ is given as follows:

$$A \equiv B \quad = \quad (A \supset B) \wedge (B \supset A)\ \mathit{true}$$

If A and B are logically equivalent, an occurrence of A inside any proposition may be replaced by B (or an occurrence of B by A) without changing its meaning in that the resultant proposition remains logically equivalent to the original proposition. Thus logical equivalences enable us to simplify a proof involving a proposition that is logically equivalent to a less complex proposition. For example,

$$\neg\neg\neg A \supset (\neg\neg\neg B \supset \neg(A \vee B))\ \mathit{true}$$

becomes easy (or even obvious) to prove once we transform $\neg\neg\neg A \supset (\neg\neg\neg B \supset \neg(A \vee B))$ into $(\neg A \wedge \neg B) \supset \neg(A \vee B)$ by exploiting logical equivalences $\neg\neg\neg A \equiv \neg A$ and $A \supset (B \supset C) \equiv (A \wedge B) \supset C$.

Below we list logical equivalences of propositional logic which are divided into three groups.

Commutativity and idempotence. $\wedge$ and $\vee$ are commutative and idempotent. An implication $A \supset A$ is logically meaningless and reduces to $\top$.

(C1)	$A \wedge B \equiv B \wedge A$
(C2)	$A \vee B \equiv B \vee A$
(C3)	$A \supset B \not\equiv B \supset A$
(I1)	$A \wedge A \equiv A$
(I2)	$A \vee A \equiv A$
(I3)	$A \supset A \equiv \top$

Truth and falsehood. Each logical equivalence below deals with a proposition of the form $\top\ \phi\ A$, $\bot\ \phi\ A$, $A \supset \top$, or $A \supset \bot$ where ϕ is $\wedge$, $\vee$, or $\supset$.

(M1)	$\top \wedge A \equiv A$
(M2)	$\top \vee A \equiv \top$
(M3)	$\top \supset A \equiv A$
(M4)	$\bot \wedge A \equiv \bot$
(M5)	$\bot \vee A \equiv A$
(M6)	$\bot \supset A \equiv \top$
(M7)	$A \supset \top \equiv \top$
(M8)	$A \supset \bot = \neg A$

Interaction between connectives. Each logical equivalence below deals with a proposition of the form $A\ \phi\ (B\ \phi\ C)$ or $(A\ \phi\ B) \supset C$ where ϕ is $\wedge$, $\vee$, or $\supset$.

(L1)	$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$	*(associativity of $\wedge$)*
(L2)	$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$	*(distributivity of $\wedge$ over $\vee$)*
(L3)	$A \wedge (B \supset C) \equiv ?$	*(no interaction)*
(L4)	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$	*(distributivity of $\vee$ over $\wedge$)*
(L5)	$A \vee (B \vee C) \equiv (A \vee B) \vee C$	*(associativity of $\vee$)*
(L6)	$A \vee (B \supset C) \equiv ?$	*(no interaction)*
(L7)	$A \supset (B \wedge C) \equiv (A \supset B) \wedge (A \supset C)$	*(distributivity of $\supset$ over $\wedge$)*
(L8)	$A \supset (B \vee C) \equiv ?$	*(no interaction)*
(L9)	$A \supset (B \supset C) \equiv (A \wedge B) \supset C$	
(L10)	$(A \wedge B) \supset C \equiv A \supset (B \supset C)$	
(L11)	$(A \vee B) \supset C \equiv (A \supset C) \wedge (B \supset C)$	
(L12)	$(A \supset B) \supset C \equiv ?$	*(no interaction)*

Chapter 2

Proof Terms

This chapter presents an alternative formulation of propositional logic using the principle called the *Curry-Howard isomorphism* [**?**]. As a principle connecting logic and programming languages, it states that propositions in logic correspond to types in programming languages (*propositions-as-types* correspondence) and that proofs in logic correspond to programs in programming languages (*proofs-as-programs* correspondence). Thus, by applying the Curry-Howard isomorphism to a formulation of logic, we systematically derive a formulation of a corresponding programming language. In the case of propositional logic, we obtain a basic definition of the simply-typed λ-calculus.

The basic idea behind the Curry-Howard isomorphism is to represent a proof $\mathcal{D}$ of a truth judgment $A\ \mathit{true}$ as a *proof term* M of *type* A:

$$\overset{\mathcal{D}}{A\ \mathit{true}} \quad \Longleftrightarrow \quad M : A$$

That is, a *typing judgment* $M : A$ expresses that a proof term M of type A is a (concise) representation of a proof of $A\ \mathit{true}$. When $M : A$ holds, we say that proof term M typechecks with type A. Note that A can be interpreted both as a proposition and as a type, depending on the context in which it is used.

Under the correspondence between proofs and proof terms shown above, each inference rule for deducing truth judgments is translated to a corresponding *typing rule* for deducing typing judgments; by convention, a typing rule is given the same name as the inference rule from which it is derived:

$$\frac{\cdots}{A\ \mathit{true}}\ R \quad \Longleftrightarrow \quad \frac{\cdots}{M : A}\ R$$

Thus the typing rules for proof terms constitute another natural deduction system, in which an introduction rule assigns to a proof term a type involving a particular connective whereas an elimination rule uses such a proof term in its premise.

We may choose any syntax for proof terms as long as each proof term of type A provides all necessary information to extract a corresponding proof of $A\ \mathit{true}$. Below we design proof terms according to the syntax for the simply-typed λ-calculus so as to emphasize the close connection between logic and type theory. We use metavariables $M, N, \cdots$ for terms. Figure 2.1 shows all the typing rules for proof terms in propositional logic where the set of proof terms is inductively defined as follows:

$$\begin{array}{lll} \text{proof term} & M & ::= & x \mid (M, M) \mid \mathsf{fst}\ M \mid \mathsf{snd}\ M \mid \lambda x{:}A.\, M \mid M\ M \mid \\ & & & \mathsf{inl}_A\ M \mid \mathsf{inr}_A\ M \mid \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}\ x.\, M \mid \mathsf{inr}\ x.\, M \mid () \mid \mathsf{abort}_A\ M \end{array}$$

Conjunction

Consider an application of the rule $\wedge\mathsf{I}$ in which a proof $\mathcal{D}$ of $A \wedge B\ \mathit{true}$ is constructed from a proof $\mathcal{D}_A$ of $A\ \mathit{true}$ and a proof $\mathcal{D}_B$ of $B\ \mathit{true}$. If proof terms M and N represent $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively, we use a *product term* (M, N) of type $A \wedge B$ to represent $\mathcal{D}$. Thus the rule $\wedge\mathsf{I}$ is translated to the following typing rule (of the same name):

$$\frac{A\ \mathit{true} \quad B\ \mathit{true}}{A \wedge B\ \mathit{true}}\ \wedge\mathsf{I} \quad \Longleftrightarrow \quad \frac{M : A \quad N : B}{(M, N) : A \wedge B}\ \wedge\mathsf{I}$$

We use *projection terms* fst M and snd M in translating the rule $\wedge\mathsf{E_L}$ and $\wedge\mathsf{E_R}$; fst and snd stand for 'first projection' and 'second projection,' respectively:

$$\frac{A \wedge B\ \mathit{true}}{A\ \mathit{true}}\wedge\mathsf{E_L} \quad\iff\quad \frac{M : A \wedge B}{\mathsf{fst}\ M : A}\wedge\mathsf{E_L} \qquad\qquad \frac{A \wedge B\ \mathit{true}}{B\ \mathit{true}}\wedge\mathsf{E_R} \quad\iff\quad \frac{M : A \wedge B}{\mathsf{snd}\ M : B}\wedge\mathsf{E_R}$$

Implication

Suppose that we wish to convert to a proof term a proof $\mathcal{D}$ of $A \supset B\ \mathit{true}$ that applies the rule $\supset\mathsf{I}$ to a hypothetical proof $\mathcal{E}$ of $B\ \mathit{true}$:

$$\mathcal{D}\left\{\begin{array}{c}\dfrac{\mathcal{E}\left\{\begin{array}{c}\overline{A\ \mathit{true}}^{\,x}\\ \vdots\\ B\ \mathit{true}\end{array}\right.}{A \supset B\ \mathit{true}}\supset\mathsf{I}^x\end{array}\right.$$

In order to build a proof term M representing $\mathcal{E}$, we first need to assign a proof term to the hypothesis $\overline{A\ \mathit{true}}^{\,x}$. Since $A\ \mathit{true}$ is just a hypothesis without a concrete proof, its corresponding proof term is also unknown. Hence we represent $\overline{A\ \mathit{true}}^{\,x}$ as a *variable* x, for which we can later substitute another proof term (like we substitute a concrete proof of $A\ \mathit{true}$ for the hypothesis $\overline{A\ \mathit{true}}^{\,x}$):

$$\overline{A\ \mathit{true}}^{\,x} \quad\iff\quad \overline{x : A}$$

If M represents $\mathcal{E}$, we use a *λ-abstraction* $\lambda x\!:\!A.\,M$ to represent $\mathcal{D}$:

$$\dfrac{\begin{array}{c}\overline{A\ \mathit{true}}^{\,x}\\ \vdots\\ B\ \mathit{true}\end{array}}{A \supset B\ \mathit{true}}\supset\mathsf{I}^x \quad\iff\quad \dfrac{\begin{array}{c}\overline{x : A}\\ \vdots\\ M : B\end{array}}{\lambda x\!:\!A.\,M : A \supset B}\supset\mathsf{I}$$

We say that variable x is bound in the λ-abstraction $\lambda x\!:\!A.\,M$. Note that we may rename x to another variable without changing the meaning of $\lambda x\!:\!A.\,M$. For example, both $\lambda x\!:\!A.\,(x,x)$ and $\lambda y\!:\!A.\,(y,y)$ represent the same proof, since using a different label for the same hypothesis does not alter the structure of the proof. (Renaming a bound variable in a λ-abstraction is commonly called *α-conversion*.)

Similarly to the rule $\supset\mathsf{I}$ in propositional logic, the typing rule $\supset\mathsf{I}$ restricts the scope of the hypothesis $\overline{x : A}$ to its premise. As a result, the hypothesis $\overline{x : A}$ is discharged when the rule $\supset\mathsf{I}$ is applied, and variable x in $\lambda x\!:\!A.\,M$ can be assigned type A only if it appears within M. For example, $\lambda x\!:\!A.\,x$ has type $A \supset A$, but $(\lambda x\!:\!A.\,x, x)$ cannot be assigned a type and fails to typecheck. Also the hypothesis $\overline{x : A}$ may be used not just once but as many times as necessary. Hence proof term M in $\lambda x\!:\!A.\,M$ may contain any number of occurrences of variable x, as illustrated below:

$$\dfrac{\dfrac{\overline{x : B} \quad \begin{array}{c}\overline{y : A}\\ \textit{(not used in the proof)}\end{array}}{\lambda y\!:\!A.\,x : A \supset B}\supset\mathsf{I}}{\lambda x\!:\!B.\,\lambda y\!:\!A.\,x : B \supset (A \supset B)}\supset\mathsf{I} \qquad \dfrac{\overline{x : A}}{\lambda x\!:\!A.\,x : A \supset A}\supset\mathsf{I} \qquad \dfrac{\dfrac{\overline{x : A} \quad \overline{x : A}}{(x,x) : A \wedge A}\wedge\mathsf{I}}{\lambda x\!:\!A.\,(x,x) : A \supset (A \wedge A)}\supset\mathsf{I}$$

(See Page 3 for proofs of corresponding truth judgments.)

As a proof term corresponding to the rule $\supset\mathsf{E}$, we use a *λ-application* $M\ N$:

$$\frac{A \supset B\ \mathit{true} \quad A\ \mathit{true}}{B\ \mathit{true}}\supset\mathsf{E} \quad\iff\quad \frac{M : A \supset B \quad N : A}{M\ N : B}\supset\mathsf{E}$$

The following example uses the rule $\supset\mathsf{E}$ to typecheck $\lambda x\!:\!A \supset B.\,\lambda y\!:\!A.\,x\ y$:

$$\dfrac{\dfrac{\dfrac{\overline{x : A \supset B} \quad \overline{y : A}}{x\ y : B}\supset\mathsf{E}}{\lambda y\!:\!A.\,x\ y : A \supset B}\supset\mathsf{I}}{\lambda x\!:\!A \supset B.\,\lambda y\!:\!A.\,x\ y : (A \supset B) \supset (A \supset B)}\supset\mathsf{I}$$

Disjunction

As proof terms corresponding to the rule $\vee\mathsf{I_L}$ and $\vee\mathsf{I_R}$, we use *injection terms* $\mathsf{inl}_A\ M$ and $\mathsf{inr}_A\ M$; inl and inr stand for 'injection left' and 'injection right,' respectively:

$$\frac{A\ \mathit{true}}{A \vee B\ \mathit{true}}\ \vee\mathsf{I_L} \quad\Longleftrightarrow\quad \frac{M : A}{\mathsf{inl}_B\ M : A \vee B}\ \vee\mathsf{I_L} \qquad \frac{B\ \mathit{true}}{A \vee B\ \mathit{true}}\ \vee\mathsf{I_R} \quad\Longleftrightarrow\quad \frac{M : B}{\mathsf{inr}_A\ M : A \vee B}\ \vee\mathsf{I_R}$$

We annotate an injection term $\mathsf{inl}_A\ M$ or $\mathsf{inr}_A\ M$ with a type A so that whenever M typechecks, the whole injection term also typechecks with a unique type.

For the elimination rule $\vee\mathsf{E}$, we use a *case term* $\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}\ x.\, N \mid \mathsf{inr}\ y.\, N'$; as with the rule $\supset\mathsf{I}$, we represent hypotheses $\overline{A\ \mathit{true}}^{\,x}$ and $\overline{B\ \mathit{true}}^{\,y}$ in the premise as variables x and y:

$$\frac{A \vee B\ \mathit{true} \qquad \begin{array}{c}\overline{A\ \mathit{true}}^{\,x}\\ \vdots\\ C\ \mathit{true}\end{array} \qquad \begin{array}{c}\overline{B\ \mathit{true}}^{\,y}\\ \vdots\\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\ \vee\mathsf{E}^{x,y} \quad\Longleftrightarrow\quad \frac{M : A \vee B \qquad \begin{array}{c}\overline{x : A}\\ \vdots\\ N : C\end{array} \qquad \begin{array}{c}\overline{y : B}\\ \vdots\\ N' : C\end{array}}{\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}\ x.\, N \mid \mathsf{inr}\ y.\, N' : C}\ \vee\mathsf{E}$$

Variables x and y are bound in the case term $\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}\ x.\, N \mid \mathsf{inr}\ y.\, N'$, and remain valid only within N and N', respectively. As an example, here is a proof term of type $(A \vee B) \supset (B \vee A)$:

$$\frac{\dfrac{\overline{x : A \vee B} \qquad \dfrac{\overline{y : A}}{\mathsf{inr}_B\ y : B \vee A}\ \vee\mathsf{I_R} \qquad \dfrac{\overline{z : B}}{\mathsf{inl}_A\ z : B \vee A}\ \vee\mathsf{I_L}}{\mathsf{case}\ x\ \mathsf{of}\ \mathsf{inl}\ y.\, \mathsf{inr}_B\ y \mid \mathsf{inr}\ z.\, \mathsf{inl}_A\ z : B \vee A}\ \vee\mathsf{E}}{\lambda x{:}A \vee B.\, \mathsf{case}\ x\ \mathsf{of}\ \mathsf{inl}\ y.\, \mathsf{inr}_B\ y \mid \mathsf{inr}\ z.\, \mathsf{inl}_A\ z : (A \vee B) \supset (B \vee A)}\ \supset\mathsf{I}$$

Truth and falsehood

We use a *unit term* $()$ as a proof term for $\top\ \mathit{true}$:

$$\frac{}{\top\ \mathit{true}}\ \top\mathsf{I} \quad\Longleftrightarrow\quad \frac{}{() : \top}\ \top\mathsf{I}$$

Just like there is no logical content in $\top\ \mathit{true}$, a unit term carries no useful information. As truth $\top$ has no elimination rule, there is no more rule for $()$.

Since falsehood $\perp$ has no introduction rule, there is no proof term for type $\perp$. For the elimination rule $\perp\mathsf{E}$, we use an *abort term* $\mathsf{abort}_C\ M$:

$$\frac{\perp\ \mathit{true}}{C\ \mathit{true}}\ \perp\mathsf{E} \quad\Longleftrightarrow\quad \frac{M : \perp}{\mathsf{abort}_C\ M : C}\ \perp\mathsf{E}$$

We annotate an abort term with a type C so that an unambiguous type can be assigned when M has type $\perp$.

$$\frac{M : A \quad N : B}{(M, N) : A \wedge B}\ \wedge\mathsf{I} \qquad \frac{M : A \wedge B}{\mathsf{fst}\ M : A}\ \wedge\mathsf{E_L} \qquad \frac{M : A \wedge B}{\mathsf{snd}\ M : B}\ \wedge\mathsf{E_R} \qquad \frac{\begin{array}{c}\overline{x : A}\\ \vdots \\ M : B\end{array}}{\lambda x{:}A.\, M : A \supset B}\ \supset\mathsf{I} \qquad \frac{M : A \supset B \quad N : A}{M\ N : B}\ \supset\mathsf{E}$$

$$\frac{M : A}{\mathsf{inl}_B\ M : A \vee B}\ \vee\mathsf{I_L} \qquad \frac{M : B}{\mathsf{inr}_A\ M : A \vee B}\ \vee\mathsf{I_R} \qquad \frac{M : A \vee B \quad \begin{array}{c}\overline{x : A}\\ \vdots \\ N : C\end{array} \quad \begin{array}{c}\overline{y : B}\\ \vdots \\ N' : C\end{array}}{\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}\ x.\, N \mid \mathsf{inr}\ y.\, N' : C}\ \vee\mathsf{E}$$

$$\frac{}{() : \top}\ \top\mathsf{I} \qquad \frac{M : \bot}{\mathsf{abort}_C\ M : C}\ \bot\mathsf{E}$$

Figure 2.1: Typing rules for proof terms in propositional logic

Chapter 3

First-Order Logic

This chapter develops *first-order logic, i.e.,* logic with universal and existential quantifications. Developing first-order logic is the first step toward a practical reasoning system which inevitably demands an apparatus for expressing that a given property holds *for all, or* $\forall$, objects or that there *exists, or* $\exists$, a certain object satisfying a given property. Here we deal with pure first-order logic which does not stipulate a particular class of objects. Later we will enrich it in such a way that we can express properties of specific classes of objects such as natural numbers, trees, or boolean values.

3.1 Terms

In propositional logic, expressing properties of objects under consideration requires us to define propositional constants which denote atomic propositions. For example, in order to express that 1 is equal to 1 itself, we would need a propositional constant Eq_1 denoting an atomic proposition *'1 is equal to 1.'* While logical connectives provide us with an elegant mechanism for reasoning about such atomic propositions, the need for a separate propositional constant for each atomic proposition makes propositional logic too limited in its expressive power. For example, in order to express that every natural number is equal to itself, we would have to define an infinite array of propositional constants Eq_i denoting *'i is equal to i.'*

First-order logic replaces propositional constants in propositional logic by *predicates*. A predicate may have arguments and expresses a relation between its arguments. (For this reason, first-order logic is also called *predicate logic*.) For example, we can define a predicate *Eq* so that $Eq(t_1, t_2)$ denotes a proposition *'t_1 is equal to t_2.'* Here the predicate *Eq* has two arguments t_1 and t_2 and expresses an equality between t_1 and t_2. The arguments t_1 and t_2 are called *terms* in first-order logic and may be interpreted as particular mathematical objects (such as natural numbers). Thus first-order logic is a system in which we use predicates to express properties of terms.

Note that first-order logic itself does not enforce a specific way of interpreting terms. As an example, consider two terms $\mathbf{0}$ and $\mathsf{s}(\mathbf{0})$. As usual, we could interpret $\mathbf{0}$ as zero and $\mathsf{s}(\mathbf{0})$ as the successor of zero, but such an interpretation is just a specific way of assigning mathematical objects to terms. Thus it is also fine to interpret $\mathbf{0}$ as the natural number one and or $\mathsf{s}(\mathbf{0})$ as the predecessor of one. In general, we do not formalize how to relate terms to mathematical objects, and first-order logic in our discussion (which is based on proof theory) deals only with terms and not with their interpretations. Thus predicates directly express properties of uninterpreted terms.

