Computation and Deduction

Frank Pfenning
Carnegie Mellon University

$\boxed{\text{Draft of February 27, 1997}}$

Draft notes for a course given at Carnegie Mellon University during the fall semester of 1994. Please send comments to `fp@cs.cmu.edu`. Do not cite, copy, or distribute without the express written consent of Frank Pfenning and the National Football League.

Contents

Chapter 7

Natural Deduction

Ich wollte zunächst einmal einen Formalismus aufstellen, der dem wirklichen Schließen möglichst nahe kommt. So ergab sich ein ,,Kalkül des natürlichen Schließens".

— Gerhard Gentzen
Untersuchungen über das logische Schließen [Gen35]

In Chapter 2 we introduced the functional language Mini-ML. This language was defined by its type system and its operational semantics, both specified via deductive systems. The type system gives us a means to verify some properties of Mini-ML programs. Assume, for example, we can show that $e : \mathsf{nat}$ for some expression e. Then we know that if e evaluates to v (that is, $e \hookrightarrow v$ is derivable) then v is a value by Theorem 2.1. We know furthermore that $v : \mathsf{nat}$ by type preservation (Theorem 2.5). It is then easy to show that v must have the form $\mathbf{s}\ (\mathbf{s} \ldots (\mathbf{s}\ \mathbf{z}) \ldots)$. In other words, v must represent a natural number. As another example, consider an expression e of type $\mathsf{nat} \rightarrow \mathsf{nat}$. Then e represents some partial function from natural numbers to natural numbers: for each argument $v : \mathsf{nat}$ the expression $e\ v$ either evaluates to some v' which represents a natural number or does not have a value. The uniqueness of v' is the subject of Exercise 2.16.

The type system can only capture simple properties. It can express that some program e represents a partial function from natural numbers to natural numbers, but it cannot express *which* partial function. This simplicity has a great advantage: the question if an expression has a given type is effectively decidable, and thus a compiler can mechanically verify program properties expressed as types. In order to reason about more complex properties of programs within some formal system, we need a more expressive language of specifications and some way of connecting programs to the specifications they satisfy. Designing more expressive type systems, usually called *type theories*, is one path which is the subject of much current research. For some of these systems, type checking remains decidable; for others it becomes undecidable.

We will take a different path and consider *predicate logic* as a means of expressing specifications. The connection to programs will later be made via the so-called Curry-Howard isomorphism which interprets constructive proofs as programs and formulas as types. There are many ways a logical system can be specified, and relationships between them have been intensively investigated. The most important styles of presentation are *axiomatic systems* (often associated with Hilbert [HB34]), *systems of natural deduction*, and *sequent calculi* (the latter two are due to Gentzen [Gen35]). In an axiomatic system, the logic is described by some axioms and a minimal set of inference rules in order to derive new theorems from the axioms and prior theorems. Systems of natural deduction on the other hand try to explicate the meaning of the logical connectives and quantifiers by means of inference rules only. This is in the same spirit as the approach of LF whose design was based on natural deduction. Sequent calculi can instead be considered as calculi for proof search. For the investigation of the connections between proofs and programs a system of natural deduction is most appropriate.

In Chapter 8 we will have occasion to consider *logic programming*, which is based on a different computational mechanism than functional languages. Rather than evaluation via substitution, computation is based on the notion of search for a proof in a logic following a particular strategy. There, too, the system of natural deduction will be of great importance.

7.1 Natural Deduction

The system of natural deduction we describe below is basically Gentzen's system NJ [Gen35] or the system which may be found in Prawitz [Pra65]. The calculus of natural deduction was devised by Gentzen in the 1930's out of a dissatisfaction with axiomatic systems in the Hilbert tradition, which did not seem to capture mathematical reasoning practices very directly. Instead of a number of axioms and a small set of inference rules, deductions are described through inference rules only, which at the same time explain the meaning of the logical quantifiers and connectives in terms of their proof rules. This is often called *proof-theoretic semantics*, an approach which has gained popularity in computer science through the work of de Bruijn [dB80] and Martin-Löf [ML80, ML85a].

A language of (first-order) *terms* is built up from *variables* x, y, *etc.*, *function symbols* f, g, *etc.*, each with a unique arity, and *parameters* a, b, *etc.* in the usual way.

$$\textit{Terms} \quad t \quad ::= \quad x \mid a \mid f(t_1, \ldots, t_n)$$

A constant c is simply a function symbol with arity 0 and we write c instead of $c()$. Exactly which function symbols are available is left unspecified in the general development of predicate logic and only made concrete for specific theories, such as the theory of natural numbers. However, variables and parameters are always available. We will use t and s to range over terms.

The language of *formulas* is built up from *predicate symbols* P, Q, *etc.* and terms in the usual way.

$$\begin{array}{lcl} \textit{Formulas} \quad A & ::= & P(t_1, \ldots, t_n) \mid A_1 \wedge A_2 \mid A_1 \supset A_2 \mid A_1 \vee A_2 \mid \neg A \mid \bot \mid \top \\ & & \mid \forall x.\ A \mid \exists x.\ A \end{array}$$

A propositional constant P is simply a predicate symbol with no arguments and we write P instead of $P()$. We will use A, B, and C to range over formulas. Exactly which predicate symbols are available is left unspecified in the general development of predicate logic and only made concrete for specific theories.