Formally we define terms as follows:

$$\text{term} \quad t, s \quad ::= \quad x \mid y \mid \cdots \mid a \mid b \mid \cdots \mid f(t_1, \cdots, t_n) \mid c$$

$x, y, \cdots$ are called *term variables* which range over the set of all terms. We may substitute terms for term variables and we write $[s/x]t$ for the result of substituting s for x in t. $a, b, \cdots$ are called *parameters* and denote arbitrary/unspecified terms about which we can make no assumption. The difference between term variables and parameters is that a term variable is just a placeholder for another term whereas a parameter is understood as an arbitrary term about which nothing is known. (We will see the use of parameters in inference rules for first-order logic.)

f is called a *function symbol* and has zero or more arguments. We write $f(t_1, \cdots, t_n)$ for a term where f is a function symbol of arity n and $t_1, \cdots, t_n$ are its arguments. A *constant* c is a function symbol of zero arity; that is, c is an abbreviation of $c()$. Note that although it is usually interpreted as a function in the mathematical sense, a function symbol f is *not* a function because $f(t_1, \cdots, t_n)$ is a term in itself and does not reduce to another term. For example, $\mathsf{s}(\mathbf{0})$, which comprises of a function symbol s and its argument $\mathbf{0}$, does not reduce to another term, say $\mathbf{1}$, because it is a term in itself.

Now we can define a set of terms by specifying function symbols with their arities. Here are a few examples:

- To obtain terms for natural numbers, we use a constant $\mathbf{0}$ for zero and a function symbol s of arity one to be interpreted as the successor function.
- To obtain terms for boolean values, we use two constants $\mathbf{true}$ and $\mathbf{false}$.
- To obtain terms for binary trees, we use a constant $\mathbf{leaf}$ for leaf nodes and a function symbol $\mathbf{node}$ of arity two for inner nodes.

Terms are not to be confused with proof terms. Terms can represent any kinds of objects (*e.g.*, natural numbers, boolean values, student names, *etc.*) whereas proof terms represent proofs in logic. For example, we can say that a proof term $\lambda x\!:\!A.\,x$ represents a proof of $A \supset A$, but it makes no sense to judge the truth or falsehood of a term $\mathsf{s}(\mathbf{0})$.

3.2 Propositions in first-order logic

In addition to logical connectives from propositional logic, first-order logic uses predicates and two forms of quantifications over terms. An inductive definition of propositions is given as follows:

$$\text{proposition} \quad A \quad ::= \quad P(t_1, \cdots, t_n) \mid \cdots \mid \forall x.A \mid \exists x.A$$

Alternatively we may use three new formation rules:

$$\frac{}{P(t_1, \cdots, t_n)\ \mathit{prop}}\ P\mathsf{F} \qquad \frac{A\ \mathit{prop}}{\forall x.A\ \mathit{prop}}\ \forall\mathsf{F} \qquad \frac{A\ \mathit{prop}}{\exists x.A\ \mathit{prop}}\ \exists\mathsf{F}$$

P is called a *predicate symbol*. A predicate $P(t_1, \cdots, t_n)$ is a proposition that expresses a certain relation between terms $t_1, \cdots, t_n$. For example, we may use $\mathit{Nat}(t)$ to mean that term t is a natural number, or $\mathit{Eq}(t_1, t_2)$ to mean that terms t_1 and t_2 are equal. A propositional constant P is a predicate symbol of zero arity; that is, P is an abbreviation of $P()$.

$\forall x.A$ uses a *universal quantifier* $\forall$ to introduce a term variable x. Roughly speaking, the truth of $\forall x.A$ means that A is true for "every" term x. $\exists x.A$ uses an *existential quantifier* $\exists$ to introduce a term variable x. Roughly speaking, the truth of $\exists x.A$ means that we can present "some" term x for which A is true. Quantifiers $\forall$ and $\exists$ have the lowest operator precedence. For example, $\forall x.A \supset B$ is understood as $\forall x.(A \supset B)$; similarly $\exists x.A \supset B$ is understood as $\exists x.(A \supset B)$.

As quantifiers introduce term variables, there arises a need for substitutions for term variables in propositions or proofs. We write $[t/x]A$ for the result of substituting t for x in proposition A. Similarly we write $[t/x]\mathcal{D}$ for the result of substituting t for x throughout proof $\mathcal{D}$. Extending substitutions for term variables, we write $[t/a]A$ and $[t/a]\mathcal{D}$ for the result of substituting t for parameter a in A and $\mathcal{D}$, respectively. These substitutions for term variables and parameters are considerably simpler to define than substitutions in the simply-typed λ-calculus because variable captures never occur in first-order logic. That is, in a substitution $[t/x]A$ or $[t/x]\mathcal{D}$, term t is always closed and contains no free term variables.

3.3 Universal quantification

A universal quantification $\forall x.A$ is true if A is true for every term x. For example, given that $\mathbf{0}$, $\mathsf{s}(\mathbf{0})$, $\mathsf{s}(\mathsf{s}(\mathbf{0})), \cdots$ constitute the set of terms, we can deduce $\forall x.\mathit{Eq}(x,x)\ \mathit{true}$ if $\mathit{Eq}(\mathbf{0},\mathbf{0})\ \mathit{true}$, $\mathit{Eq}(\mathsf{s}(\mathbf{0}),\mathsf{s}(\mathbf{0}))\ \mathit{true}$, $\mathit{Eq}(\mathsf{s}(\mathsf{s}(\mathbf{0})),\mathsf{s}(\mathsf{s}(\mathbf{0})))\ \mathit{true}, \cdots$ are all provable. Hence it helps to think of $\forall x.A$ as an infinite conjunction

$$[t_1/x]A \wedge [t_2/x]A \wedge \cdots \wedge [t_i/x]A \wedge \cdots$$

where $t_1, t_2, \cdots, t_i, \cdots$ enumerate all terms.

The inference rules for universal quantifications are given as follows:

$$\dfrac{[a/x]A\ \mathit{true}}{\forall x.A\ \mathit{true}}\ \forall\mathsf{I}^a \qquad \dfrac{\forall x.A\ \mathit{true}}{[t/x]A\ \mathit{true}}\ \forall\mathsf{E}$$

In the rule $\forall\mathsf{I}^a$, parameter a denotes an arbitrary term about which we can make no assumption. Thus we may read $[a/x]A\ \mathit{true}$ as a shorthand for a sequence of judgments

$$[t_1/x]A\ \mathit{true} \quad [t_2/x]A\ \mathit{true} \quad \cdots \quad [t_i/x]A\ \mathit{true} \quad \cdots$$

where $t_1, t_2, \cdots, t_i, \cdots$ enumerate all terms. In the rule $\forall\mathsf{E}$, t can be any term — a constant, a function symbol, a term variable, or even an existing parameter. We justify the rule $\forall\mathsf{E}$ by reading $\forall x.A\ \mathit{true}$ as

$$[t_1/x]A \wedge [t_2/x]A \wedge \cdots \wedge [t_i/x]A \wedge \cdots\ \mathit{true}$$

where $t_1, t_2, \cdots, t_i, \cdots$ enumerate all terms.

It is important that in the rule $\forall\mathsf{I}^a$, parameter a must be fresh and not found in any undischarged hypothesis. For example, a proof of $\forall x.Nat(x)\ \mathit{true}$ introducing a fresh parameter a must not contain any hypothesis of the form $\overline{P(a)}$, which is an assumption on an arbitrary term about which we can make no assumption! The presence of such a hypothesis implies that parameter a is already declared elsewhere and thus cannot be interpreted as an arbitrary term. The following example, which tries to prove that y is a natural number whenever x is a natural number, shows that using the same parameter twice in difference instances of the rule $\forall\mathsf{I}$ results in a wrong proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{Nat(a)\ \mathit{true}}^{\,w}}{\forall x.Nat(x)\ \mathit{true}}\ \forall\mathsf{I}^a\ \mathit{(wrong)}}{Nat(b)\ \mathit{true}}\ \forall\mathsf{E}}{Nat(a)\supset Nat(b)\ \mathit{true}}\ \supset\mathsf{I}^w}{\forall y.Nat(a)\supset Nat(y)\ \mathit{true}}\ \forall\mathsf{I}^b}{\forall x.\forall y.Nat(x)\supset Nat(y)\ \mathit{true}}\ \forall\mathsf{I}^a$$

Here is an example of a proof involving universal quantifiers where we exploit $[a/x](A \wedge B) = [a/x]A \wedge [a/x]B$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\forall x.A\wedge B\ \mathit{true}}^{\,w}}{[a/x](A\wedge B)\ \mathit{true}}\ \forall\mathsf{E}}{[a/x]A\ \mathit{true}}\ \wedge\mathsf{E_L}}{\forall x.A\ \mathit{true}}\ \forall\mathsf{I}^a \quad \dfrac{\dfrac{\dfrac{\overline{\forall x.A\wedge B\ \mathit{true}}^{\,w}}{[a/x](A\wedge B)\ \mathit{true}}\ \forall\mathsf{E}}{[a/x]B\ \mathit{true}}\ \wedge\mathsf{E_R}}{\forall x.B\ \mathit{true}}\ \forall\mathsf{I}^a}{(\forall x.A)\wedge(\forall x.B)\ \mathit{true}}\ \wedge\mathsf{I}}{(\forall x.A\wedge B)\supset(\forall x.A)\wedge(\forall x.B)\ \mathit{true}}\ \supset\mathsf{I}^w$$

3.4 Existential quantification

An existential quantification $\exists x.A$ is true if there exists a term x satisfying A. For example, if $Eq(\mathbf{0},\mathbf{0})\ \mathit{true}$ is provable, we can deduce $\exists x.Eq(x,x)$ because substituting a concrete term $\mathbf{0}$ for x makes $Eq(x,x)\ \mathit{true}$ provable. Hence it helps to think of $\exists x.A$ as an infinite disjunction

$$[t_1/x]A \vee [t_2/x]A \vee \cdots \wedge [t_i/x]A \vee \cdots$$

where $t_1, t_2, \cdots, t_i, \cdots$ enumerate all terms.

The inference rules for existential quantifications are given as follows:

$$\dfrac{[t/x]A\ \mathit{true}}{\exists x.A\ \mathit{true}}\ \exists\mathsf{I} \qquad \dfrac{\exists x.A\ \mathit{true} \qquad \begin{array}{c}\overline{[a/x]A\ \mathit{true}}^{\,w}\\ \vdots\\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\ \exists\mathsf{E}^{a,w}$$

The rule $\exists\mathsf{I}$ says that we prove $\exists x.A\ \mathit{true}$ by presenting a concrete term, or a *witness*, t such that $[t/x]A\ \mathit{true}$ is provable. We justify the rule $\exists\mathsf{I}$ by reading $\exists x.A\ \mathit{true}$ as

$$[t_1/x]A \lor [t_2/x]A \lor \cdots \lor [t_i/x]A \lor \cdots\ \mathit{true}$$

where $t_1,\ t_2,\ \cdots,\ t_i,\ \cdots$ enumerate all terms and $t_i = t$ holds. In the rule $\exists\mathsf{E}^{a,w}$, we annotate the hypothesis $\overline{[a/x]A\ \mathit{true}}$ with label w. We also introduce a fresh parameter a because the witness for the proof of $\exists x.A\ \mathit{true}$ is unknown and thus we cannot make any assumption about it. Thus we may read $\begin{array}{c}\overline{[a/x]A\ \mathit{true}}^{\,w}\\ \vdots\\ C\ \mathit{true}\end{array}$ as a shorthand for a sequence of hypothetical proofs

$$\begin{array}{c}\overline{[t_1/x]A\ \mathit{true}}^{\,w}\\ \vdots\\ C\ \mathit{true}\end{array}\qquad \begin{array}{c}\overline{[t_2/x]A\ \mathit{true}}^{\,w}\\ \vdots\\ C\ \mathit{true}\end{array}\qquad \cdots \qquad \begin{array}{c}\overline{[t_i/x]A\ \mathit{true}}^{\,w}\\ \vdots\\ C\ \mathit{true}\end{array}\qquad \cdots$$

where $t_1, t_2, \cdots, t_i, \cdots$ enumerate all terms.

In the rule $\exists\mathsf{E}^{a,w}$, parameter a must be fresh and not found in proposition A or any undischarged hypothesis. In particular, it must not be found in proposition C. Otherwise the rule ends up with a conclusion that makes too strong an assumption about the witness, namely that the witness can be an arbitrary term! For example, the following proof exploits a proof of $\exists x.Nat(x) \land Eq(x, \mathbf{0})\ \mathit{true}$ to draw a (nonsensical) conclusion that an arbitrary term is equal to a natural number $\mathbf{0}$, as it allows parameter a to appear in the conclusion:

$$\dfrac{\begin{array}{c}\vdots\\ \exists x.Nat(x)\land Eq(x,\mathbf{0})\ \mathit{true}\end{array}\qquad \dfrac{\overline{Nat(a)\land Eq(a,\mathbf{0})\ \mathit{true}}^{\,w}}{Eq(a,\mathbf{0})\ \mathit{true}}\land\mathsf{E_R}}{Eq(a,\mathbf{0})\ \mathit{true}}\exists\mathsf{E}^{a,w}$$

In essence, the rule $\exists\mathsf{E}^{a,w}$ introduces parameter a in the course of proving $C\ \mathit{true}$ after fixing proposition C, which implies that C is oblivious to a.

An important aspect of the rule $\exists\mathsf{I}$ is that in order to prove $\exists x.A\ \mathit{true}$, it is not enough to show that there only "exists" a witness x satisfying A without actually knowing what it is. The necessity of such a witness is indeed a distinguishing feature of constructive logic. In contrast, a proof of $\exists x.A\ \mathit{true}$ in classical logic only needs to show that there exists a term t, *which may or may not be known,* such that $[t/x]A\ \mathit{true}$ is provable. In other words, a proof of $\exists x.A\ \mathit{true}$ essentially shows that it cannot happen that there exists no term t such that $[t/x]A\ \mathit{true}$ is provable. As a consequence, $\exists x.A$ is no different from $\neg\forall x.\neg A$ in classical logic.

To better understand the nature of existential quantifications in constructive logic, let us consider a few examples. First $\exists x.\neg A \supset \neg\forall x.A\ \mathit{true}$ is provable. Intuitively a proof of $\exists x.\neg A\ \mathit{true}$ gives us a witness t such that $[t/x]\neg A\ \mathit{true}$ is provable, and we can use t to refute $\forall x.A\ \mathit{true}$.

$$\dfrac{\dfrac{\dfrac{\overline{\exists x.\neg A\ \mathit{true}}^{\,w}\qquad \dfrac{\overline{[a/x]\neg A\ \mathit{true}}^{\,y}\quad \dfrac{\overline{\forall x.A\ \mathit{true}}^{\,z}}{[a/x]A\ \mathit{true}}\forall\mathsf{E}}{\bot\ \mathit{true}}\neg\mathsf{E}}{\bot\ \mathit{true}}\exists\mathsf{E}^{a,y}}{\neg\forall x.A\ \mathit{true}}\neg\mathsf{I}^{z}}{\exists x.\neg A\supset\neg\forall x.A\ \mathit{true}}\supset\mathsf{I}^{w}$$

The converse $\neg\forall x.A \supset \exists x.\neg A\ \mathit{true}$ is not provable, however. Intuitively a proof of $\exists x.\neg A\ \mathit{true}$ requires a witness t such that $[t/x]\neg A\ \mathit{true}$ is provable, but no proof of $\neg\forall x.A\ \mathit{true}$ gives such a witness.

$$\dfrac{\dfrac{\dfrac{\overline{\neg\forall x.A\ \mathit{true}}^{\,w}\qquad \overset{?}{\forall x.A\ \mathit{true}}}{\bot\ \mathit{true}}\neg\mathsf{E}}{\exists x.\neg A\ \mathit{true}}\bot\mathsf{E}}{\neg\forall x.A\supset\exists x.\neg A\ \mathit{true}}\supset\mathsf{I}^{w}$$

$$\frac{[a/x]A\ \mathit{true}}{\forall x.A\ \mathit{true}}\ \forall\mathsf{I}^a \qquad \frac{\forall x.A\ \mathit{true}}{[t/x]A\ \mathit{true}}\ \forall\mathsf{E} \qquad \frac{[t/x]A\ \mathit{true}}{\exists x.A\ \mathit{true}}\ \exists\mathsf{I} \qquad \frac{\exists x.A\ \mathit{true} \qquad \begin{array}{c}\overline{[a/x]A\ \mathit{true}}^{\,w}\\ \vdots \\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\ \exists\mathsf{E}^{a,w}$$

Figure 3.1: Natural deduction system for first-order logic

Perhaps surprisingly, $(\forall x.A) \supset (\exists x.A)\ \mathit{true}$ is *not* provable. The reason is that although $\forall x.A\ \mathit{true}$ states that $[t/x]A\ \mathit{true}$ is provable for any term t, it does not decide a concrete term t such that $[t/x]A\ \mathit{true}$ is provable. In particular, if the set of terms is empty, $\forall x.A\ \mathit{true}$ holds trivially (because there is no term), but $\exists x.A\ \mathit{true}$ never holds because it is impossible to choose a term t for x, regardless of proposition A.

$$\cfrac{\cfrac{\cfrac{\overline{\forall x.A\ \mathit{true}}^{\,w}}{[t/x]A\ \mathit{true}?}\ \forall\mathsf{E}}{\exists x.A\ \mathit{true}}\ \exists\mathsf{I}}{(\forall x.A) \supset (\exists x.A)\ \mathit{true}}\ \supset\mathsf{I}^w$$

On the other hand, $\forall y.(\forall x.A) \supset (\exists x.A)\ \mathit{true}$ *is* provable even if y does not occur free in A. The difference from the previous example is that $\forall y$ allows us to make an assumption that the set of terms is not empty. In the proof shown below, parameter a denotes an arbitrary term in the set of terms, and its presence implies that the set of terms is not empty.

$$\cfrac{\cfrac{\cfrac{\cfrac{\overline{\forall x.A\ \mathit{true}}^{\,w}}{[a/x]A\ \mathit{true}}\ \forall\mathsf{E}}{\exists x.A\ \mathit{true}}\ \exists\mathsf{I}}{(\forall x.A) \supset (\exists x.A)\ \mathit{true}}\ \supset\mathsf{I}^w}{\forall y.(\forall x.A) \supset (\exists x.A)\ \mathit{true}}\ \forall\mathsf{I}^a$$

These two examples illustrate that in constructive logic, $\forall x.A$ is not equivalent to A even if x does not occur free in A at all: $\forall x.A$ asserts A on the assumption that the set of terms is not empty, whereas A without a universal quantifier cannot exploit such an assumption.

Figure 3.1 shows the inference rules for first-order logic.

3.5 Examples

As a concrete example of reasoning in first-order logic, let us characterize natural numbers. We use $\mathbf{0}$ as a term denoting zero and s as a function symbol denoting the successor function. We also use three predicates: $\mathit{Nat}(t)$ to mean that t is a natural number, $\mathit{Eq}(t, t')$ to mean that t and t' are equal, and $\mathit{Lt}(t, t')$ to mean that t is less than t'.

First we need axioms as a means of defining the three predicates:

$$\frac{}{\mathit{Nat}(\mathbf{0})\ \mathit{true}}\ \mathit{Zero} \qquad \frac{}{\forall x.\mathit{Nat}(x) \supset \mathit{Nat}(\mathbf{s}(x))\ \mathit{true}}\ \mathit{Succ}$$

$$\frac{}{\forall x.\mathit{Eq}(x, x)\ \mathit{true}}\ \mathit{Eq}_i \qquad \frac{}{\forall x.\forall y.\forall z.(\mathit{Eq}(x, y) \wedge \mathit{Eq}(x, z)) \supset \mathit{Eq}(y, z)\ \mathit{true}}\ \mathit{Eq}_t$$

$$\frac{}{\forall x.\mathit{Lt}(x, \mathbf{s}(x))\ \mathit{true}}\ \mathit{Lt}_s \qquad \frac{}{\forall x.\forall y.\mathit{Eq}(x, y) \supset \neg\mathit{Lt}(x, y)\ \mathit{true}}\ \mathit{Lt}_\neg$$

The lower four axioms may be thought of as translations of the following mathematical properties:

- $x = x$.
- If $x = y$ and $x = z$, then $y = z$.
- $x < x + 1$.
- If $x = y$, then $x \not< y$.

Combined with these axioms, first-order logic allows us to prove new theorems about these predicates. As a trivial example, here is a proof of $Nat(\mathbf{s}(\mathbf{s}(\mathbf{0})))\ true$, which states that $\mathbf{s}(\mathbf{s}(\mathbf{0}))$ is a natural number:

$$\dfrac{\dfrac{\dfrac{}{\forall x.Nat(x) \supset Nat(\mathbf{s}(x))\ true}\,Succ}{Nat(\mathbf{s}(\mathbf{0})) \supset Nat(\mathbf{s}(\mathbf{s}(\mathbf{0})))\ true}\,\forall\mathsf{E} \qquad \dfrac{\dfrac{\dfrac{}{\forall x.Nat(x) \supset Nat(\mathbf{s}(x))\ true}\,Succ}{Nat(\mathbf{0}) \supset Nat(\mathbf{s}(\mathbf{0}))\ true}\,\forall\mathsf{E} \qquad \dfrac{}{Nat(\mathbf{0})\ true}\,Zero}{Nat(\mathbf{s}(\mathbf{0}))\ true}\,\supset\mathsf{E}}{Nat(\mathbf{s}(\mathbf{s}(\mathbf{0})))\ true}\,\supset\mathsf{E}$$

Note that the two applications of the rule $\supset$E substitute different terms, namely $\mathbf{s}(\mathbf{0})$ and $\mathbf{0}$, for term variable x in $\forall x.Nat(x) \supset Nat(\mathbf{s}(x))$.

An example of using an existential quantification is a proof of $\forall x.Nat(x) \supset (\exists y.Nat(y) \wedge Eq(x, y))\ true$ which states that if x is a natural number, there exists a natural number y such that $x = y$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Nat(a)\ true}\,z \qquad \dfrac{\dfrac{}{\forall x.Eq(x, x)\ true}\,Eq_i}{Eq(a, a)\ true}\,\forall\mathsf{E}}{Nat(a) \wedge Eq(a, a)\ true}\,\wedge\mathsf{I}}{\exists y.Nat(y) \wedge Eq(a, y)\ true}\,\exists\mathsf{I}}{Nat(a) \supset (\exists y.Nat(y) \wedge Eq(a, y))\ true}\,\supset\mathsf{I}^z}{\forall x.Nat(x) \supset (\exists y.Nat(y) \wedge Eq(x, y))\ true}\,\forall\mathsf{I}^a$$

In the application of the rule $\exists$I, we use parameter a as a witness.

Here are two more examples. The first states the commutativity of equality: $x = y$ implies $y = x$ The second states that there is no term x such that $x = 0$ and $x = 1$.

- Proof of $\forall x.\forall y.Eq(x, y) \supset Eq(y, x)\ true$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall x.\forall y.\forall z.(Eq(x, y) \wedge Eq(x, z)) \supset Eq(y, z)\ true}\,Eq_t}{\forall y.\forall z.(Eq(a, y) \wedge Eq(a, z)) \supset Eq(y, z)\ true}\,\forall\mathsf{E}}{\forall z.(Eq(a, b) \wedge Eq(a, z)) \supset Eq(b, z)\ true}\,\forall\mathsf{E}}{(Eq(a, b) \wedge Eq(a, a)) \supset Eq(b, a)\ true}\,\forall\mathsf{E} \qquad \dfrac{\dfrac{}{Eq(a, b)\ true}\,w \qquad \dfrac{\dfrac{}{\forall x.Eq(x, x)\ true}\,Eq_i}{Eq(a, a)\ true}\,\forall\mathsf{E}}{Eq(a, b) \wedge Eq(a, a)\ true}\,\wedge\mathsf{I}}{Eq(b, a)\ true}\,\supset\mathsf{E}}{Eq(a, b) \supset Eq(b, a)\ true}\,\supset\mathsf{I}^w}{\forall y.Eq(a, y) \supset Eq(y, a)\ true}\,\forall\mathsf{I}^b}{\forall x.\forall y.Eq(x, y) \supset Eq(y, x)\ true}\,\forall\mathsf{I}^a$$

- Proof of $\neg\exists x.Eq(x, \mathbf{0}) \wedge Eq(x, \mathbf{s}(\mathbf{0}))\ true$:

$$\dfrac{\dfrac{\dfrac{}{\exists x.Eq(x, \mathbf{0}) \wedge Eq(x, \mathbf{s}(\mathbf{0}))\ true}\,w \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall x.\forall y.Eq(x, y) \supset \neg Lt(x, y)\ true}\,Lt_\neg}{\forall y.Eq(\mathbf{0}, y) \supset \neg Lt(\mathbf{0}, y)\ true}\,\forall\mathsf{E}}{Eq(\mathbf{0}, \mathbf{s}(\mathbf{0})) \supset \neg Lt(\mathbf{0}, \mathbf{s}(\mathbf{0}))\ true}\,\forall\mathsf{E} \qquad \begin{matrix}\mathcal{D}\\ Eq(\mathbf{0}, \mathbf{s}(\mathbf{0}))\ true\end{matrix}}{\neg Lt(\mathbf{0}, \mathbf{s}(\mathbf{0}))\ true}\,\supset\mathsf{E} \qquad \dfrac{\dfrac{}{\forall x.Lt(x, \mathbf{s}(x))\ true}\,Lt_s}{Lt(\mathbf{0}, \mathbf{s}(\mathbf{0}))\ true}\,\forall\mathsf{E}}{\bot\ true}\,\neg\mathsf{E}}{\bot\ true}\,\exists\mathsf{E}^{a,z}}{\neg\exists x.Eq(x, \mathbf{0}) \wedge Eq(x, \mathbf{s}(\mathbf{0}))\ true}\,\neg\mathsf{I}^w$$

where we let

$$\mathcal{D} \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\forall x.\forall y.\forall z.(Eq(x, y) \wedge Eq(x, z)) \supset Eq(y, z)\ true}\,Eq_t}{\forall y.\forall z.(Eq(a, y) \wedge Eq(a, z)) \supset Eq(y, z)\ true}\,\forall\mathsf{E}}{\forall z.(Eq(a, \mathbf{0}) \wedge Eq(a, z)) \supset Eq(\mathbf{0}, z)\ true}\,\forall\mathsf{E}}{(Eq(a, \mathbf{0}) \wedge Eq(a, \mathbf{s}(\mathbf{0}))) \supset Eq(\mathbf{0}, \mathbf{s}(\mathbf{0}))\ true}\,\forall\mathsf{E} \qquad \dfrac{}{Eq(a, \mathbf{0}) \wedge Eq(a, \mathbf{s}(\mathbf{0}))\ true}\,z$$
$$\overline{\qquad\qquad Eq(\mathbf{0}, \mathbf{s}(\mathbf{0}))\ true \qquad\qquad}\,\supset\mathsf{E}$$

3.6 Proof terms

As in propositional logic, we use the Curry-Howard isomorphism to represent proofs of truth judgments as proof terms. Proof terms for first-order logic are given as follows:

$$\text{proof term} \quad M \quad ::= \quad \cdots \mid \lambda x.\, M \mid M\ t \mid \langle t, M \rangle \mid \mathsf{let}\ \langle x, w \rangle = M\ \mathsf{in}\ M$$

$\lambda x.\, M$, a proof term of type $\forall x.A$, is a λ-abstraction that takes a term t and returns a proof term of type $[t/x]A$. (Recall that propositions and types are equivalent under the Curry-Howard isomorphism.) It is similar to a λ-abstraction from propositional logic except that it takes a term, instead of a proof term, as its argument. For example, given a term t (denoting a natural number), $\lambda x.\, M$ may return a proof term of type $Nat(t) \supset Nat(\mathsf{s}(t))$. A corresponding λ-application $M\ t$ is a proof term of type $[t/x]A$ if M is a proof term of type $\forall x.A$.

The typing rules for $\lambda x.\, M$ and $M\ t$ are given as follows:

$$\frac{[a/x]M : [a/x]A}{\lambda x.\, M : \forall x.A}\ \forall\mathsf{I}^a \qquad \frac{M : \forall x.A}{M\ t : [t/x]A}\ \forall\mathsf{E}$$

Here we write $[t/x]M$ for a substitution of term t for term variable x in proof term M. The rule $\forall\mathsf{I}^a$ proves that $\lambda x.\, M$ has type $\forall x.A$ by introducing a fresh parameter a and proving that $[a/x]M$ has type $[a/x]A$. For example, a proof that $\lambda x.\, M$ has type $\forall x.Eq(x, x)$ could show that $[a/x]M$ has type $Eq(a, a)$ for some parameter a. Note that in the rule $\forall\mathsf{I}^a$, term a appears in both proof term M and type A. This feature of first-order logic is manifested in the rule $\forall\mathsf{E}$: terms may appear not only in types but also in proof terms. Intuitively a proof about a specific term t needs to mention t somewhere in it. (Otherwise how can we prove a fact about t at all?) Hence a proof term whose type contains t also mentions t somewhere in it. For example, we could use $\mathsf{Eq_i}\ \mathbf{0}$ as a proof term of type $Eq(\mathbf{0}, \mathbf{0})$ where $\mathsf{Eq_i}$ assumes type $\forall x.Eq(x, x)$. As a consequence, a substitution $[t/x]M$ on proof term M may need a substitution $[t/x]A$ on type A if x occurs inside A in M.

$\langle t, M \rangle$ is a proof term of type $\exists x.A$. Intuitively a proof of $\exists x.A$ *true* requires a concrete witness t and a proof that t satisfies A. Hence a proof term of type $\exists x.A$ contains such a witness t and a proof term M of type $[t/x]A$. For example, a proof term of type $\exists x.Eq(x, x)$ ("there exists a term x such that x is equal to x itself") may contain a witness $\mathbf{0}$ and a proof term of type $Eq(\mathbf{0}, \mathbf{0})$ ("$\mathbf{0}$ is equal to $\mathbf{0}$). Thus we obtain the following typing rule for $\langle t, M \rangle$:

$$\frac{M : [t/x]A}{\langle t, M \rangle : \exists x.A}\ \exists\mathsf{I}$$

Given that M has type $\exists x.A$, a proof term $\mathsf{let}\ \langle x, w \rangle = M\ \mathsf{in}\ N$ decides the type of N after binding x and w to a witness t and a proof term of type $[t/x]A$, respectively. (Note that x is a term variable whereas w is a variable ranging over proof terms.) Since such a witness is unknown in general (*e.g.*, if M is a variable), we have to assume an arbitrary witness a and assign type $[a/x]A$ to w. Accordingly we replace x in N by a. Thus we obtain the following typing rule for $\mathsf{let}\ \langle x, w \rangle = M\ \mathsf{in}\ N$:

$$\frac{M : \exists x.A \qquad \begin{array}{c} \overline{w : [a/x]A} \\ \vdots \\ [a/x]N : C \end{array}}{\mathsf{let}\ \langle x, w \rangle = M\ \mathsf{in}\ N : C}\ \exists\mathsf{E}^a$$

In practice, we may use the following typing rule with an extra assumption that x is a fresh term variable:

$$\frac{M : \exists x.A \qquad \begin{array}{c} \overline{w : A} \\ \vdots \\ N : C \end{array}}{\mathsf{let}\ \langle x, w \rangle = M\ \mathsf{in}\ N : C}\ \exists\mathsf{E}$$

Figure 3.2 shows all the typing rules for proof terms in first-order logic.

$$\dfrac{[a/x]M : [a/x]A}{\lambda x.\, M : \forall x.A}\,\forall\mathsf{I}^a \qquad \dfrac{M : \forall x.A}{M\;t : [t/x]A}\,\forall\mathsf{E} \qquad \dfrac{M : [t/x]A}{\langle t, M\rangle : \exists x.A}\,\exists\mathsf{I} \qquad \dfrac{M : \exists x.A \qquad \begin{array}{c}\overline{w : [a/x]A}\\ \vdots \\ [a/x]N : C\end{array}}{\mathsf{let}\ \langle x, w\rangle = M\ \mathsf{in}\ N : C}\,\exists\mathsf{E}^a$$

Figure 3.2: Typing rules for proof terms in first-order logic

3.7 Examples of proof terms

This section rewrites all the proofs in Section 3.5 using proof terms. First we need constant proof terms for axioms:

$$\dfrac{}{\mathsf{Nat_0} : Nat(\mathbf{0})}\,\mathit{Zero} \qquad \dfrac{}{\mathsf{Nat_s} : \forall x.Nat(x) \supset Nat(\mathsf{s}(x))}\,\mathit{Succ}$$

$$\dfrac{}{\mathsf{Eq_i} : \forall x.Eq(x,x)}\,\mathit{Eq_i} \qquad \dfrac{}{\mathsf{Eq_t} : \forall x.\forall y.\forall z.(Eq(x,y) \wedge Eq(x,z)) \supset Eq(y,z)}\,\mathit{Eq_t}$$

$$\dfrac{}{\mathsf{Lt_s} : \forall x.Lt(x,\mathsf{s}(x))}\,\mathit{Lt_s} \qquad \dfrac{}{\mathsf{Lt_\neg} : \forall x.\forall y.Eq(x,y) \supset \neg Lt(x,y)}\,\mathit{Lt_\neg}$$

The proof of $Nat(\mathsf{s}(\mathsf{s}(\mathbf{0})))$ *true* corresponds to a proof term $\mathsf{Nat_s}\ \mathsf{s}(\mathbf{0})\ (\mathsf{Nat_s}\ \mathbf{0}\ \mathsf{Nat_0})$ as shown in the following derivation tree:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{Nat_s} : \forall x.Nat(x) \supset Nat(\mathsf{s}(x))}\,\mathit{Succ}}{\mathsf{Nat_s}\ \mathsf{s}(\mathbf{0}) : Nat(\mathsf{s}(\mathbf{0})) \supset Nat(\mathsf{s}(\mathsf{s}(\mathbf{0})))}\,\forall\mathsf{E} \qquad \dfrac{\dfrac{\dfrac{}{\mathsf{Nat_s} : \forall x.Nat(x) \supset Nat(\mathsf{s}(x))}\,\mathit{Succ}}{\mathsf{Nat_s}\ \mathbf{0} : Nat(\mathbf{0}) \supset Nat(\mathsf{s}(\mathbf{0}))}\,\forall\mathsf{E} \qquad \dfrac{}{\mathsf{Nat_0} : Nat(\mathbf{0})}\,\mathit{Zero}}{\mathsf{Nat_s}\ \mathbf{0}\ \mathsf{Nat_0} : Nat(\mathsf{s}(\mathbf{0}))}\,\supset\mathsf{E}}{\mathsf{Nat_s}\ \mathsf{s}(\mathbf{0})\ (\mathsf{Nat_s}\ \mathbf{0}\ \mathsf{Nat_0}) : Nat(\mathsf{s}(\mathsf{s}(\mathbf{0})))}\,\supset\mathsf{E}$$

In the same fashion, we obtain the following proof terms:

- Proof term of type $\forall x.Nat(x) \supset (\exists y.Nat(y) \wedge Eq(x,y))$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{z : Nat(a)} \qquad \dfrac{\dfrac{}{\mathsf{Eq_i} : \forall x.Eq(x,x)}\,\mathit{Eq_i}}{\mathsf{Eq_i}\ a : Eq(a,a)}\,\forall\mathsf{E}}{(z, \mathsf{Eq_i}\ a) : Nat(a) \wedge Eq(a,a)}\,\wedge\mathsf{I}}{\langle a, (z, \mathsf{Eq_i}\ a)\rangle : \exists y.Nat(y) \wedge Eq(a,y)}\,\exists\mathsf{I}}{\lambda z{:}Nat(a).\, \langle a, (z, \mathsf{Eq_i}\ a)\rangle : Nat(a) \supset (\exists y.Nat(y) \wedge Eq(a,y))}\,\supset\mathsf{I}^z}{\lambda x.\, \lambda z{:}Nat(x).\, \langle x, (z, \mathsf{Eq_i}\ x)\rangle : \forall x.Nat(x) \supset (\exists y.Nat(y) \wedge Eq(x,y))}\,\forall\mathsf{I}^a$$

- Proof term of type $\forall x.\forall y.Eq(x,y) \supset Eq(y,x)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{Eq_t} : \forall x.\forall y.\forall z.(Eq(x,y) \wedge Eq(x,z)) \supset Eq(y,z)}\,\mathit{Eq_t}}{\mathsf{Eq_t}\ a : \forall y.\forall z.(Eq(a,y) \wedge Eq(a,z)) \supset Eq(y,z)}\,\forall\mathsf{E}}{\mathsf{Eq_t}\ a\ b : \forall z.(Eq(a,b) \wedge Eq(a,z)) \supset Eq(b,z)}\,\forall\mathsf{E}}{\mathsf{Eq_t}\ a\ b\ a : (Eq(a,b) \wedge Eq(a,a)) \supset Eq(b,a)}\,\forall\mathsf{E} \qquad \dfrac{\dfrac{}{w : Eq(a,b)} \qquad \dfrac{\dfrac{}{\mathsf{Eq_i} : \forall x.Eq(x,x)}\,\mathit{Eq_i}}{\mathsf{Eq_i}\ a : Eq(a,a)}\,\forall\mathsf{E}}{(w, \mathsf{Eq_i}\ a) : Eq(a,b) \wedge Eq(a,a)}\,\wedge\mathsf{I}}{\mathsf{Eq_t}\ a\ b\ a\ (w, \mathsf{Eq_i}\ a) : Eq(b,a)}\,\supset\mathsf{E}}{\lambda w{:}Eq(a,b).\, \mathsf{Eq_t}\ a\ b\ a\ (w, \mathsf{Eq_i}\ a) : Eq(a,b) \supset Eq(b,a)}\,\supset\mathsf{I}^w}{\lambda y.\, \lambda w{:}Eq(a,y).\, \mathsf{Eq_t}\ a\ y\ a\ (w, \mathsf{Eq_i}\ a) : \forall y.Eq(a,y) \supset Eq(y,a)}\,\forall\mathsf{I}^b}{\lambda x.\, \lambda y.\, \lambda w{:}Eq(x,y).\, \mathsf{Eq_t}\ x\ y\ x\ (w, \mathsf{Eq_i}\ x) : \forall x.\forall y.Eq(x,y) \supset Eq(y,x)}\,\forall\mathsf{I}^a$$

- Proof term of type $\neg\exists x.Eq(x,\mathbf{0}) \wedge Eq(x,\mathsf{s}(\mathbf{0}))$:

$$\dfrac{\dfrac{\dfrac{}{w : \exists x.Eq(x,\mathbf{0}) \wedge Eq(x,\mathsf{s}(\mathbf{0}))} \qquad \dfrac{\begin{array}{c}\mathcal{E}\\ (\mathsf{Lt_\neg}\ \mathbf{0}\ \mathsf{s}(\mathbf{0}))\ (\mathsf{Eq_t}\ a\ \mathbf{0}\ \mathsf{s}(\mathbf{0})\ z) : \neg Lt(\mathbf{0},\mathsf{s}(\mathbf{0}))\end{array} \qquad \dfrac{\dfrac{}{\mathsf{Lt_s} : \forall x.Lt(x,\mathsf{s}(x))}\,\mathit{Lt_s}}{\mathsf{Lt_s}\ \mathbf{0} : Lt(\mathbf{0},\mathsf{s}(\mathbf{0}))}\,\forall\mathsf{E}}{(\mathsf{Lt_\neg}\ \mathbf{0}\ \mathsf{s}(\mathbf{0}))\ (\mathsf{Eq_t}\ a\ \mathbf{0}\ \mathsf{s}(\mathbf{0})\ z)\ (\mathsf{Lt_s}\ \mathbf{0}) : \bot}\,\neg\mathsf{E}}{\mathsf{let}\ \langle x, z\rangle = w\ \mathsf{in}\ (\mathsf{Lt_\neg}\ \mathbf{0}\ \mathsf{s}(\mathbf{0}))\ (\mathsf{Eq_t}\ x\ \mathbf{0}\ \mathsf{s}(\mathbf{0})\ z)\ (\mathsf{Lt_s}\ \mathbf{0}) : \bot}\,\exists\mathsf{E}^a}{\lambda w{:}\exists x.Eq(x,\mathbf{0}) \wedge Eq(x,\mathsf{s}(\mathbf{0})).\, \mathsf{let}\ \langle x, z\rangle = w\ \mathsf{in}\ (\mathsf{Lt_\neg}\ \mathbf{0}\ \mathsf{s}(\mathbf{0}))\ (\mathsf{Eq_t}\ x\ \mathbf{0}\ \mathsf{s}(\mathbf{0})\ z)\ (\mathsf{Lt_s}\ \mathbf{0}) : \neg\exists x.Eq(x,\mathbf{0}) \wedge Eq(x,\mathsf{s}(\mathbf{0}))}\,\neg\mathsf{I}^w$$

where we let

$$\mathcal{E} \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{Lt}_\neg : \forall x.\forall y.Eq(x,y) \supset \neg Lt(x,y)}\;Lt_\neg}{\mathsf{Lt}_\neg\ \mathbf{0} : \forall y.Eq(\mathbf{0},y) \supset \neg Lt(\mathbf{0},y)}\;\forall\mathsf{E}}{\mathsf{Lt}_\neg\ \mathbf{0}\ \mathrm{s}(\mathbf{0}) : Eq(\mathbf{0},\mathrm{s}(\mathbf{0})) \supset \neg Lt(\mathbf{0},\mathrm{s}(\mathbf{0}))}\;\forall\mathsf{E} \quad \begin{array}{c}\mathcal{D}\\ \mathsf{Eq_t}\ a\ \mathbf{0}\ \mathrm{s}(\mathbf{0})\ z : Eq(\mathbf{0},\mathrm{s}(\mathbf{0}))\end{array}}{(\mathsf{Lt}_\neg\ \mathbf{0}\ \mathrm{s}(\mathbf{0}))\ (\mathsf{Eq_t}\ a\ \mathbf{0}\ \mathrm{s}(\mathbf{0})\ z) : \neg Lt(\mathbf{0},\mathrm{s}(\mathbf{0}))}\;\supset\mathsf{E}$$

where we let

$$\mathcal{D} \quad = \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{Eq_t} : \forall x.\forall y.\forall z.(Eq(x,y) \wedge Eq(x,z)) \supset Eq(y,z)}\;Eq_t}{\mathsf{Eq_t}\ a : \forall y.\forall z.(Eq(a,y) \wedge Eq(a,z)) \supset Eq(y,z)}\;\forall\mathsf{E}}{\mathsf{Eq_t}\ a\ \mathbf{0} : \forall z.(Eq(a,\mathbf{0}) \wedge Eq(a,z)) \supset Eq(\mathbf{0},z)}\;\forall\mathsf{E}}{\mathsf{Eq_t}\ a\ \mathbf{0}\ \mathrm{s}(\mathbf{0}) : (Eq(a,\mathbf{0}) \wedge Eq(a,\mathrm{s}(\mathbf{0}))) \supset Eq(\mathbf{0},\mathrm{s}(\mathbf{0}))}\;\forall\mathsf{E} \quad \dfrac{}{z : Eq(a,\mathbf{0}) \wedge Eq(a,\mathrm{s}(\mathbf{0}))}}{\mathsf{Eq_t}\ a\ \mathbf{0}\ \mathrm{s}(\mathbf{0})\ z : Eq(\mathbf{0},\mathrm{s}(\mathbf{0}))}\;\supset\mathsf{E}$$

Chapter 4

Datatypes

In pure first-order logic, term variables are assumed to range over all kinds of terms and their domains are left unspecified. Hence we can restrict the domain of a term variable only indirectly by using a predicate corresponding to a particular domain. For example, we may use a predicate $Nat(x)$ to specify that x ranges over natural numbers, as in:

$$\forall x.Nat(x) \supset A$$
$$\exists x.Nat(x) \wedge A$$

This chapter develops first-order logic with *datatypes* which explicitly specifies the domain of each term variable bound by a quantifier $\forall$ or $\exists$. We write $\forall x \in \tau.A$ and $\exists x \in \tau.A$ to specify that term variable x in proposition A ranges over datatype τ. For example, the above two propositions can now be concisely written as

$$\forall x \in \mathsf{nat}.A$$
$$\exists x \in \mathsf{nat}.A$$

where nat is a datatype for natural numbers.