The notions of *free* and *bound* variables in terms and formulas are defined in the usual way: the variable x is bound in formulas of the form $\forall x.\ A$ and $\exists x.\ A$. We use parentheses to disambiguate and assume that $\wedge$ and $\vee$ bind more tightly than $\supset$. It is convenient to assume that formulas have no free individual variables; we use parameters instead where necessary. Our notation for substitution is $[t/x]A$ for the result of substituting the term t for the variable x in A. Because of the restriction on occurrences of free variables, we can assume that t is free of individual variables, and thus capturing cannot occur.

The main judgment of natural deduction is the derivability of a formula C, written as $\vdash C$, from assumptions $\vdash A_1, \ldots, \vdash A_n$. We will model this as a hypothetical judgment. This means that certain structural properties of derivations are tacitly assumed, independently of any logical inferences.

Assumption. If we have an assumption $\vdash A$ than we can conclude $\vdash A$. For example, $\vdash A$ by itself represents a deduction of $\vdash A$ from assumption $\vdash A$.

Weakening. If $\vdash C$ is derivable from $\vdash A_1, \ldots, \vdash A_n$ then $\vdash C$ is also derivable from $\vdash A_1, \ldots, \vdash A_n, \vdash A_{n+1}$. Alternatively, we could say that assumptions need not be used. For example, $\vdash A$ by itself also represents a deduction of $\vdash A$ from assumptions $\vdash A$ and $\vdash B$, even through B is not even mentioned.

Duplication. Assumptions can be used more than once.

Exchange. The order of assumptions is irrelevant.

In keeping with general mathematical practice in the discussion of natural deduction, we will omit the turnstile $\vdash$ and let a formula A itself stand for the judgment $\vdash A$. It is important to keep in mind that this is merely a shorthand, and that we are defining a judgment via inference rules in the same manner as in earlier chapters in this book.

In natural deduction each logical connective and quantifier is characterized by its *introduction rule(s)* which specifies how to infer a conjunction, disjunction, *etc.*

The *elimination rule* for the logical constant tells us how we can assumptions in the form of a conjunction, disjunction, *etc.* The introduction and elimination rules must match in a certain way in order to guarantee the consistency of the system: if we introduce a connective and then immediately eliminate it, we should be able to erase this detour and find a more direct derivation of the conclusion. The rules are summarized on page 7.1.

Conjunction. $A \wedge B$ should be derivable if both A and B are derivable. Thus we have the following introduction rule.

$$\frac{A \qquad B}{A \wedge B}\wedge\mathrm{I}$$

If we consider this as a complete definition, we should be able to recover both A and B if we know $A \wedge B$. We are thus led to two elimination rules.

$$\frac{A \wedge B}{A}\wedge\mathrm{E_L} \qquad \frac{A \wedge B}{B}\wedge\mathrm{E_R}$$

To check our intuition we consider a deduction which ends in an introduction followed by an elimination:

$$\cfrac{\cfrac{\begin{matrix}\mathcal{D}\\A\end{matrix} \qquad \begin{matrix}\mathcal{E}\\B\end{matrix}}{A \wedge B}\wedge\mathrm{I}}{A}\wedge\mathrm{E_L}$$

Clearly, it is unnecessary to first introduce the conjunction and then eliminate it: a more direct proof of the same conclusion from the same (or fewer) assumptions would be simply

$$\begin{matrix}\mathcal{D}\\A\end{matrix}$$

Formulated as a transformation or *reduction* between derivations we have

$$\cfrac{\cfrac{\begin{matrix}\mathcal{D}\\A\end{matrix} \qquad \begin{matrix}\mathcal{E}\\B\end{matrix}}{A \wedge B}\wedge\mathrm{I}}{A}\wedge\mathrm{E_L} \quad \Longrightarrow_L \quad \begin{matrix}\mathcal{D}\\A\end{matrix}$$

and symmetrically

$$\cfrac{\cfrac{\begin{matrix}\mathcal{D}\\A\end{matrix} \qquad \begin{matrix}\mathcal{E}\\B\end{matrix}}{A \wedge B}\wedge\mathrm{I}}{B}\wedge\mathrm{E_R} \quad \Longrightarrow_L \quad \begin{matrix}\mathcal{E}\\B\end{matrix}$$

The new judgment

$$\mathcal{D} :: \vdash A \quad \Longrightarrow_L \quad \mathcal{E} :: \vdash A$$

relates derivations with the same conclusion. We say $\mathcal{D}$ *locally reduces to* $\mathcal{E}$. Later in Section 7.7 we will define a another judgment of reduction in which local reductions can be applied to any subderivation.