The main judgment for first-order logic with datatypes is $t \in \tau$:

$$t \in \tau \quad \Leftrightarrow \quad \textit{term } t \textit{ has datatype } \tau$$

As in propositional logic and pure first-order logic, we base the development of datatypes on natural deduction. For example, each datatype τ is accompanied by introduction and elimination rules for deducing and exploiting judgments $t \in \tau$. We use metavariables τ and σ for datatypes, and t and s for terms.

From this chapter on, we adopt a new notation $A(x)$ to mean that proposition A contains term variable x, as in $\forall x \in \mathsf{nat}.A(x)$ and $\exists x \in \mathsf{nat}.A(x)$. Accordingly $A(t)$ stands for A in which every occurrence of x has been replaced by t. That is, we have $A(t) = [t/x]A$.

4.1 Basic constructors for datatypes

Before we consider concrete datatypes such as bool for boolean values and nat for natural numbers, we develop basic constructors for datatypes to obtain a general language similar to the simply-typed λ-calculus:

$$\text{datatype} \quad \tau \quad ::= \quad \cdots \mid \tau \rightarrow \tau \mid \tau \times \tau \mid \tau + \tau \mid \mathsf{unit} \mid \mathsf{void}$$

We call $\tau \rightarrow \sigma$ a function type, $\tau \times \sigma$ a product type, $\tau + \sigma$ a sum type, unit a unit type, and void a void type. We will use terms of these datatypes as programs for manipulating ordinary terms of such concrete datatypes. For example, we use a term of datatype $\mathsf{nat} \rightarrow \mathsf{bool}$ as a function mapping natural numbers to boolean values, and a term of datatype $\mathsf{nat} \times \mathsf{nat}$ to carry a pair of natural numbers. We assume that $\rightarrow, \times, +$ are all right-associative.

The basic constructors for datatypes have their counterparts in the simply-typed λ-calculus as follows:

datatype	$\rightarrow$	$\times$	$+$	unit	void
type	$\supset$	$\wedge$	$\vee$	$\top$	$\bot$

$$\dfrac{\begin{array}{c}\overline{x \in \tau}\\ \vdots \\ t \in \sigma\end{array}}{\lambda x \in \tau.\, t \in \tau \to \sigma}\to\mathsf{I} \quad \dfrac{t \in \tau\to\sigma \quad s\in\tau}{t\,s \in \sigma}\to\mathsf{E} \quad \dfrac{t \in \tau \quad s \in \sigma}{\langle t, s\rangle \in \tau\times\sigma}\times\mathsf{I} \quad \dfrac{t \in \tau\times\sigma}{\mathbf{fst}\ t \in \tau}\times\mathsf{E_L} \quad \dfrac{t \in \tau\times\sigma}{\mathbf{snd}\ t \in \sigma}\times\mathsf{E_R}$$

$$\dfrac{t \in \tau}{\mathbf{inl}_\sigma\ t \in \tau+\sigma}+\mathsf{I_L} \quad \dfrac{t \in \sigma}{\mathbf{inr}_\tau\ t \in \tau\vee\sigma}+\mathsf{I_R} \quad \dfrac{t \in \tau+\tau' \quad \begin{array}{c}\overline{x \in \tau}\\ \vdots \\ s \in \sigma\end{array} \quad \begin{array}{c}\overline{y \in \tau'}\\ \vdots \\ s' \in \sigma\end{array}}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}\ x.\, s \mid \mathbf{inr}\ y.\, s' \in \sigma}+\mathsf{E}$$

$$\dfrac{}{\langle\rangle \in \mathsf{unit}}\,\mathsf{unitI} \quad \dfrac{t \in \mathsf{void}}{\mathbf{abort}_\tau\ t \in \tau}\,\mathsf{voidE}$$

Figure 4.1: Typing rules for terms

For example, function types of the form $\tau\to\sigma$ correspond to types of the form $A \supset B$. Terms for these datatypes also have their counterparts in the simply-typed λ-calculus. For example, as we use a λ-abstraction $\lambda x\!:\!A.\, M$ as a proof term of type $A \supset B$, we use another form of λ-abstraction $\lambda x \in \tau.\, t$ as a term of datatype $\tau\to\sigma$. The definition of terms reuses the syntax for proof terms in the simply-typed λ-calculus with a few cosmetic changes:

$$\begin{array}{lcl}\text{term} \quad t & ::= & \cdots \mid \lambda x \in \tau.\, t \mid t\ t \mid \langle t, t\rangle \mid \mathbf{fst}\ t \mid \mathbf{snd}\ t \mid \mathbf{inl}_\tau\ t \mid \mathbf{inr}_\tau\ t \mid \mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}\ x.\, t \mid \mathbf{inr}\ x.\, t \mid \\ & & \langle\rangle \mid \mathbf{abort}_\tau\ t\end{array}$$

Figure 4.1 shows the typing rules for terms, all of which are obtained in an analogous way to the typing rules for the simply-typed λ-calculus in Figure 2.1.

Since the typing rules are all based on the principle of natural deduction, we obtain so called β-reductions and η-expansions of terms defined as follows:

$$\begin{array}{rcll}
(\lambda x \in \tau.\, t)\ s & \Longrightarrow_\beta & [s/x]t & \\
\mathsf{fst}\ (t, s) & \Longrightarrow_\beta & t & \\
\mathsf{snd}\ (t, s) & \Longrightarrow_\beta & s & \\
\mathsf{case}\ \mathsf{inl}_\sigma\ t\ \mathsf{of}\ \mathsf{inl}\ x.\, s \mid \mathsf{inr}\ y.\, s' & \Longrightarrow_\beta & [t/x]s & \\
\mathsf{case}\ \mathsf{inr}_\tau\ t\ \mathsf{of}\ \mathsf{inl}\ x.\, s \mid \mathsf{inr}\ y.\, s' & \Longrightarrow_\beta & [t/y]s' & \\
t \in \tau\to\sigma & \Longrightarrow_\eta & \lambda x \in \tau.\, t\ x & \quad (x \textit{ is not free in } t)\\
t \in \tau\times\sigma & \Longrightarrow_\eta & \langle \mathbf{fst}\ t, \mathbf{snd}\ t\rangle & \\
t \in \tau+\sigma & \Longrightarrow_\eta & \mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}\ x.\, \mathbf{inl}_\sigma\ x \mid \mathbf{inr}\ y.\, \mathbf{inr}_\tau\ y & \\
t \in \mathsf{unit} & \Longrightarrow_\eta & \langle\rangle & \\
t \in \mathsf{void} & \Longrightarrow_\eta & \mathbf{abort}_{\mathsf{void}}\ t &
\end{array}$$

Here $[s/x]t$, similar to $[N/x]M$, denotes a capture-avoiding substitution of s for x in t. With β-reductions and η-expansions available, these terms constitute a general language of their own.

4.2 Natural deduction for datatypes

We now consider two concrete datatypes bool for boolean values and nat for natural numbers. Again we explain the meaning of judgments $t \in \mathsf{bool}$ and $t \in \mathsf{nat}$ using the principle of natural deduction. For example, an introduction rule for bool specifies how to deduce a new judgment $t \in \mathsf{bool}$ whereas an elimination rule for bool specifies how to exploit an existing judgment $t \in \mathsf{bool}$. Usually we first *design* introduction rules according to the intuition behind a given datatype and then *derive* elimination rules from these introduction rules. In fact, elimination rules for a datatype can be automatically derived from its introduction rules as long as terms of the datatype are defined inductively.

Let us consider datatype bool for boolean values:

$$\text{datatype} \quad \tau \quad ::= \quad \cdots \mid \mathsf{bool}$$

As a boolean value allows us to choose one of two different options, we associate with datatype bool two terms, **true** and **false**, indicating which option to choose:

$$\frac{}{\mathbf{true} \in \mathsf{bool}}\ \mathsf{boolI_t} \qquad \frac{}{\mathbf{false} \in \mathsf{bool}}\ \mathsf{boolI_f}$$

Suppose now that we have a judgment $t \in \mathsf{bool}$ that we wish to exploit in deducing another judgment. Since it is in general unknown whether t is equivalent to **true** or **false**, we provide for both possibilities using a term matching t with **true** and **false** in turn:

$$\frac{t \in \mathsf{bool} \quad t_1 \in \tau \quad t_2 \in \tau}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{true} \Rightarrow t_1 \mid \mathbf{false} \Rightarrow t_2 \in \tau}\ \mathsf{boolE}$$

Note that although the rule boolE eliminates a term of datatype bool, the term in its conclusion may have a different datatype τ.

We choose to abbreviate $\mathbf{case}\ t\ \mathbf{of}\ \mathbf{true} \Rightarrow t_1 \mid \mathbf{false} \Rightarrow t_2$ as $\mathbf{if}\ t\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2$ familiar from programming languages. Thus we obtain the following definition of terms for datatype bool:

$$\text{term} \quad t \quad ::= \quad \cdots \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\ t\ \mathbf{then}\ t\ \mathbf{else}\ t$$

For datatype bool, we obtain the following β-reductions and η-expansion:

$$\begin{array}{rcl} \mathbf{if}\ \mathbf{true}\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 & \Longrightarrow_\beta & t_1 \\ \mathbf{if}\ \mathbf{false}\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 & \Longrightarrow_\beta & t_2 \\ t \in \mathsf{bool} & \Longrightarrow_\eta & \mathbf{if}\ t\ \mathbf{then}\ \mathbf{true}\ \mathbf{else}\ \mathbf{false} \end{array}$$

Here are a few examples of functions manipulating boolean values. and and or compute the logical conjunction and disjunction, respectively, of two boolean values. not computes the logical negation of a boolean value.

$$\begin{array}{ll} \mathit{and} \in \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} & \mathit{and} = \lambda x \in \mathsf{bool}.\, \lambda y \in \mathsf{bool}.\, \mathbf{if}\ x\ \mathbf{then}\ y\ \mathbf{else}\ \mathbf{false} \\ \mathit{or} \in \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} & \mathit{or} = \lambda x \in \mathsf{bool}.\, \lambda y \in \mathsf{bool}.\, \mathbf{if}\ x\ \mathbf{then}\ \mathbf{true}\ \mathbf{else}\ y \\ \mathit{not} \in \mathsf{bool} \to \mathsf{bool} & \mathit{not} = \lambda x \in \mathsf{bool}.\, \mathbf{if}\ x\ \mathbf{then}\ \mathbf{false}\ \mathbf{else}\ \mathbf{true} \end{array}$$

Datatype nat defines a natural number as either zero $\mathbf{0}$ or a successor $\mathsf{s}(t)$ of another natural number t.

$$\text{datatype} \quad \tau \quad ::= \quad \cdots \mid \mathsf{nat}$$

$$\frac{}{\mathbf{0} \in \mathsf{nat}}\ \mathsf{natI_0} \qquad \frac{t \in \mathsf{nat}}{\mathsf{s}(t) \in \mathsf{nat}}\ \mathsf{natI_s}$$

The elimination rule is similar to the rule boolE and considers two cases for a given term t of datatype nat: when t matches zero and when t matches a successor of another natural number. The difference is that in the second case, the elimination rule binds a term variable, say x, to the predecessor of t which is unknown in general. Thus the elimination rule for datatype nat uses a hypothesis of $x \in \mathsf{nat}$:

$$\frac{t \in \mathsf{nat} \quad t_0 \in \tau \quad \begin{array}{c} \overline{x \in \mathsf{nat}} \\ \vdots \\ t_s \in \tau \end{array}}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{0} \Rightarrow t_0 \mid \mathsf{s}(x) \Rightarrow t_s \in \tau}\ \mathsf{natE}$$

Here x is a local term variable whose scope is restricted to t_s, and we may rename it whenever necessary. Thus we obtain the following definition of terms for datatype nat:

$$\text{term} \quad t \quad ::= \quad \cdots \mid \mathbf{0} \mid \mathsf{s}(t) \mid \mathbf{case}\ t\ \mathbf{of}\ \mathbf{0} \Rightarrow t \mid \mathsf{s}(x) \Rightarrow t$$

We use the following β-reductions and η-expansion for datatype nat:

$$\begin{array}{rcl} \mathbf{case}\ \mathbf{0}\ \mathbf{of}\ \mathbf{0} \Rightarrow t_0 \mid \mathsf{s}(x) \Rightarrow t_s & \Longrightarrow_\beta & t_0 \\ \mathbf{case}\ \mathsf{s}(t)\ \mathbf{of}\ \mathbf{0} \Rightarrow t_0 \mid \mathsf{s}(x) \Rightarrow t_s & \Longrightarrow_\beta & [t/x]t_s \\ t \in \mathsf{nat} & \Longrightarrow_\eta & \mathbf{case}\ t\ \mathbf{of}\ \mathbf{0} \Rightarrow \mathbf{0} \mid \mathsf{s}(x) \Rightarrow \mathsf{s}(x) \end{array}$$

Now we can define various functions returning different results depending on whether a given natural number is zero or not. For example, we define a function returning the predecessor of a given natural number as follows:

$$\mathit{pred} \in \mathsf{nat} \rightarrow \mathsf{nat} \qquad \mathit{pred} \;=\; \lambda x \in \mathsf{nat}.\, \mathbf{case}\ x\ \mathbf{of}\ \mathbf{0} \Rightarrow \mathbf{0} \mid \mathbf{s}(y) \Rightarrow y$$

The extent to which we define such functions is limited, however, because we have no machinery for defining *recursive* functions. For example, the following definition of a function doubling a given natural number is not valid because double in the body of the λ-abstraction is a free term variable whose definition is still incomplete:

$$\mathit{double} \in \mathsf{nat} \rightarrow \mathsf{nat} \qquad \mathit{double} \;=\; \lambda x \in \mathsf{nat}.\, \mathbf{case}\ x\ \mathbf{of}\ \mathbf{0} \Rightarrow \mathbf{0} \mid \mathbf{s}(y) \Rightarrow \mathbf{s}(\mathbf{s}(\mathit{double}\ y))$$

In the next section, we revise the rule natE so that we can define recursive functions over natural numbers. Instead of a general form of recursion, we base the rule natE on *primitive recursion* which guarantees that every recursive call eventually terminates. Not every recursive function is definable with primitive recursion (*e.g.*, the Ackermann function), but in the study of first-order logic with datatypes, we seldom need such recursive functions.

4.3 Primitive recursion

The revised rule natE based on primitive recursion is another elimination rule for datatype nat:

$$\dfrac{t \in \mathsf{nat} \quad t_0 \in \tau \quad \begin{matrix} \overline{x \in \mathsf{nat}} \quad \overline{f(x) \in \tau} \\ \vdots \\ t_s \in \tau \end{matrix}}{\mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t_0 \mid f(\mathbf{s}(x)) \Rightarrow t_s \in \tau}\ \mathsf{natE}$$

We may think of $\mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t_0 \mid f(\mathbf{s}(x)) \Rightarrow t_s$ as a primitive recursive function f applied to t. In the base case where t is $\mathbf{0}$, we take t_0. Hence t_0 is not permitted to make a recursive call to f. In the recursive case where t matches $\mathbf{s}(x)$, we take t_s. Inside t_s, we use x to denote the predecessor of t and $f(x)$ to denote a recursive call to f. Sometimes we write $\mathbf{rec}\ f(t)\ \mathbf{of} \begin{cases} f(\mathbf{0}) \Rightarrow t_0 \\ f(\mathbf{s}(x)) \Rightarrow t_s \end{cases}$ for visual clarity.

It is important that t_s may contain a recursive call to f, but only with x as its argument. For example, such terms as $f(\mathit{pred}\ x)$ and $f(\mathbf{s}(\mathbf{0}))$ are disallowed in t_s. This syntactic restriction is the characteristic feature of primitive recursion which prohibits an infinite sequence of recursive calls and thus guarantees that a primitive recursive function always terminates regardless of its actual argument. Since every recursive call to f within t_s always takes x as its argument, we regard $f(x)$ as not an application but a term variable in itself.

Note that if t_s contains no recursive call to f, the whole term $\mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t_0 \mid f(\mathbf{s}(x)) \Rightarrow t_s$ simplifies to $\mathbf{case}\ t\ \mathbf{of}\ \mathbf{0} \Rightarrow t_0 \mid \mathbf{s}(x) \Rightarrow t_s$. Since the previous rule natE is a special case of the revised rule natE, we revise the definition of terms for datatype nat as follows:

$$\text{term} \quad t \quad ::= \quad \cdots \mid \mathbf{0} \mid \mathbf{s}(t) \mid \mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t \mid f(\mathbf{s}(x)) \Rightarrow t$$

The β-reductions and η-expansion for datatype nat are given as follows:

$$\begin{array}{rcl} \mathbf{rec}\ f(\mathbf{0})\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t_0 \mid f(\mathbf{s}(x)) \Rightarrow t_s & \Longrightarrow_\beta & t_0 \\ \mathbf{rec}\ f(\mathbf{s}(t))\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t_0 \mid f(\mathbf{s}(x)) \Rightarrow t_s & \Longrightarrow_\beta & [\mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t_0 \mid f(\mathbf{s}(x)) \Rightarrow t_s/f(x)][t/x]t_s \\ t \in \mathsf{nat} & \Longrightarrow_\eta & \mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow \mathbf{0} \mid f(\mathbf{s}(x)) \Rightarrow \mathbf{s}(x) \end{array}$$

In the first β-reduction, the left term reduces to t_0 because the argument to f is $\mathbf{0}$. In the second β-reduction where the argument $\mathbf{s}(t)$ matches $\mathbf{s}(x)$, we apply an equality $x = t$ throughout t_s by replacing x by t and $f(x)$ by a term denoting a call to f with an argument of t, namely $\mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t_0 \mid f(\mathbf{s}(x)) \Rightarrow t_s$. The η-expansion does not involve a recursive call.

Now we can define a wide range of recursive functions. For example, we specify a function $double$ doubling a given natural number as follows:

$$\begin{array}{rcl} double\ \mathbf{0} & = & \mathbf{0} \\ double\ \mathbf{s}(x) & = & \mathbf{s}(\mathbf{s}(double\ x)) \end{array}$$

The above specification translates to the following definition:

$$\begin{array}{l} double \in \mathsf{nat} \to \mathsf{nat} \\ double\ =\ \lambda x \in \mathsf{nat}.\, \mathbf{rec}\ d(x)\ \mathbf{of}\ d(\mathbf{0}) \Rightarrow \mathbf{0} \mid d(\mathbf{s}(y)) \Rightarrow \mathbf{s}(\mathbf{s}(d(y))) \end{array}$$

The following sequence of β-reductions (which are applied to subterms when necessary) shows that $double\ \mathbf{s}(\mathbf{0})$ reduces to $\mathbf{s}(\mathbf{s}(\mathbf{0}))$:

$$\begin{array}{rcl} double\ \mathbf{s}(\mathbf{0}) & \Longrightarrow_\beta & \mathbf{rec}\ d(\mathbf{s}(\mathbf{0}))\ \mathbf{of}\ d(\mathbf{0}) \Rightarrow \mathbf{0} \mid d(\mathbf{s}(y)) \Rightarrow \mathbf{s}(\mathbf{s}(d(y))) \\ & \Longrightarrow_\beta & \mathbf{s}(\mathbf{s}(\mathbf{rec}\ d(\mathbf{0})\ \mathbf{of}\ d(\mathbf{0}) \Rightarrow \mathbf{0} \mid d(\mathbf{s}(y)) \Rightarrow \mathbf{s}(\mathbf{s}(d(y))))) \\ & \Longrightarrow_\beta & \mathbf{s}(\mathbf{s}(\mathbf{0})) \end{array}$$

As another example, we specify and define a function $plus$ adding two natural numbers as follows:

$$\begin{array}{rcl} plus\ \mathbf{0}\ y & = & y \\ plus\ \mathbf{s}(x)\ y & = & \mathbf{s}(plus\ x\ y) \end{array}$$

$$\begin{array}{l} plus \in \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat} \\ plus\ =\ \lambda x \in \mathsf{nat}.\, \lambda y \in \mathsf{nat}.\, \mathbf{rec}\ p(x)\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow y \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)) \end{array}$$

The following sequence of β-reductions shows that $\mathbf{s}(\mathbf{0})$ and t add to $\mathbf{s}(t)$:

$$\begin{array}{rcl} plus\ \mathbf{s}(\mathbf{0})\ t & \Longrightarrow_\beta & (\lambda y \in \mathsf{nat}.\, \mathbf{rec}\ p(\mathbf{s}(\mathbf{0}))\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow y \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)))\ t \\ & \Longrightarrow_\beta & \mathbf{rec}\ p(\mathbf{s}(\mathbf{0}))\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow t \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)) \\ & \Longrightarrow_\beta & \mathbf{s}(\mathbf{rec}\ p(\mathbf{0})\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow t \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z))) \\ & \Longrightarrow_\beta & \mathbf{s}(t) \end{array}$$

Alternatively we may define $plus$ in such a way that it recurses over the first argument x before taking the second argument y:

$$plus\ =\ \lambda x \in \mathsf{nat}.\, \mathbf{rec}\ p(x)\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow \lambda y \in \mathsf{nat}.\, y \mid p(\mathbf{s}(z)) \Rightarrow \lambda y \in \mathsf{nat}.\, \mathbf{s}(p(z)\ y)$$

In this case, the reduction of $plus\ \mathbf{s}(\mathbf{0})\ t$ requires one more step:

$$\begin{array}{rcl} plus\ \mathbf{s}(\mathbf{0})\ t & \Longrightarrow_\beta & \left(\mathbf{rec}\ p(\mathbf{s}(\mathbf{0}))\ \mathbf{of}\ \left\{\begin{array}{l} p(\mathbf{0}) \Rightarrow \lambda y \in \mathsf{nat}.\, y \\ p(\mathbf{s}(z)) \Rightarrow \lambda y \in \mathsf{nat}.\, \mathbf{s}(p(z)\ y) \end{array}\right.\right)\ t \\ & \Longrightarrow_\beta & \lambda y \in \mathsf{nat}.\, \mathbf{s}(\left(\mathbf{rec}\ p(\mathbf{0})\ \mathbf{of}\ \left\{\begin{array}{l} p(\mathbf{0}) \Rightarrow \lambda y \in \mathsf{nat}.\, y \\ p(\mathbf{s}(z)) \Rightarrow \lambda y \in \mathsf{nat}.\, \mathbf{s}(p(z)\ y) \end{array}\right.\right)\ y)\ t \\ & \Longrightarrow_\beta & \mathbf{s}(\left(\mathbf{rec}\ p(\mathbf{0})\ \mathbf{of}\ \left\{\begin{array}{l} p(\mathbf{0}) \Rightarrow \lambda y \in \mathsf{nat}.\, y \\ p(\mathbf{s}(z)) \Rightarrow \lambda y \in \mathsf{nat}.\, \mathbf{s}(p(z)\ y) \end{array}\right.\right)\ t) \\ & \Longrightarrow_\beta & \mathbf{s}((\lambda y \in \mathsf{nat}.\, y)\ t) \\ & \Longrightarrow_\beta & \mathbf{s}(t) \end{array}$$

It is important to note that we have *derived*, as opposed to *designed*, the rule natE from the inductive definition of terms for datatype nat. In general, once we design introduction rules for a datatype so as to obtain an inductive definition of terms, the principle of primitive recursion determines a unique elimination rule. Below we consider another example of a datatype in order to further elucidate the process of deriving an elimination rule from introduction rules.

We use $\mathsf{list}\ \tau$ as a datatype for lists of terms of datatype τ:

$$\text{datatype} \quad \tau \quad ::= \quad \cdots \mid \mathsf{list}\ \tau$$

We use $\mathbf{nil}^\tau$ for an empty list of datatype $\mathsf{list}\ \tau$ and $t :: s$ for a list consisting of a head element t and a tail list s:

$$\frac{}{\mathbf{nil}^\tau \in \mathsf{list}\ \tau}\ \mathsf{listI_n} \qquad \frac{t \in \tau \quad s \in \mathsf{list}\ \tau}{t :: s \in \mathsf{list}\ \tau}\ \mathsf{listI_c}$$

From these introduction rules, we derive the following elimination rule based on primitive recursion:

$$\dfrac{t \in \mathsf{list}\,\tau \quad s_n \in \sigma \quad \begin{array}{c}\overline{x \in \tau} \quad \overline{l \in \mathsf{list}\,\tau} \quad \overline{f(l) \in \sigma} \\ \vdots \\ s_c \in \sigma\end{array}}{\mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{nil}) \Rightarrow s_n \mid f(x :: l) \Rightarrow s_c \in \sigma}\ \mathsf{list}\mathsf{E}$$

The first branch, corresponding to the rule $\mathsf{list}\mathsf{I}_\mathsf{n}$, uses no term variable because $\mathbf{nil}^\tau$ has no subterm. In the second branch, we use two term variables x and l because the rule $\mathsf{list}\mathsf{I}_\mathsf{c}$ uses two terms in its premise. As in the rule $\mathsf{nat}\mathsf{E}$, we treat $f(l)$ as a term variable. Thus we obtain the following definition of terms for datatype $\mathsf{list}\,\tau$:

$$\text{term} \quad t \quad ::= \quad \cdots \mid \mathbf{nil}^\tau \mid t :: t \mid \mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{nil}) \Rightarrow t \mid f(x :: x) \Rightarrow t$$

The derivation of the β-reductions and η-expansion is also similar to the case of datatype nat:

$$\begin{array}{rcl}
\mathbf{rec}\ f(\mathbf{nil}^\tau)\ \mathbf{of}\ f(\mathbf{nil}) \Rightarrow s_n \mid f(x :: l) \Rightarrow s_c & \Longrightarrow_\beta & s_n \\
\mathbf{rec}\ f(t :: t')\ \mathbf{of}\ f(\mathbf{nil}) \Rightarrow s_n \mid f(x :: l) \Rightarrow s_c & \Longrightarrow_\beta & [\mathbf{rec}\ f(t')\ \mathbf{of}\ f(\mathbf{nil}) \Rightarrow s_n \mid f(x :: l) \Rightarrow s_c/f(l)][t'/l][t/x]s_c \\
t \in \mathsf{list}\,\tau & \Longrightarrow_\eta & \mathbf{rec}\ f(t)\ \mathbf{of}\ f(\mathbf{nil}) \Rightarrow \mathbf{nil}^\tau \mid f(x :: l) \Rightarrow x :: l
\end{array}$$

As examples of recursive functions over lists, we define a function *append* concatenating two lists and another function *length* calculating the length of a list:

$$\begin{array}{rcl}
\mathit{append}\ \mathbf{nil}^\tau\ t & = & t \\
\mathit{append}\ (x :: l)\ t & = & x :: (\mathit{append}\ l\ t)
\end{array}$$

$$\begin{array}{l}
\mathit{append} \in \mathsf{list}\,\tau \rightarrow \mathsf{list}\,\tau \rightarrow \mathsf{list}\,\tau \\
\mathit{append}\ =\ \lambda y \in \mathsf{list}\,\tau.\, \lambda z \in \mathsf{list}\,\tau.\, \mathbf{rec}\ f(y)\ \mathbf{of}\ f(\mathbf{nil}) \Rightarrow z \mid f(x :: l) \Rightarrow x :: f(l)
\end{array}$$

$$\begin{array}{rcl}
\mathit{length}\ \mathbf{nil}^\tau & = & \mathbf{0} \\
\mathit{length}\ (x :: l) & = & \mathsf{s}(\mathit{length}\ l)
\end{array}$$

$$\begin{array}{l}
\mathit{length} \in \mathsf{list}\,\tau \rightarrow \mathsf{nat} \\
\mathit{length}\ =\ \lambda y \in \mathsf{list}\,\tau.\, \mathbf{rec}\ f(y)\ \mathbf{of}\ f(\mathbf{nil}) \Rightarrow \mathbf{0} \mid f(x :: l) \Rightarrow \mathsf{s}(f(l))
\end{array}$$

4.4 First-order logic with datatypes

So far, we have designed a system for creating terms and specifying their datatypes. As our study focuses on logic rather than programming languages, we are ultimately interested in proving properties of terms rather than in manipulating terms. For example, the goal of designing a system for natural numbers is not to demonstrate how to multiply two natural numbers, but to formally prove such properties as that every non-zero natural number is a product of two prime numbers.

In order to state, let alone prove, interesting properties of terms, we need appropriate predicates. For example, we may need a predicate $LT(m, n)$ to state that a natural number m is less than another natural number n. We also need universal and existential quantifications so as to express that a property holds for all terms of a specific datatype, or that there exists a certain term satisfying a given property. For example, we may use $\forall x \in \mathsf{nat}. LT(x, \mathsf{s}(x))$ to state that for every natural number is less than its successor, or $\exists x \in \mathsf{nat}. LT(x, \mathsf{s}(\mathbf{0}))$ to state that there exists a natural number less than one. Then constructing proofs of judgments $\forall x \in \mathsf{nat}. LT(x, \mathsf{s}(x))$ *true* and $\exists x \in \mathsf{nat}. LT(x, \mathsf{s}(\mathbf{0}))$ *true* amounts to formally proving these properties.

Before we investigate how to define predicates, we consider universal and existential quantifications in the presence of datatypes. The development is similar to the case of pure first-order logic except that we now specify the datatype for each term variable.

The inductive definition of propositions is extended as follows:

$$\text{proposition} \quad A \quad ::= \quad \cdots \mid \forall x \in \tau. A(x) \mid \exists x \in \tau. A(x)$$

$$\dfrac{\begin{array}{c}\overline{x\in\tau}\\ \vdots\\ A(x)\ \mathit{true}\end{array}}{\forall x\in\tau.A(x)\ \mathit{true}}\ \forall\mathsf{I} \qquad \dfrac{\forall x\in\tau.A(x)\ \mathit{true} \quad t\in\tau}{A(t)\ \mathit{true}}\ \forall\mathsf{E}$$

$$\dfrac{t\in\tau \quad A(t)\ \mathit{true}}{\exists x\in\tau.A(x)\ \mathit{true}}\ \exists\mathsf{I} \qquad \dfrac{\exists x\in\tau.A(x)\ \mathit{true} \qquad \begin{array}{c}\overline{x\in\tau}\quad \overline{A(x)\ \mathit{true}}^{\,w}\\ \vdots\\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\ \exists\mathsf{E}^{w}$$

Figure 4.2: Natural deduction system for first-order logic with datatypes

$$\dfrac{\begin{array}{c}\overline{x\in\tau}\\ \vdots\\ M : A(x)\end{array}}{\lambda x\in\tau.\,M : \forall x\in\tau.A(x)}\ \forall\mathsf{I} \qquad \dfrac{M : \forall x\in\tau.A(x) \quad t\in\tau}{M\ t : A(t)}\ \forall\mathsf{E}$$

$$\dfrac{t\in\tau \quad M : A(t)}{\langle t, M\rangle : \exists x\in\tau.A(x)}\ \exists\mathsf{I} \qquad \dfrac{M : \exists x\in\tau.A(x) \qquad \begin{array}{c}\overline{x\in\tau}\quad \overline{w : A(x)}\\ \vdots\\ N : C\end{array}}{\mathsf{let}\ \langle x, w\rangle = M\ \mathsf{in}\ N : C}\ \exists\mathsf{E}$$

Figure 4.3: Typing rules for proof terms in first-order logic with datatypes

The formation rules for $\forall x\in\tau.A$ and $\exists x\in\tau.A$ use a hypothesis of $x\in\tau$:

$$\dfrac{\begin{array}{c}\overline{x\in\tau}\\ \vdots\\ A(x)\ \mathit{prop}\end{array}}{\forall x\in\tau.A(x)\ \mathit{prop}}\ \forall\mathsf{F} \quad \dfrac{\begin{array}{c}\overline{x\in\tau}\\ \vdots\\ A(x)\ \mathit{prop}\end{array}}{\exists x\in\tau.A(x)\ \mathit{prop}}\ \exists\mathsf{F}$$

As we may freely rename x in $\forall x\in\tau.A(x)$ and $\exists x\in\tau.A(x)$ to a different term variable, we assume that term variables declared in universal and existential quantifications are all distinct.

Figure 4.2 shows the inference rules for first-order logic with datatypes. These inference rules have two important differences from those in pure first-order logic. First the rules $\forall$I and $\exists$E no longer replace a term variable x by a fresh parameter a as in $[a/x]A\ \mathit{true}$, but use a hypothesis of $x\in\tau$ specifying the datatype for x. Second the rules $\forall$E and $\exists$I require a separate judgment $t\in\tau$.

The rule $\forall$I resembles the rule $\supset$I in that it uses a hypothesis whose scope is local to the premise. The difference from the rule $\supset$I is that the meaning of proposition $A(x)$ in the premise is dependent on the meaning of term variable x in the hypothesis. In contrast, in the rule $\supset$I for proving $A \supset B\ \mathit{true}$, the meaning of proposition B is independent of the meaning of proposition A. Hence, if we collapse the distinction between types and datatypes and use $x\in A$ instead of $x : A$, an implication $A\supset B$ becomes a special case of a universal quantification $\forall x\in A.B$ where B contains no occurrence of x. Then the rule $\supset$E also becomes a special case of the rule $\forall$E.

Proof terms are the same as in pure first-order logic except that we use λ-abstractions that explicitly specify the datatype of term variables:

$$\text{proof term} \quad M \quad ::= \quad \cdots \mid \lambda x\in\tau.\,M \mid M\ t \mid \langle t, M\rangle \mid \mathsf{let}\ \langle x, w\rangle = M\ \mathsf{in}\ M$$

Figure 4.3 shows the typing rules for these proof terms. The typing rules $\forall$I and $\forall$E show that it is no coincidence that we use λ-abstractions and λ-applications as proofs terms for universal quantifications as well as for implications, since implications are essentially a special case of universal quantifications.

Here are a few examples of proof terms whose types involve universal or existential quantifications:

- A proof term of type $(\forall x \in \tau. A(x) \wedge B(x)) \supset \forall x \in \tau. A(x)$ is

  $$\lambda z : \forall x \in \tau. A(x) \wedge B(x).\, \lambda x \in \tau.\, \mathsf{fst}\ (z\ x).$$

  Note that z is a variable whereas x is a term variable.

- A proof term of type $(\exists x \in \tau. A(x) \vee B(x)) \supset ((\exists x \in \tau. A(x)) \vee (\exists x \in \tau. B(x)))$ is

  $$\lambda z : \exists x \in \tau. A(x) \vee B(x).\, \mathsf{let}\ \langle x, w\rangle = z\ \mathsf{in}\ \mathsf{case}\ w\ \mathsf{of}\ \mathsf{inl}\ y_1.\, \mathsf{inl}_{\exists x \in \tau. B(x)}\ \langle x, y_1\rangle \mid \mathsf{inr}\ y_2.\, \mathsf{inr}_{\exists x \in \tau. A(x)}\ \langle x, y_2\rangle.$$

- A proof term of type $((\exists x \in \tau. A(x)) \vee (\exists x \in \tau. B(x))) \supset (\exists x \in \tau. A(x) \vee B(x))$ is

  $$\begin{array}{l} \lambda z : (\exists x \in \tau. A(x)) \vee (\exists x \in \tau. B(x)). \\ \quad \mathsf{case}\ z\ \mathsf{of}\ \mathsf{inl}\ y_1.\, \mathsf{let}\ \langle x, w\rangle = y_1\ \mathsf{in}\ \langle x, \mathsf{inl}_{B(x)}\ w\rangle \mid \mathsf{inr}\ y_2.\, \mathsf{let}\ \langle x, w\rangle = y_2\ \mathsf{in}\ \langle x, \mathsf{inr}_{A(x)}\ w\rangle \end{array}.$$

The β-reduction and η-expansion for universal and existential quantifications are given as follows:

$$\begin{array}{rcll} (\lambda x \in \tau.\, M)\ t & \Longrightarrow_\beta & [t/x]M & \\ M : \forall x \in \tau. A(x) & \Longrightarrow_\eta & \lambda x \in \tau.\, M\ x & (x \textit{ is not free in } M) \end{array}$$

$$\begin{array}{rcl} \mathsf{let}\ \langle x, w\rangle = \langle t, M\rangle\ \mathsf{in}\ N & \Longrightarrow_\beta & [M/w][t/x]N \\ M : \exists x \in \tau. A(x) & \Longrightarrow_\eta & \mathsf{let}\ \langle x, w\rangle = M\ \mathsf{in}\ \langle x, w\rangle \end{array}$$

4.5 Natural deduction for predicates

We now investigate how to define predicates on terms. As with all the logical connectives in first-order logic, we base the definition of every predicate on the principle of natural deduction. That is, we use an introduction rule to deduce a new judgment involving the predicate, and an elimination rule to exploit an existing judgment involving the predicate. Moreover, as in the definition of such datatypes as bool and nat, we first *design* introduction rules to characterize the predicate and then *derive* elimination rules from these introduction rules.

As a running example, we define a predicate $LT(m, n)$ to mean a natural number m is less than another natural number n. We abbreviate $LT(m, n)$ as $m < n$.

$$\text{proposition} \quad A \quad ::= \quad \cdots \mid m < n$$

The formation rule for $m < n$ requires both m and n to be of datatype nat:

$$\frac{m \in \mathsf{nat} \quad n \in \mathsf{nat}}{m < n\ \mathit{prop}} <\!\mathsf{F}$$

By the rule <F, every judgment $m < n\ \mathit{true}$ implicitly assumes that both m and n are of datatype nat.

We use the following introduction rules:

$$\frac{}{\mathbf{0} < \mathsf{s}(n)\ \mathit{true}} <\!\mathsf{I}_0 \qquad \frac{m < n\ \mathit{true}}{\mathsf{s}(m) < \mathsf{s}(n)\ \mathit{true}} <\!\mathsf{I}_\mathsf{s}$$

The rule $<\mathsf{I}_0$ states that $\mathbf{0}$ is less than the successor of any natural number; the rule $<\mathsf{I}_\mathsf{s}$ states that proving $\mathsf{s}(m) < \mathsf{s}(n)\ \mathit{true}$ reduces to proving $m < n\ \mathit{true}$. Now the two introduction rules determine a unique meaning for $<$ which is a comparison relation applicable to any pair of natural numbers. If we choose to include the rule $<\mathsf{I}_0$ but omit $<\mathsf{I}_\mathsf{s}$, we obtain a different but still valid meaning for $<$ which is a relation testing whether a given natural number is greater than zero or not. Thus the two introduction rules provide just a specific way to characterize $<$, which can be defined in many different ways.

In order to derive elimination rules, we consider four possible cases of the judgment $m < n\ \mathit{true}$:

- $\mathbf{0} < \mathbf{0}\ \mathit{true}$ is impossible to prove. The corresponding elimination rule may deduce any judgment $C\ \mathit{true}$.

- $\mathsf{s}(m) < \mathbf{0}\ \mathit{true}$ is impossible to prove. The corresponding elimination rule may deduce any judgment $C\ \mathit{true}$.
- $\mathbf{0} < \mathsf{s}(n)\ \mathit{true}$ holds trivially by the rule $<\mathsf{I}_0$ whose premise is empty. Hence there is no corresponding elimination rule.
- $\mathsf{s}(m) < \mathsf{s}(n)\ \mathit{true}$ holds by the rule $<\mathsf{I}_\mathsf{s}$ whose premise is $m < n\ \mathit{true}$. Thus the corresponding elimination rule deduces $m < n\ \mathit{true}$.

We combine the first two cases to obtain a single elimination rule:

$$\frac{m < \mathbf{0}\ \mathit{true}}{C\ \mathit{true}}\ {<}\mathsf{E}_0 \qquad \frac{\mathsf{s}(m) < \mathsf{s}(n)\ \mathit{true}}{m < n\ \mathit{true}}\ {<}\mathsf{E}_\mathsf{s}$$

Note that the rules $<\mathsf{I}_\mathsf{s}$ and $<\mathsf{E}_\mathsf{s}$, which have $<$ in both the conclusion and the premise, do not destroy the orthogonality of the system because $<$ is not a logical connective but a predicate. If $<$ were a logical connective, we lose the orthogonality of the system because we explain the meaning of $<$ using $<$ itself.

Here are two examples of proofs using the rules for $<$:

$$\frac{\dfrac{}{\mathbf{0} < \mathsf{s}(\mathbf{0})\ \mathit{true}}\ {<}\mathsf{I}_0}{\mathsf{s}(\mathbf{0}) < \mathsf{s}(\mathsf{s}(\mathbf{0}))\ \mathit{true}}\ {<}\mathsf{I}_\mathsf{s} \qquad \frac{\dfrac{\overline{m < \mathbf{0}\ \mathit{true}}^{\,z}}{\bot\ \mathit{true}}\ {<}\mathsf{E}_0}{\neg(m < \mathbf{0})\ \mathit{true}}\ \neg\mathsf{I}^z$$

We can also represent a proof of $m < n\ \mathit{true}$ as a proof term of type $m < n$. Note that we refer to $m < n$ as a "type," which is nothing strange because propositions and types are equivalent under the Curry-Howard isomorphism. We use the following definition of proof terms each of which corresponds to an inference rule as shown below:

$$\text{proof term} \quad M \quad ::= \quad \cdots \mid \mathsf{ltI}_0 \mid \mathsf{ltI}_\mathsf{s}(M) \mid \mathsf{ltE}_0(M) \mid \mathsf{ltE}_\mathsf{s}(M)$$

$$\frac{}{\mathsf{ltI}_0 : \mathbf{0} < \mathsf{s}(n)}\ {<}\mathsf{I}_0 \qquad \frac{M : m < n}{\mathsf{ltI}_\mathsf{s}(M) : \mathsf{s}(m) < \mathsf{s}(n)}\ {<}\mathsf{I}_\mathsf{s} \qquad \frac{M : m < \mathbf{0}}{\mathsf{ltE}_0(M) : C}\ {<}\mathsf{E}_0 \qquad \frac{M : \mathsf{s}(m) < \mathsf{s}(n)}{\mathsf{ltE}_\mathsf{s}(M) : m < n}\ {<}\mathsf{E}_\mathsf{s}$$

Here are proof terms of types $\mathsf{s}(\mathbf{0}) < \mathsf{s}(\mathsf{s}(\mathbf{0}))$ and $\neg(m < \mathbf{0})$:

$$\frac{\dfrac{}{\mathsf{ltI}_0 : \mathbf{0} < \mathsf{s}(\mathbf{0})}\ {<}\mathsf{I}_0}{\mathsf{ltI}_\mathsf{s}(\mathsf{ltI}_0) : \mathsf{s}(\mathbf{0}) < \mathsf{s}(\mathsf{s}(\mathbf{0}))}\ {<}\mathsf{I}_\mathsf{s} \qquad \frac{\dfrac{\overline{z : m < \mathbf{0}}^{\,z}}{\mathsf{ltE}_0(z) : \bot}\ {<}\mathsf{E}_0}{\lambda z{:}m < \mathbf{0}.\, \mathsf{ltE}_0(z) : \neg(m < \mathbf{0})}\ \supset\!\mathsf{I}$$

As another example, we consider a predicate $EQ(m, n)$ to mean that natural numbers m and n are equal. We abbreviate $EQ(m, n)$ as $m =_\mathsf{N} n$.

$$\text{proposition} \quad A \quad ::= \quad \cdots \mid m =_\mathsf{N} n$$

$$\frac{m \in \mathsf{nat} \quad n \in \mathsf{nat}}{m =_\mathsf{N} n\ \mathit{prop}}\ {=_\mathsf{N}}\mathsf{F}$$

Note that $m =_\mathsf{N} n$, which says that m and n represent the same natural number, is different from $m = n$, which says that m and n are syntactically identical.

Similarly to the predicate $m < n$, we use two introduction rules:

$$\frac{}{\mathbf{0} =_\mathsf{N} \mathbf{0}\ \mathit{true}}\ {=_\mathsf{N}}\mathsf{I}_0 \qquad \frac{m =_\mathsf{N} n\ \mathit{true}}{\mathsf{s}(m) =_\mathsf{N} \mathsf{s}(n)\ \mathit{true}}\ {=_\mathsf{N}}\mathsf{I}_\mathsf{s}$$

From these introduction rules, we derive the following elimination rules:

$$\frac{\mathbf{0} =_\mathsf{N} \mathsf{s}(n)\ \mathit{true}}{C\ \mathit{true}}\ {=_\mathsf{N}}\mathsf{E}_{0\mathsf{s}} \qquad \frac{\mathsf{s}(m) =_\mathsf{N} \mathbf{0}\ \mathit{true}}{C\ \mathit{true}}\ {=_\mathsf{N}}\mathsf{E}_{\mathsf{s}0} \qquad \frac{\mathsf{s}(m) =_\mathsf{N} \mathsf{s}(n)\ \mathit{true}}{m =_\mathsf{N} n\ \mathit{true}}\ {=_\mathsf{N}}\mathsf{E}_\mathsf{s}$$

There is no elimination rule for $\mathbf{0} =_\mathsf{N} \mathbf{0}\ \mathit{true}$ because the premise of the rule ${=_\mathsf{N}}\mathsf{I}_0$ is empty.

We use the following definition of proof terms for the predicate $m =_{\mathsf{N}} n$:

$$\text{proof term} \quad M \quad ::= \quad \cdots \mid \mathsf{eqI}_0 \mid \mathsf{eqI}_{\mathsf{s}}(M) \mid \mathsf{eqE}_{0\mathsf{s}}(M) \mid \mathsf{eqE}_{\mathsf{s}0}(M) \mid \mathsf{eqE}_{\mathsf{s}}(M)$$

$$\frac{}{\mathsf{eqI}_0 : \mathbf{0} =_{\mathsf{N}} \mathbf{0}} =_{\mathsf{N}}\mathsf{I}_0 \qquad \frac{M : m =_{\mathsf{N}} n}{\mathsf{eqI}_{\mathsf{s}}(M) : \mathbf{s}(m) =_{\mathsf{N}} \mathbf{s}(n)} =_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}$$

$$\frac{M : \mathbf{0} =_{\mathsf{N}} \mathbf{s}(n)}{\mathsf{eqE}_{0\mathsf{s}}(M) : C} =_{\mathsf{N}}\mathsf{E}_{0\mathsf{s}} \qquad \frac{M : \mathbf{s}(m) =_{\mathsf{N}} \mathbf{0}}{\mathsf{eqE}_{\mathsf{s}0}(M) : C} =_{\mathsf{N}}\mathsf{E}_{\mathsf{s}0} \qquad \frac{M : \mathbf{s}(m) =_{\mathsf{N}} \mathbf{s}(n)}{\mathsf{eqE}_{\mathsf{s}}(M) : m =_{\mathsf{N}} n} =_{\mathsf{N}}\mathsf{E}_{\mathsf{s}}$$

Now that we have a couple of predicates, we may attempt to prove interesting properties of natural numbers in conjunction with universal and existential quantifications. For example, we may attempt to prove that for any natural number x, there exists a natural number y such that $x < y$, for example, by choosing $y = \mathbf{s}(x)$. A formal proof of $\forall x \in \mathsf{nat}. \exists y \in \mathsf{nat}. x < y \ \mathit{true}$, however, is not so simple:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{x \in \mathsf{nat}}}{\mathbf{s}(x) \in \mathsf{nat}}\mathsf{natI}_{\mathsf{s}} \qquad \cfrac{???}{x < \mathbf{s}(x)\ \mathit{true}}}{\exists y \in \mathsf{nat}. x < y \ \mathit{true}}}{\forall x \in \mathsf{nat}. \exists y \in \mathsf{nat}. x < y \ \mathit{true}} \forall\mathsf{I}$$

In fact, we cannot prove even such a simple judgment $\forall x \in \mathsf{nat}. x =_{\mathsf{N}} x$. Intuitively we have to prove an infinite number of judgments $\mathbf{0} =_{\mathsf{N}} \mathbf{0}$, $\mathbf{s}(\mathbf{0}) =_{\mathsf{N}} \mathbf{s}(\mathbf{0})$, $\mathbf{s}(\mathbf{s}(\mathbf{0})) =_{\mathsf{N}} \mathbf{s}(\mathbf{s}(\mathbf{0}))$, and so on, but we do not have a mechanism by which we represent all these proofs as a single proof of finite size.

In the next section, we introduce yet another form of elimination rule for datatypes which provides such a mechanism.

4.6 Induction on terms

Suppose that we wish to prove $A(x)\ \mathit{true}$ for every boolean value x. Since there are only two terms $\mathbf{true}$ and $\mathbf{false}$, it suffices to prove $A(\mathbf{true})\ \mathit{true}$ and $A(\mathbf{false})\ \mathit{true}$ separately, which is expressed in the following elimination rule for datatype bool:

$$\frac{t \in \mathsf{bool} \quad A(\mathbf{true})\ \mathit{true} \quad A(\mathbf{false})\ \mathit{true}}{A(t)\ \mathit{true}} \mathsf{boolE}_I$$

Note that unlike the previous elimination rule boolE which deduces only a judgment of the form $s \in \tau$, the rule boolE_I exploits a proof of $t \in \mathsf{bool}$ to deduce a judgment $A(t)\ \mathit{true}$ where $A(t)$ can be any proposition involving t. Thus we have derived a new form of elimination rule which connects different forms of judgments.

Now suppose that we wish to prove $A(x)\ \mathit{true}$ for every natural number x. Since there are an infinite sequence of natural numbers, a naive approach similar to the case of datatype bool would be clearly infeasible:

$$\frac{t \in \mathsf{nat} \quad A(\mathbf{0})\ \mathit{true} \quad A(\mathbf{s}(\mathbf{0}))\ \mathit{true} \quad A(\mathbf{s}(\mathbf{s}(\mathbf{0})))\ \mathit{true} \quad \cdots}{A(t)\ \mathit{true}} \mathsf{natE}_I$$

Thus we are led to derive an elimination rule that allows mathematical induction on natural numbers inside a proof. Specifically it needs to show that $A(\mathbf{0})\ \mathit{true}$ holds and that an induction hypothesis $A(x)\ \mathit{true}$ implies $A(\mathbf{s}(x))\ \mathit{true}$:

$$\frac{t \in \mathsf{nat} \quad A(\mathbf{0})\ \mathit{true} \quad \begin{array}{c} \cfrac{}{x \in \mathsf{nat}} \quad \cfrac{}{A(x)\ \mathit{true}}u(x) \\ \vdots \\ A(\mathbf{s}(x))\ \mathit{true} \end{array}}{A(t)\ \mathit{true}} \mathsf{natE}_I^{u(x)}$$

The second premise states that $A(x)\ \mathit{true}$ holds for $x = \mathbf{0}$, and corresponds to the base case in mathematical induction. The third premise states that a hypothesis of $A(x)\ \mathit{true}$ (with label $u(x)$) leads to a proof of $A(\mathsf{s}(x))$ *true*, and corresponds to the inductive case in mathematical induction. Hence the second and third premises constitute a valid proof of $A(x)$ *true* for every natural number x. Note that the first premise just provides a specific natural number t which is to be substituted for x in $A(x)$ *true* and is thus not essential in completing a proof by mathematical induction. Often t is just a term variable, in which case it is called an *induction variable*.

Using the new elimination rule, we can now complete the proof of $\forall x\in\mathsf{nat}.\exists y\in\mathsf{nat}.x < y\ \mathit{true}$. In the proof shown below, we use x as an induction variable and let $A(x) = x < \mathsf{s}(x)$ in the rule $\mathsf{natE}_I^{u(x)}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \in \mathsf{nat}}}{\mathsf{s}(x) \in \mathsf{nat}}\,\mathsf{natI_s} \qquad \dfrac{\dfrac{}{x\in\mathsf{nat}} \quad \dfrac{}{\mathbf{0} < \mathsf{s}(\mathbf{0})\ \mathit{true}}\,{<}\mathsf{I_0} \quad \dfrac{\dfrac{}{x<\mathsf{s}(x)\ \mathit{true}}\,u(x)}{\mathsf{s}(x) < \mathsf{s}(\mathsf{s}(x))\ \mathit{true}}\,{<}\mathsf{I_s}}{x < \mathsf{s}(x)\ \mathit{true}}\,\mathsf{natE}_I^{u(x)}}{\exists y\in\mathsf{nat}.x<y\ \mathit{true}}}{\forall x\in\mathsf{nat}.\exists y\in\mathsf{nat}.x<y\ \mathit{true}}\,\forall\mathsf{I}$$

Generalizing the case of datatype nat, we can derive from the definition of a datatype, or from its introduction rules, an elimination rule that is based on *induction on terms* and builds inductive proofs on terms. For example, the definition of datatype $\mathsf{list}\,\tau$ results in the following elimination rule:

$$\dfrac{t \in \mathsf{list}\,\tau \quad A(\mathbf{nil}^\tau)\ \mathit{true} \quad \begin{array}{c}\overline{x\in\tau}\quad \overline{l\in\mathsf{list}\,\tau}\quad \overline{A(l)\ \mathit{true}}^{\,u(l)}\\ \vdots \\ A(x :: l)\ \mathit{true}\end{array}}{A(t)\ \mathit{true}}\,\mathsf{list\,E}_I^{u(l)}$$

We can also devise proof terms for the new elimination rules. For example, we use the following proof term for the rule natE_I:

$$\text{proof term} \quad M \quad ::= \quad \cdots \mid \mathsf{ind}\ u(t)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow M \mid u(\mathsf{s}(x)) \Rightarrow N$$

$$\dfrac{t \in \mathsf{nat} \quad M : A(\mathbf{0}) \quad \begin{array}{c}\overline{x\in\mathsf{nat}}\quad \overline{u(x) : A(x)}\\ \vdots \\ N : A(\mathsf{s}(x))\end{array}}{\mathsf{ind}\ u(t)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow M \mid u(\mathsf{s}(x)) \Rightarrow N : A(t)}\,\mathsf{natE}_I$$

We can think of $\mathsf{ind}\ u(t)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow M \mid u(\mathsf{s}(x)) \Rightarrow N$ as an *inductive function* u applied to t. If N does not use $u(x)$, it degenerates to a case analysis construct and may be written as $\mathsf{case}\ t\ \mathsf{of}\ \mathbf{0} \Rightarrow M \mid \mathsf{s}(x) \Rightarrow N$:

$$\text{proof term} \quad M \quad ::= \quad \cdots \mid \mathsf{case}\ t\ \mathsf{of}\ \mathbf{0} \Rightarrow M \mid \mathsf{s}(x) \Rightarrow N$$

$$\dfrac{t \in \mathsf{nat} \quad M : A(\mathbf{0}) \quad \begin{array}{c}\overline{x\in\mathsf{nat}}\\ \vdots \\ N : A(\mathsf{s}(x))\end{array}}{\mathsf{case}\ t\ \mathsf{of}\ \mathbf{0} \Rightarrow M \mid \mathsf{s}(x) \Rightarrow N : A(t)}\,\mathsf{natE}_I$$

As an example, here is a proof term of type $\forall x\in\mathsf{nat}.\exists y\in\mathsf{nat}.x < y$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \in \mathsf{nat}}}{\mathsf{s}(x) \in \mathsf{nat}}\,\mathsf{natI_s} \qquad \dfrac{\dfrac{}{x\in\mathsf{nat}} \quad \dfrac{}{\mathsf{ltI_0} : \mathbf{0} < \mathsf{s}(\mathbf{0})}\,{<}\mathsf{I_0} \quad \dfrac{\dfrac{}{u(x) : x<\mathsf{s}(x)}}{\mathsf{ltI_s}(u(x)) : \mathsf{s}(x) < \mathsf{s}(\mathsf{s}(x))}\,{<}\mathsf{I_s}}{\mathsf{ind}\ u(t)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow \mathsf{ltI_0} \mid u(\mathsf{s}(x)) \Rightarrow \mathsf{ltI_s}(u(x)) : x < \mathsf{s}(x)}\,\mathsf{natE}_I}{\langle \mathsf{s}(x), \mathsf{ind}\ u(t)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow \mathsf{ltI_0} \mid u(\mathsf{s}(x)) \Rightarrow \mathsf{ltI_s}(u(x))\rangle : \exists y\in\mathsf{nat}.x<y}}{\lambda x \in \mathsf{nat}.\ \langle \mathsf{s}(x), \mathsf{ind}\ u(t)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow \mathsf{ltI_0} \mid u(\mathsf{s}(x)) \Rightarrow \mathsf{ltI_s}(u(x))\rangle : \exists y\in\mathsf{nat}.x<y : \forall x\in\mathsf{nat}.\exists y\in\mathsf{nat}.x<y}\,\forall\mathsf{I}$$

An elimination rule based on induction on terms gives rise to new β-reductions. For example, an introduction rule $\mathsf{natI_0}$ or $\mathsf{natI_s}$ (proving $t \in \mathsf{nat}$) followed by the elimination rule natE_I (proving

$A(t)$ *true*) forms a new pattern of detour, and removing such a detour corresponds to a β-reduction of a term of type $A(t)$. In the case of datatype nat, we obtain the following β-reductions:

$$\begin{array}{rcl}
\mathsf{ind}\ u(\mathbf{0})\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow M \mid u(\mathbf{s}(x)) \Rightarrow N & \Longrightarrow_\beta & M \\
\mathsf{ind}\ u(\mathbf{s}(t))\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow M \mid u(\mathbf{s}(x)) \Rightarrow N & \Longrightarrow_\beta & [\mathsf{ind}\ u(t)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow M \mid u(\mathbf{s}(x)) \Rightarrow N/u(x)][t/x]N
\end{array}$$

In the second β-reduction where $\mathbf{s}(t)$ matches $\mathbf{s}(x)$, we replace $u(x)$ in N by a proof term of type $A(t)$, namely $\mathsf{ind}\ u(t)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow M \mid u(\mathbf{s}(x)) \Rightarrow N$.

Note that elimination rules based on induction on terms are irrelevant to η-expansions. For example, an η-expansion of $t \in \mathsf{nat}$ must return another term of datatype nat, but the rule natE_I yields a proof term instead of a term. That is, the rule natE_I eliminates a judgment $t \in \mathsf{nat}$ to produce an incompatible judgment $M : A(t)$.

4.7 Examples

We have seen in Section 4.5 that the introduction rules for a predicate specify a unique set of elimination rules. For example, the introduction rules for the predicate $m =_\mathsf{N} n$

$$\frac{}{\mathbf{0} =_\mathsf{N} \mathbf{0}\ \mathit{true}} =_\mathsf{N}\mathsf{I}_0 \qquad \frac{m =_\mathsf{N} n\ \mathit{true}}{\mathbf{s}(m) =_\mathsf{N} \mathbf{s}(n)\ \mathit{true}} =_\mathsf{N}\mathsf{I}_\mathsf{s}$$

specify the following elimination rules:

$$\frac{\mathbf{0} =_\mathsf{N} \mathbf{s}(n)\ \mathit{true}}{C\ \mathit{true}} =_\mathsf{N}\mathsf{E}_{0\mathsf{s}} \qquad \frac{\mathbf{s}(m) =_\mathsf{N} \mathbf{0}\ \mathit{true}}{C\ \mathit{true}} =_\mathsf{N}\mathsf{E}_{\mathsf{s}0} \qquad \frac{\mathbf{s}(m) =_\mathsf{N} \mathbf{s}(n)\ \mathit{true}}{m =_\mathsf{N} n\ \mathit{true}} =_\mathsf{N}\mathsf{E}_\mathsf{s}$$

We have also seen in Section 4.6 that the introduction rules for a datatype specify an elimination rule based on induction on terms. For example, the introduction rules for datatype nat

$$\frac{}{\mathbf{0} \in \mathsf{nat}}\ \mathsf{natI}_0 \qquad \frac{t \in \mathsf{nat}}{\mathbf{s}(t) \in \mathsf{nat}}\ \mathsf{natI}_\mathsf{s}$$

specify the following elimination rule:

$$\frac{t \in \mathsf{nat} \quad A(\mathbf{0})\ \mathit{true} \quad \begin{array}{c} \overline{x \in \mathsf{nat}} \quad \overline{A(x)\ \mathit{true}}^{\,u(x)} \\ \vdots \\ A(\mathbf{s}(x))\ \mathit{true} \end{array}}{A(t)\ \mathit{true}}\ \mathsf{natE}_I^{u(x)}$$

Here we consider a few examples which use these rules to prove properties of natural numbers.

Example 1. $\forall x \in \mathsf{nat}. x =_\mathsf{N} x$

A judgment $\forall x \in \mathsf{nat}. x =_\mathsf{N} x\ \mathit{true}$ states that every natural number is equal to itself. Note that the judgment does *not* hold trivially because $=_\mathsf{N}$ is not a syntactic equality relation but a notational abbreviation of a predicate symbol EQ such that $EQ(m, n)$ means $m =_\mathsf{N} n$. That is, there is no reason that $x =_\mathsf{N} x$ should hold just because we intend $=_\mathsf{N}$ as an equality relation between natural numbers.

We begin with an inductive proof of $x =_\mathsf{N} x\ \mathit{true}$ where x is assumed to be an arbitrary natural number:

Proof. By induction on x.

Base case $x = \mathbf{0}$:
$\mathbf{0} =_\mathsf{N} \mathbf{0}\ \mathit{true}$ from $\frac{}{\mathbf{0} =_\mathsf{N} \mathbf{0}\ \mathit{true}} =_\mathsf{N}\mathsf{I}_0$

Inductive case $x = \mathbf{s}(x')$:
$x' =_\mathsf{N} x'\ \mathit{true}$ by induction hypothesis

$\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(x')\ \mathit{true}$ from $\dfrac{x' =_{\mathsf{N}} x'\ \mathit{true}}{\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(x')\ \mathit{true}} =_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}$

□

From this inductive proof, we obtain a derivation tree for the judgment $\forall x \in \mathsf{nat}. x =_{\mathsf{N}} x\ \mathit{true}$:

$$\dfrac{\dfrac{\overline{x \in \mathsf{nat}} \quad \dfrac{}{\mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}} =_{\mathsf{N}}\mathsf{I}_0 \quad \dfrac{\overline{x' =_{\mathsf{N}} x'\ \mathit{true}}^{\,u(x')}}{\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(x')\ \mathit{true}} =_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}}{x =_{\mathsf{N}} x\ \mathit{true}} \mathsf{natE}_I^{u(x')}}{\forall x \in \mathsf{nat}. x =_{\mathsf{N}} x\ \mathit{true}} \forall\mathsf{I}$$

Then we obtain a proof term of type $\forall x \in \mathsf{nat}. x =_{\mathsf{N}} x$ by assigning a proof term to every part of the derivation tree:

$$\dfrac{\dfrac{\overline{x \in \mathsf{nat}} \quad \dfrac{}{\mathsf{eqI}_0 : \mathbf{0} =_{\mathsf{N}} \mathbf{0}} =_{\mathsf{N}}\mathsf{I}_0 \quad \dfrac{\overline{u(x') : x' =_{\mathsf{N}} x'}}{\mathsf{eqI}_{\mathsf{s}}(u(x')) : \mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(x')} =_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}}{\mathsf{ind}\ u(x)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow \mathsf{eqI}_0 \mid u(\mathsf{s}(x')) \Rightarrow \mathsf{eqI}_{\mathsf{s}}(u(x')) : x =_{\mathsf{N}} x} \mathsf{natE}_I^{u(x')}}{\lambda x \in \mathsf{nat}.\ \mathsf{ind}\ u(x)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow \mathsf{eqI}_0 \mid u(\mathsf{s}(x')) \Rightarrow \mathsf{eqI}_{\mathsf{s}}(u(x')) : \forall x \in \mathsf{nat}. x =_{\mathsf{N}} x} \forall\mathsf{I}$$

An equivalent but easier way to obtain such a proof term is to begin with its specification. For example, we can derive a proof term *eqNat* of type $\forall x \in \mathsf{nat}. x =_{\mathsf{N}} x$ from the specification that *eqNat* x returns a proof term of type $x =_{\mathsf{N}} x$:

$$\begin{array}{lcll} & x & & \text{proof term of type } x =_{\mathsf{N}} x \\ \mathit{eqNat} & \mathbf{0} & = & \mathsf{eqI}_0 \\ \mathit{eqNat} & \mathsf{s}(x') & = & \mathsf{eqI}_{\mathsf{s}}(\mathit{eqNat}\ x') \end{array}$$

Note that *eqNat* may be recursively called only with argument x', just like a primitive recursive function applied to $\mathsf{s}(x')$ may be recursively called only with argument x'. Then we introduce an inductive function u and rewrite the specification into the definition of *eqNat* where *eqNat* x' changes to $u(x')$:

$$\mathit{eqNat} \ = \ \lambda x \in \mathsf{nat}.\ \mathsf{ind}\ u(x)\ \mathsf{of}\ u(\mathbf{0}) \Rightarrow \mathsf{eqI}_0 \mid u(\mathsf{s}(x')) \Rightarrow \mathsf{eqI}_{\mathsf{s}}(u(x'))$$

Example 2. $\forall x \in \mathsf{nat}. \forall y \in \mathsf{nat}. \forall z \in \mathsf{nat}. x =_{\mathsf{N}} y \supset y =_{\mathsf{N}} z \supset x =_{\mathsf{N}} z$

A judgment $\forall x \in \mathsf{nat}. \forall y \in \mathsf{nat}. \forall z \in \mathsf{nat}. x =_{\mathsf{N}} y \supset y =_{\mathsf{N}} z \supset x =_{\mathsf{N}} z\ \mathit{true}$ expresses the transitivity of the equality relation $=_{\mathsf{N}}$. An inductive proof of $x =_{\mathsf{N}} y \supset y =_{\mathsf{N}} z \supset x =_{\mathsf{N}} z\ \mathit{true}$ is given as follows:

Proof. By induction on x. We consider subcases on y and z. In each case, we assume $x =_{\mathsf{N}} y\ \mathit{true}$ and $y =_{\mathsf{N}} z\ \mathit{true}$ to show $x =_{\mathsf{N}} z\ \mathit{true}$.

Base case $x = \mathbf{0}$. We need to show $\mathbf{0} =_{\mathsf{N}} y \supset y =_{\mathsf{N}} z \supset \mathbf{0} =_{\mathsf{N}} z\ \mathit{true}$:
  Subcase $y = \mathbf{0}$:
    Subcase $z = \mathbf{0}$. We need to show $\mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$.
      $\mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$ by the rule $=_{\mathsf{N}}\mathsf{I}_0$
    Subcase $z = \mathsf{s}(z')$. We need to show $\mathbf{0} =_{\mathsf{N}} \mathsf{s}(z')\ \mathit{true}$.
      $\mathbf{0} =_{\mathsf{N}} \mathsf{s}(z')\ \mathit{true}$ from the assumption $y =_{\mathsf{N}} z\ \mathit{true}$
  Subcase $y = \mathsf{s}(y')$. We need to show $\mathbf{0} =_{\mathsf{N}} z\ \mathit{true}$.
    $\mathbf{0} =_{\mathsf{N}} \mathsf{s}(y')\ \mathit{true}$ from the assumption $x =_{\mathsf{N}} y\ \mathit{true}$
    $\mathbf{0} =_{\mathsf{N}} z\ \mathit{true}$ from $\dfrac{\mathbf{0} =_{\mathsf{N}} \mathsf{s}(y')\ \mathit{true}}{\mathbf{0} =_{\mathsf{N}} z\ \mathit{true}} =_{\mathsf{N}}\mathsf{E}_{0\mathsf{s}}$

Inductive case $x = \mathsf{s}(x')$. We need to show $\mathsf{s}(x') =_{\mathsf{N}} y \supset y =_{\mathsf{N}} z \supset \mathsf{s}(x') =_{\mathsf{N}} z\ \mathit{true}$:
  $x' =_{\mathsf{N}} y' \supset y' =_{\mathsf{N}} z' \supset x' =_{\mathsf{N}} z'\ \mathit{true}$ for any y' and z' by induction hypothesis
  Subcase $y = \mathbf{0}$. We need to show $\mathsf{s}(x') =_{\mathsf{N}} z\ \mathit{true}$.
    $\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$ from the assumption $x =_{\mathsf{N}} y\ \mathit{true}$
    $\mathsf{s}(x') =_{\mathsf{N}} z\ \mathit{true}$ from $\dfrac{\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}\ \mathit{true}}{\mathsf{s}(x') =_{\mathsf{N}} z\ \mathit{true}} =_{\mathsf{N}}\mathsf{E}_{\mathsf{s}0}$

Subcase $y = \mathsf{s}(y')$:

Subcase $z = \mathbf{0}$. We need to show $\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}$ *true*.

$\mathsf{s}(y') =_{\mathsf{N}} \mathbf{0}$ *true* — from the assumption $y =_{\mathsf{N}} z$ *true*

$\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}$ *true* — from $\dfrac{\mathsf{s}(y') =_{\mathsf{N}} \mathbf{0}\ \textit{true}}{\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}\ \textit{true}} =_{\mathsf{N}}\mathsf{E}_{\mathsf{s0}}$

Subcase $z = \mathsf{s}(z')$. We need to show $\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(z')$ *true*:

$\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(y')$ *true* — from the assumption $x =_{\mathsf{N}} y$ *true*

$x' =_{\mathsf{N}} y'$ *true* — from $\dfrac{\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(y')\ \textit{true}}{x' =_{\mathsf{N}} y'\ \textit{true}} =_{\mathsf{N}}\mathsf{E}_{\mathsf{s}}$

$\mathsf{s}(y') =_{\mathsf{N}} \mathsf{s}(z')$ *true* — from the assumption $y =_{\mathsf{N}} z$ *true*

$y' =_{\mathsf{N}} z'$ *true* — from $\dfrac{\mathsf{s}(y') =_{\mathsf{N}} \mathsf{s}(z')\ \textit{true}}{y' =_{\mathsf{N}} z'\ \textit{true}} =_{\mathsf{N}}\mathsf{E}_{\mathsf{s}}$

$x' =_{\mathsf{N}} z'$ *true* — from $x' =_{\mathsf{N}} y' \supset y' =_{\mathsf{N}} z' \supset x' =_{\mathsf{N}} z'$ *true*, $x' =_{\mathsf{N}} y'$ *true*, $y' =_{\mathsf{N}} z'$ *true*

$\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(z')$ *true* — from $\dfrac{x' =_{\mathsf{N}} z'\ \textit{true}}{\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(z')\ \textit{true}} =_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}$

□

Instead of rewriting the inductive proof as a derivation tree and then obtaining a corresponding proof term (which is tedious), we obtain a proof term *trans* directly from its specification:

	x	y	z	$v : x =_{\mathsf{N}} y$	$w : y =_{\mathsf{N}} z$		proof term of type $x =_{\mathsf{N}} z$
trans	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$v : \mathbf{0} =_{\mathsf{N}} \mathbf{0}$	$w : \mathbf{0} =_{\mathsf{N}} \mathbf{0}$	=	$\mathsf{eqI_0}$ or v or w
trans	$\mathbf{0}$	$\mathbf{0}$	$\mathsf{s}(z')$	$v : \mathbf{0} =_{\mathsf{N}} \mathbf{0}$	$w : \mathbf{0} =_{\mathsf{N}} \mathsf{s}(z')$	=	w or $\mathsf{eqE_{0s}}(w)$
trans	$\mathbf{0}$	$\mathsf{s}(y')$	z	$v : \mathbf{0} =_{\mathsf{N}} \mathsf{s}(y')$	$w : \mathsf{s}(y') =_{\mathsf{N}} z$	=	$\mathsf{eqE_{0s}}(v)$
trans	$\mathsf{s}(x')$	$\mathbf{0}$	z	$v : \mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}$	$w : \mathbf{0} =_{\mathsf{N}} z$	=	$\mathsf{eqE_{s0}}(v)$
trans	$\mathsf{s}(x')$	$\mathsf{s}(y')$	$\mathbf{0}$	$v : \mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(y')$	$w : \mathsf{s}(y') =_{\mathsf{N}} \mathbf{0}$	=	$\mathsf{eqE_{s0}}(w)$
trans	$\mathsf{s}(x')$	$\mathsf{s}(y')$	$\mathsf{s}(z')$	$v : \mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(y')$	$w : \mathsf{s}(y') =_{\mathsf{N}} \mathsf{s}(z')$	=	$\mathsf{eqI_s}(\textit{trans}\ x'\ y'\ z'\ \mathsf{eqE_s}(v)\ \mathsf{eqE_s}(w))$

It requires a bit of thinking to obtain a correct definition of *trans*. For example, here is a wrong definition of *trans* in which we mistakenly apply induction on x after taking y and z:

$$
\begin{array}{l}
\lambda x \in \mathsf{nat}.\, \lambda y \in \mathsf{nat}.\, \lambda z \in \mathsf{nat}. \\
\mathsf{ind}\ u(x)\ \mathsf{of}
\begin{cases}
u(\mathbf{0}) \Rightarrow \mathsf{case}\ y\ \mathsf{of}
\begin{cases}
\mathbf{0} \Rightarrow \mathsf{case}\ z\ \mathsf{of}
\begin{cases}
\mathbf{0} \Rightarrow \lambda v\!:\!\mathbf{0} =_{\mathsf{N}} \mathbf{0}.\, \lambda w\!:\!\mathbf{0} =_{\mathsf{N}} \mathbf{0}.\, \mathsf{eqI_0} \\
\mathsf{s}(z') \Rightarrow \lambda v\!:\!\mathbf{0} =_{\mathsf{N}} \mathbf{0}.\, \lambda w\!:\!\mathbf{0} =_{\mathsf{N}} \mathsf{s}(z').\, \mathsf{eqE_{0s}}(w)
\end{cases} \\
\mathsf{s}(y') \Rightarrow \lambda v\!:\!\mathbf{0} =_{\mathsf{N}} \mathsf{s}(y').\, \lambda w\!:\!\mathsf{s}(y') =_{\mathsf{N}} z.\, \mathsf{eqE_{0s}}(v)
\end{cases} \\
u(\mathsf{s}(x')) \Rightarrow \mathsf{case}\ y\ \mathsf{of}
\begin{cases}
\mathbf{0} \Rightarrow \lambda v\!:\!\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}.\, \lambda w\!:\!\mathbf{0} =_{\mathsf{N}} z.\, \mathsf{eqE_{s0}}(v) \\
\mathsf{s}(y') \Rightarrow \mathsf{case}\ z\ \mathsf{of}
\begin{cases}
\mathbf{0} \Rightarrow \lambda v\!:\!\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(y').\, \lambda w\!:\!\mathsf{s}(y') =_{\mathsf{N}} \mathbf{0}.\, \mathsf{eqE_{s0}}(w) \\
\mathsf{s}(z') \Rightarrow \lambda v\!:\!\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(y').\, \lambda w\!:\!\mathsf{s}(y') =_{\mathsf{N}} \mathsf{s}(z'). \\
\qquad \mathsf{eqI_s}(u(x')\ y'\ z'\ \mathsf{eqE_s}(v)\ \mathsf{eqE_s}(w))
\end{cases}
\end{cases}
\end{cases}
\end{array}
$$

This definition is wrong because $u(x')\ y'\ z'\ \mathsf{eqE_s}(v)\ \mathsf{eqE_s}(w)$ fails to typecheck: $u(x')$ has type $x' =_{\mathsf{N}} y \supset y =_{\mathsf{N}} z \supset x' =_{\mathsf{N}} z$, but it is applied to two terms y' and z' instead of two proof terms of types $x' =_{\mathsf{N}} y$ and $y =_{\mathsf{N}} z$. Neither does dropping y' and z' help because $\mathsf{eqE_s}(v)$ and $\mathsf{eqE_s}(w)$ have different types $x' =_{\mathsf{N}} y'$ and $y' =_{\mathsf{N}} z'$, respectively. The problem in this definition is that y and z are already fixed when induction on x starts, leaving no chance to use $u(x')$ to build a proof term of type $x' =_{\mathsf{N}} z'$ *true* from proof terms of types $x' =_{\mathsf{N}} y'$ and $y' =_{\mathsf{N}} z'$. Thus a correct definition of proof term *trans* starts induction on x before taking y and z as arguments:

$$
\begin{array}{l}
\lambda x \in \mathsf{nat}. \\
\mathsf{ind}\ u(x)\ \mathsf{of}
\begin{cases}
u(\mathbf{0}) \Rightarrow \lambda y \in \mathsf{nat}.\, \lambda z \in \mathsf{nat}. \\
\qquad \mathsf{case}\ y\ \mathsf{of}
\begin{cases}
\mathbf{0} \Rightarrow \mathsf{case}\ z\ \mathsf{of}
\begin{cases}
\mathbf{0} \Rightarrow \lambda v\!:\!\mathbf{0} =_{\mathsf{N}} \mathbf{0}.\, \lambda w\!:\!\mathbf{0} =_{\mathsf{N}} \mathbf{0}.\, \mathsf{eqI_0} \\
\mathsf{s}(z') \Rightarrow \lambda v\!:\!\mathbf{0} =_{\mathsf{N}} \mathbf{0}.\, \lambda w\!:\!\mathbf{0} =_{\mathsf{N}} \mathsf{s}(z').\, \mathsf{eqE_{0s}}(w)
\end{cases} \\
\mathsf{s}(y') \Rightarrow \lambda v\!:\!\mathbf{0} =_{\mathsf{N}} \mathsf{s}(y').\, \lambda w\!:\!\mathsf{s}(y') =_{\mathsf{N}} z.\, \mathsf{eqE_{0s}}(v)
\end{cases} \\
u(\mathsf{s}(x')) \Rightarrow \lambda y \in \mathsf{nat}.\, \lambda z \in \mathsf{nat}. \\
\qquad \mathsf{case}\ y\ \mathsf{of}
\begin{cases}
\mathbf{0} \Rightarrow \lambda v\!:\!\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}.\, \lambda w\!:\!\mathbf{0} =_{\mathsf{N}} z.\, \mathsf{eqE_{s0}}(v) \\
\mathsf{s}(y') \Rightarrow \mathsf{case}\ z\ \mathsf{of}
\begin{cases}
\mathbf{0} \Rightarrow \lambda v\!:\!\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(y').\, \lambda w\!:\!\mathsf{s}(y') =_{\mathsf{N}} \mathbf{0}.\, \mathsf{eqE_{s0}}(w) \\
\mathsf{s}(z') \Rightarrow \lambda v\!:\!\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(y').\, \lambda w\!:\!\mathsf{s}(y') =_{\mathsf{N}} \mathsf{s}(z'). \\
\qquad \mathsf{eqI_s}(u(x')\ y'\ z'\ \mathsf{eqE_s}(v)\ \mathsf{eqE_s}(w))
\end{cases}
\end{cases}
\end{cases}
\end{array}
$$

In this definition, $u(x)$ has type $\forall y \in \mathsf{nat}.\forall z \in \mathsf{nat}.x' =_{\mathsf{N}} y \supset y =_{\mathsf{N}} z \supset x' =_{\mathsf{N}} z$, allowing us to build a proof term of type $x' =_{\mathsf{N}} z'$ *true* from proof terms of types $x' =_{\mathsf{N}} y'$ and $y' =_{\mathsf{N}} z'$.

Example 3. $\forall x \in \mathsf{nat}.\neg(x =_{\mathsf{N}} \mathbf{0}) \supset \exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} x$

A judgment $\forall x \in \mathsf{nat}.\neg(x =_{\mathsf{N}} \mathbf{0}) \supset \exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} x$ *true* states that every non-zero natural number is the successor of some natural number. A proof of $\neg(x =_{\mathsf{N}} \mathbf{0}) \supset \exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} x$ *true*, which is not an inductive proof but reuses the proof of $\forall z \in \mathsf{nat}.z =_{\mathsf{N}} z$ *true*, is given as follows:

Proof. By case analysis of x.

Case $x = \mathbf{0}$. We need to show $\neg(\mathbf{0} =_{\mathsf{N}} \mathbf{0}) \supset \exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} \mathbf{0}$:

$\neg(\mathbf{0} =_{\mathsf{N}} \mathbf{0})$ *true*	assumption
$\exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} \mathbf{0}$ *true*	from $\neg(\mathbf{0} =_{\mathsf{N}} \mathbf{0})$ *true* and $\dfrac{}{\mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}} =_{\mathsf{N}}\mathsf{I}_0$

Case $x = \mathsf{s}(x')$. We need to show $\neg(\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}) \supset \exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} \mathsf{s}(x')$:

$\neg(\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0})$ *true*	assumption (which is not used in this case)
$x' =_{\mathsf{N}} x'$ *true*	from the proof of $\forall z \in \mathsf{nat}.z =_{\mathsf{N}} z$ *true* and $x' \in \mathsf{nat}$
$\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(x')$ *true*	from $\dfrac{x' =_{\mathsf{N}} x'\ \mathit{true}}{\mathsf{s}(x') =_{\mathsf{N}} \mathsf{s}(x')\ \mathit{true}} =_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}$
$\exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} \mathsf{s}(x')$ *true*	from $\dfrac{x' \in \mathsf{nat} \quad \mathsf{s}(x') =_{\mathsf{N}} x\ \mathit{true}}{\exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} x\ \mathit{true}} \exists\mathsf{I}$

□

The specification of a proof term *pred* of type $\forall x \in \mathsf{nat}.\neg(x =_{\mathsf{N}} \mathbf{0}) \supset \exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} x$ is:

$$\begin{array}{lll c l} & x & v : \neg(x =_{\mathsf{N}} \mathbf{0}) & & \text{proof term of type } \exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} x \\ \mathit{pred} & \mathbf{0} & v : \neg(\mathbf{0} =_{\mathsf{N}} \mathbf{0}) & = & \mathsf{abort}_{\exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} \mathbf{0}}\ (v\ \mathsf{eqI}_0) \\ \mathit{pred} & \mathsf{s}(x') & v : \neg(\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}) & = & \langle x', \mathsf{eqI}_{\mathsf{s}}(\mathit{eqNat}\ x')\rangle \end{array}$$

From this specification, we obtain the following definition of *pred*:

$$\mathit{pred} \;=\; \lambda x \in \mathsf{nat}.\,\mathsf{case}\ x\ \mathsf{of}\ \begin{cases} \mathbf{0} \Rightarrow \lambda v{:}\neg(\mathbf{0} =_{\mathsf{N}} \mathbf{0}).\,\mathsf{abort}_{\exists y \in \mathsf{nat}.\mathsf{s}(y) =_{\mathsf{N}} \mathbf{0}}\ (v\ \mathsf{eqI}_0) \\ \mathsf{s}(x') \Rightarrow \lambda v{:}\neg(\mathsf{s}(x') =_{\mathsf{N}} \mathbf{0}).\,\langle x', \mathsf{eqI}_{\mathsf{s}}(\mathit{eqNat}\ x')\rangle \end{cases}$$

Exercise 4.1. Can you give a definition of *pred* of the following form?

$$\mathit{pred} \;=\; \lambda x \in \mathsf{nat}.\,\lambda v{:}\neg(x =_{\mathsf{N}} \mathbf{0}).\,\mathsf{case}\ x\ \mathsf{of}\ \begin{cases} \mathbf{0} \Rightarrow \ldots \\ \mathsf{s}(x') \Rightarrow \ldots \end{cases}$$

Exercise 4.2. Give a proof of $\forall x \in \mathsf{nat}.\forall y \in \mathsf{nat}.x =_{\mathsf{N}} y \supset y =_{\mathsf{N}} x$ *true*. What is its proof term?

Exercise 4.3. Give a proof of $\forall x \in \mathsf{nat}.\forall y \in \mathsf{nat}.x < y \supset \neg(x =_{\mathsf{N}} y)$ *true*. What is its proof term?

4.8 Induction on predicates

In Section 4.5, we have seen how to derive a set of elimination rules for a predicate from its introduction rules. For example, the introduction rules $=_{\mathsf{N}}\mathsf{I}_0$ and $=_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}$ for the predicate $m =_{\mathsf{N}} n$ specify the elimination rules $=_{\mathsf{N}}\mathsf{E}_{0\mathsf{s}}$, $=_{\mathsf{N}}\mathsf{E}_{\mathsf{s}0}$, and $=_{\mathsf{N}}\mathsf{E}_{\mathsf{s}}$. Now we show how to derive yet another elimination rule from the introduction rules for a predicate. Such an elimination rule is based on *induction on predicates* and allows us to prove properties of terms satisfying certain predicates.

Suppose that we wish to show a property that whenever $m_0 =_{\mathsf{N}} n_0$ *true* holds, we have a proof of $A(m_0, n_0)$ where $A(m_0, n_0)$ can be any proposition involving terms m_0 and n_0. For example, we may have $A(m_0, n_0) = n_0 =_{\mathsf{N}} m_0$, in which case we attempt to prove the judgment in Exercise 4.2. We consider two cases of building a proof of $m_0 =_{\mathsf{N}} n_0$ *true*:

- The proof of $m_0 =_{\mathsf{N}} n_0$ *true* uses the rule $=_{\mathsf{N}}\mathsf{I}_0$. In this case, we have $m_0 = \mathbf{0}$ and $n_0 = \mathbf{0}$, and thus need to prove $A(\mathbf{0}, \mathbf{0})$ *true*.

- The proof of $m_0 =_{\mathsf{N}} n_0\ \mathit{true}$ uses the rule $=_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}$. In this case, we have $m_0 = \mathbf{s}(m)$ and $n_0 = \mathbf{s}(n)$, and thus need to prove $A(\mathbf{s}(m), \mathbf{s}(n))\ \mathit{true}$ from the assumption of $m =_{\mathsf{N}} n\ \mathit{true}$. In addition, the principle of induction allows us to make another assumption of $A(m, n)\ \mathit{true}$ because according to the rule $=_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}$, $m =_{\mathsf{N}} n\ \mathit{true}$ uses a "smaller" predicate than $\mathbf{s}(m) =_{\mathsf{N}} \mathbf{s}(n)\ \mathit{true}$ and is assumed to already satisfy the property.

The analysis in these two cases justifies the following elimination rule:

$$\dfrac{m_0 =_{\mathsf{N}} n_0\ \mathit{true} \qquad A(\mathbf{0},\mathbf{0})\ \mathit{true} \qquad \begin{array}{c}\overline{m \in \mathsf{nat}} \quad \overline{n \in \mathsf{nat}} \quad \overline{m =_{\mathsf{N}} n}^{\,w} \quad \overline{A(m,n)\ \mathit{true}}^{\,u(m,n)} \\ \vdots \\ A(\mathbf{s}(m),\mathbf{s}(n))\ \mathit{true}\end{array}}{A(m_0,n_0)\ \mathit{true}}\ =_{\mathsf{N}}\mathsf{E}_I^{w,u(m,n)}$$

Note that as is the case for induction on terms, the second and third premises do not use m_0 and n_0 at all and constitute a valid proof of $A(m, n)\ \mathit{true}$ for every pair of natural numbers m and n. Thus the first premise just provides two specific natural number m_0 and n_0 to be substituted for m and n in $A(m, n)\ \mathit{true}$ and are not essential in completing a proof by induction on predicates.

As an example of using the rule $=_{\mathsf{N}}\mathsf{E}_I$, here is a proof of the judgment in Exercise 4.2 where we let $A(m, n) = n =_{\mathsf{N}} m$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{x =_{\mathsf{N}} y\ \mathit{true}}^{\,w} \qquad \dfrac{}{\mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}}\,=_{\mathsf{N}}\mathsf{I}_0 \qquad \dfrac{\overline{n =_{\mathsf{N}} m\ \mathit{true}}^{\,u(m,n)}}{\mathbf{s}(n) =_{\mathsf{N}} \mathbf{s}(m)\ \mathit{true}}\,=_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}}{y =_{\mathsf{N}} x\ \mathit{true}}\,=_{\mathsf{N}}\mathsf{E}_I^{w,u(m,n)}}{x =_{\mathsf{N}} y \supset y =_{\mathsf{N}} x\ \mathit{true}}\,\supset\!\mathsf{I}^w}{\forall y\!\in\!\mathsf{nat}.x =_{\mathsf{N}} y \supset y =_{\mathsf{N}} x\ \mathit{true}}\,\forall\mathsf{I}}{\forall x\!\in\!\mathsf{nat}.\forall y\!\in\!\mathsf{nat}.x =_{\mathsf{N}} y \supset y =_{\mathsf{N}} x\ \mathit{true}}\,\forall\mathsf{I}$$

In a similar way, the introduction rules for the predicate $m < n$ specify the following elimination rule based on induction on predicates:

$$\dfrac{m_0 < n_0\ \mathit{true} \qquad \begin{array}{c}\overline{n \in \mathsf{nat}} \\ \vdots \\ A(\mathbf{0},\mathbf{s}(n))\ \mathit{true}\end{array} \qquad \begin{array}{c}\overline{m \in \mathsf{nat}} \quad \overline{n \in \mathsf{nat}} \quad \overline{m < n}^{\,w} \quad \overline{A(m,n)\ \mathit{true}}^{\,u(m,n)} \\ \vdots \\ A(\mathbf{s}(m),\mathbf{s}(n))\ \mathit{true}\end{array}}{A(m_0,n_0)\ \mathit{true}}\ <\mathsf{E}_I^{w,u(m,n)}$$

We can now simplify the proof of the judgment in Exercise 4.3 by using the rule $<\mathsf{E}_I$ with $A(m, n) = \neg(m =_{\mathsf{N}} n)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{x < y\ \mathit{true}}^{\,w} \qquad \dfrac{\dfrac{\overline{\mathbf{0} =_{\mathsf{N}} \mathbf{s}(n)\ \mathit{true}}^{\,v}}{\bot\ \mathit{true}}\,=_{\mathsf{N}}\mathsf{E}_{0\mathsf{s}}}{\neg(\mathbf{0} =_{\mathsf{N}} \mathbf{s}(n))\ \mathit{true}}\,\neg\mathsf{I}^v \qquad \dfrac{\dfrac{\overline{\neg(m =_{\mathsf{N}} n)\ \mathit{true}}^{\,u(m,n)} \qquad \dfrac{\overline{\mathbf{s}(m) =_{\mathsf{N}} \mathbf{s}(n)\ \mathit{true}}^{\,v}}{m =_{\mathsf{N}} n\ \mathit{true}}\,=_{\mathsf{N}}\mathsf{E}_{\mathsf{s}}}{\bot\ \mathit{true}}\,\neg\mathsf{E}}{\neg(\mathbf{s}(m) =_{\mathsf{N}} \mathbf{s}(n))\ \mathit{true}}\,\neg\mathsf{I}^v}{\neg(x =_{\mathsf{N}} y)\ \mathit{true}}\,<\mathsf{E}_I^{w,u(m,n)}}{x < y \supset \neg(x =_{\mathsf{N}} y)\ \mathit{true}}\,\supset\!\mathsf{I}^w}{\forall y\!\in\!\mathsf{nat}.x < y \supset \neg(x =_{\mathsf{N}} y)\ \mathit{true}}\,\forall\mathsf{I}}{\forall x\!\in\!\mathsf{nat}.\forall y\!\in\!\mathsf{nat}.x < y \supset \neg(x =_{\mathsf{N}} y)\ \mathit{true}}\,\forall\mathsf{I}$$

4.9 Definitional equality

So far, we have seen how to use predicates on terms to prove various properties of datatypes. Incidentally these terms are not further reducible by β-reductions because they are either variables or built only by introduction rules, as in $\mathbf{0} =_{\mathsf{N}} \mathbf{0}$, $x =_{\mathsf{N}} y$, and $\mathbf{s}(x) =_{\mathsf{N}} \mathbf{s}(y)$. Now we consider predicates containing terms that may reduce to simpler terms by β-reductions. For example, a predicate $\mathit{plus}\ \mathbf{0}\ \mathbf{0} =_{\mathsf{N}} \mathbf{0}$ contains a term $\mathit{plus}\ \mathbf{0}\ \mathbf{0}$ which reduces to $\mathbf{0}$ by β-reductions. Note that $\mathit{plus}\ \mathbf{0}\ \mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$ is *not* provable because the introduction rules $=_{\mathsf{N}}\mathsf{I}_0$ and $=_{\mathsf{N}}\mathsf{I}_{\mathsf{s}}$ allow us to prove judgments of the form $\mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$ and $\mathsf{s}(m) =_{\mathsf{N}} \mathsf{s}(n)\ \mathit{true}$ only. Since $\mathit{plus}\ \mathbf{0}\ \mathbf{0}$ and $\mathbf{0}$ denote the same natural number, we would like to be able to prove $\mathit{plus}\ \mathbf{0}\ \mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$.

This section develops a methodology that enables us to prove such judgments as $\mathit{plus}\ \mathbf{0}\ \mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$. The basic idea is to define a notion of equality $=$, called *definitional equality*, which identifies two terms that reduce to the same term by β-reductions. For example, we have an equality $\mathit{plus}\ \mathbf{0}\ \mathbf{0} = \mathbf{0}$ because $\mathit{plus}\ \mathbf{0}\ \mathbf{0}$ reduces to $\mathbf{0}$ by β-reductions. Then the equality $\mathit{plus}\ \mathbf{0}\ \mathbf{0} = \mathbf{0}$ allows us to simplify the judgment $\mathit{plus}\ \mathbf{0}\ \mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$ to $\mathbf{0} =_{\mathsf{N}} \mathbf{0}\ \mathit{true}$, which is provable.

It is important that although definitional equality uses the symbol $=$ which usually stands for syntactic equality, it is a strict extension of syntactic equality. For example, $\mathbf{0} = \mathbf{0}$ holds under both syntactic equality (because x is syntactically equal to x) and definitional equality (because x reduces to x by zero β-reductions), but $\mathit{plus}\ \mathbf{0}\ \mathbf{0} = \mathbf{0}$ does not hold under syntactic equality (because $\mathit{plus}\ \mathbf{0}\ \mathbf{0}$ is syntactically different from $\mathbf{0}$) while it holds under definitional equality. We also note that definitional equality has nothing to do with the symbol $=_{\mathsf{N}}$ in the predicate $m =_{\mathsf{N}} n$ which is just a syntactic abbreviation of $EQ(m, n)$.

Formally we use a new judgment $t = s$ to mean that terms t and s are definitionally equal. (Instead of writing $t = s\ \mathit{true}$, we write $t = s$, omitting true.) As with predicates, we base the definition of $t = s$ on the principle of natural deduction:

$$\frac{t \Longrightarrow_\beta^* r \quad s \Longrightarrow_\beta^* r}{t = s}\ \mathit{DefEq}\mathsf{I} \qquad \frac{A(t)\ \mathit{true} \quad t = s}{A(s)\ \mathit{true}}\ \mathit{DefEq}\mathsf{E}$$

In the introduction rule $\mathit{DefEq}\mathsf{I}$, the judgment $t \Longrightarrow_\beta^* r$ means that t reduces to r by zero or more β-reductions. Here we assume that β-reductions may be applied to subterms of the term being reduced. For example, if $t_1 \Longrightarrow_\beta t_2$ holds, $\mathsf{s}(t_1) \Longrightarrow_\beta^* \mathsf{s}(t_2)$ holds because t_1 is a subterm of $\mathsf{s}(t_1)$. Thus the rule $\mathit{DefEq}\mathsf{I}$ states that two terms are definitionally equal if both reduce to the same term by β-reductions. As a special case, two terms t and s are definitionally equal if t reduces to s by β-reductions or vice versa.

- If $t \Longrightarrow_\beta^* s$ or $s \Longrightarrow_\beta^* t$, then $t = s$.

The elimination rule $\mathit{DefEq}\mathsf{E}$ states that once we build a proof of $t = s$, we cease to distinguish between (syntactically different) propositions $A(t)$ and $A(s)$. The corresponding typing rule allows us to change the type of a proof term silently without changing the proof term itself:

$$\frac{M : A(t) \quad t = s}{M : A(s)}\ \mathit{DefEq}\mathsf{E}$$

According to the rule $\mathit{DefEq}\mathsf{I}$, definitional equality is a relation between terms which is reflexive and commutative:

- $t = t$ holds for any term t.

- $t = s$ implies $s = t$.

Definitional equality is not necessarily transitive because the set of terms is open-ended and thus can be extended with new terms that destroy the transitivity of definitional equality. It is transitive, however, in the following weak sense:

- If $t = s$ and $s = r$, then $A(t)\ \mathit{true}$ implies $A(r)\ \mathit{true}$.

Here are a few examples of definitional equality:

- $\mathit{pred}\ \mathbf{0} = \mathbf{0}$ holds because we have $\mathit{pred}\ \mathbf{0} \Longrightarrow_\beta \mathbf{case}\ \mathbf{0}\ \mathbf{of}\ \mathbf{0} \Rightarrow \mathbf{0} \mid \mathsf{s}(y) \Rightarrow y \Longrightarrow_\beta \mathbf{0}$.

- The sequence of β-reductions on Page 29 proves $\mathit{plus}\ \mathsf{s}(\mathbf{0})\ t = \mathsf{s}(t)$.

- $\mathit{plus}\ \mathbf{0}\ x = x$ holds:

$$\begin{array}{lll}
\mathit{plus}\ \mathbf{0}\ x & \Longrightarrow_\beta & (\lambda y \in \mathsf{nat}.\, \mathbf{rec}\ p(\mathbf{0})\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow y \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)))\ x \\
& \Longrightarrow_\beta & \mathbf{rec}\ p(\mathbf{0})\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow x \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)) \\
& \Longrightarrow_\beta & x
\end{array}$$

- $\mathit{plus}\ x\ \mathbf{0} = x$ does *not* holds:

$$\begin{array}{lll}
\mathit{plus}\ x\ \mathbf{0} & \Longrightarrow_\beta & (\lambda y \in \mathsf{nat}.\, \mathbf{rec}\ p(x)\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow y \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)))\ \mathbf{0} \\
& \Longrightarrow_\beta & \mathbf{rec}\ p(x)\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow \mathbf{0} \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)) \\
& \Longrightarrow_\beta & ???
\end{array}$$

As an example of a proof using definitional equality, let us find a proof term of the following type which states that every natural number is either even or odd:

$$\forall x \in \mathsf{nat}.(\exists y \in \mathsf{nat}.y + y =_{\mathsf{N}} x) \vee (\exists y \in \mathsf{nat}.\mathbf{s}(y + y) =_{\mathsf{N}} x)$$

Here we write $t + s$ for $\mathit{plus}\ t\ s$. Note that without definitional equality, it is impossible to find such a proof term because there is no way to prove, for example, $y + y =_{\mathsf{N}} x$.

First we observe that $\mathbf{s}(x) + y = \mathbf{s}(x + y)$ holds:

$$\begin{array}{lll}
\mathbf{s}(x) + y & \Longrightarrow_\beta^* & \mathbf{rec}\ p(\mathbf{s}(x))\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow y \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)) \\
& \Longrightarrow_\beta & \mathbf{s}(\mathbf{rec}\ p(x)\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow y \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z))) \\
\mathbf{s}(x + y) & \Longrightarrow_\beta^* & \mathbf{s}(\mathbf{rec}\ p(x)\ \mathbf{of}\ p(\mathbf{0}) \Rightarrow y \mid p(\mathbf{s}(z)) \Rightarrow \mathbf{s}(p(z)))
\end{array}$$

Next we define a proof term comp whose type is $\forall x \in \mathsf{nat}.\forall y \in \mathsf{nat}.x + \mathbf{s}(y) =_{\mathsf{N}} \mathbf{s}(x + y)$:

$$\mathit{comp} \ = \ \lambda x \in \mathsf{nat}.\, \lambda y \in \mathsf{nat}.\, \mathsf{ind}\ u(x)\ \mathsf{of} \left\{ \begin{array}{l} u(\mathbf{0}) \Rightarrow \mathit{eqNat}\ \mathbf{s}(y) \\ u(\mathbf{s}(x')) \Rightarrow \mathsf{eql}_{\mathsf{s}}(u(x')) \end{array} \right.$$

Here $\mathit{eqNat}\ \mathbf{s}(y)$ is assigned type $\mathbf{0} + \mathbf{s}(y) =_{\mathsf{N}} \mathbf{s}(\mathbf{0} + y)$ which is equivalent to $\mathbf{s}(y) =_{\mathsf{N}} \mathbf{s}(y)$ under definitional equality. Similarly $\mathsf{eql}_{\mathsf{s}}(u(x'))$ is assigned type $\mathbf{s}(x') + \mathbf{s}(y) =_{\mathsf{N}} \mathbf{s}(\mathbf{s}(x') + y)$ which is equivalent to $\mathbf{s}(x' + \mathbf{s}(y)) =_{\mathsf{N}} \mathbf{s}(\mathbf{s}(x' + y))$ under definitional equality. Then we use the proof term trans given in Section 4.7 to obtain a proof term of the given type:

$$\begin{array}{l}
\lambda x \in \mathsf{nat}. \\
\quad \mathsf{ind}\ u(x)\ \mathsf{of} \left\{ \begin{array}{l} u(\mathbf{0}) \Rightarrow \mathsf{inl}_{\exists y \in \mathsf{nat}.\mathbf{s}(y+y) =_{\mathsf{N}} \mathbf{0}}\ \langle \mathbf{0}, \mathsf{eql}_0 \rangle \\ u(\mathbf{s}(x')) \Rightarrow \end{array} \right. \\
\qquad \mathsf{case}\ u(x')\ \mathsf{of} \left\{ \begin{array}{ll} & \mathsf{inl}\ z.\, \mathsf{let}\ \langle y, w \rangle = z\ \mathsf{in}\ \mathsf{inr}_{\exists y \in \mathsf{nat}.y+y =_{\mathsf{N}} \mathbf{s}(x')}\ \langle y, \mathsf{eql}_{\mathsf{s}}(w) \rangle \\ \mid & \mathsf{inr}\ z.\, \mathsf{let}\ \langle y, w \rangle = z\ \mathsf{in} \end{array} \right. \\
\qquad\quad \mathsf{inl}_{\exists y \in \mathsf{nat}.\mathbf{s}(y+y) =_{\mathsf{N}} \mathbf{s}(x')}\ \langle \mathbf{s}(y), \mathit{trans}\ (\mathbf{s}(y) + \mathbf{s}(y))\ (\mathbf{s}(\mathbf{s}(y + y)))\ (\mathbf{s}(x'))\ (\mathit{comp}\ \mathbf{s}(y)\ y) \mathsf{eql}_{\mathsf{s}}(w) \rangle
\end{array}$$