Implication. To derive $A \supset B$ we assume A and then derive B. Written as a hypothetical judgment:

$$\dfrac{\begin{array}{c}\overline{A}^{\,u}\\ \vdots \\ B\end{array}}{A \supset B}\supset\mathrm{I}^u$$

Thus a derivation of $A \supset B$ describes a construction by which we can transform a derivation of A into a derivation of B: we substitute the derivation of A wherever we used the assumption A in the hypothetical derivation of B. The elimination rule expresses this: if we have a derivation of $A \supset B$ and also a derivation of A, then we can obtain a derivation of B.

$$\dfrac{A \supset B \qquad A}{B}\supset\mathrm{E}$$

The reduction rule carries out the substitution of derivations explained above.

$$\dfrac{\dfrac{\begin{array}{c}\overline{A}^{\,u}\\ \mathcal{D}\\ B\end{array}}{A \supset B}\supset\mathrm{I}^u \qquad \begin{array}{c}\mathcal{E}\\ A\end{array}}{B}\supset\mathrm{E} \quad \Longrightarrow_L \quad \begin{array}{c}\mathcal{E}\\ \overline{A}^{\,u}\\ \mathcal{D}\\ B\end{array}$$

The final derivation depends on all the assumptions of $\mathcal{E}$ and $\mathcal{D}$ except u, for which we have substituted $\mathcal{E}$. An alternative notation for this substitution of derivations for assumptions as introduced in Chapter 5 is $[\mathcal{E}/u]\mathcal{D} :: \vdash B$. The local reduction described above may significantly increase the overall length of the derivation, since the deduction $\mathcal{E}$ is substituted for each occurrence of the assumption labeled u in $\mathcal{D}$ and may thus be replicated many times.

Disjunction. $A \vee B$ should be derivable if either A is derivable or B is derivable. Therefore we have two introduction rules.

$$\dfrac{A}{A \vee B}\vee\mathrm{I_L} \qquad \dfrac{B}{A \vee B}\vee\mathrm{I_R}$$

If we have an assumption $A \vee B$, we do not know how it might be inferred. That is, a proposed elimination rule

$$\frac{A \vee B}{A}\,?$$

would be incorrect, since a deduction of the form

$$\cfrac{\cfrac{\begin{array}{c}\mathcal{E}\\ B\end{array}}{A \vee B}\,\vee\mathrm{I_R}}{A}\,?$$

cannot be reduced. As a consequence, the system would be *inconsistent*: if we have at least one theorem (B, in the example) we can prove every formula (A, in the example). How do we use the assumption $A \vee B$ in informal reasoning? We often proceed with a proof by cases: we prove a conclusion C under the assumption A and also show C under the assumption B. We then conclude C, since either A or B by assumption. Thus the elimination rule employs two hypothetical judgments.

$$\frac{A \vee B \qquad \begin{array}{c}\overline{A}\,u_1\\ \vdots\\ C\end{array} \qquad \begin{array}{c}\overline{B}\,u_2\\ \vdots\\ C\end{array}}{C}\,\vee\mathrm{E}^{u_1,u_2}$$

Now one can see that the introduction and elimination rules match up in two reductions. First, the case that the disjunction was inferred by $\vee\mathrm{I_L}$.

$$\cfrac{\cfrac{\begin{array}{c}\mathcal{D}\\ A\end{array}}{A \vee B}\,\vee\mathrm{I_L} \qquad \begin{array}{c}\overline{A}\,u_1\\ \mathcal{E}_1\\ C\end{array} \qquad \begin{array}{c}\overline{B}\,u_2\\ \mathcal{E}_2\\ C\end{array}}{C}\,\vee\mathrm{E}^{u_1,u_2} \quad \Longrightarrow_L \quad \begin{array}{c}\mathcal{D}\\ \overline{A}\,u_1\\ \mathcal{E}_1\\ C\end{array}$$

The other reduction is symmetric.

$$\cfrac{\cfrac{\begin{array}{c}\mathcal{D}\\ B\end{array}}{A \vee B}\,\vee\mathrm{I_R} \qquad \begin{array}{c}\overline{A}\,u_1\\ \mathcal{E}_1\\ C\end{array} \qquad \begin{array}{c}\overline{B}\,u_2\\ \mathcal{E}_2\\ C\end{array}}{C}\,\vee\mathrm{E}^{u_1,u_2} \quad \Longrightarrow_L \quad \begin{array}{c}\mathcal{D}\\ \overline{B}\,u_2\\ \mathcal{E}_2\\ C\end{array}$$

As in the reduction for implication, the resulting derivation may be longer than the original one.

Negation. In order to derive $\neg A$ we assume A and try to derive a contradiction. Thus it seems that negation requires falsehood, and, indeed, in most literature on constructive logic, $\neg A$ is seen as an abbreviation of $A \supset \bot$. In order to give a self-contained explanation of negation by an introduction rule, we employ a judgment that is parametric in a propositional parameter p: If we can derive *any* p from the hypothesis A we conclude $\neg A$.

$$\dfrac{\begin{array}{c}\overline{A}\,u \\ \vdots \\ p\end{array}}{\neg A}\,\neg\mathrm{I}^{p,u} \qquad\qquad \dfrac{\neg A \qquad A}{C}\,\neg\mathrm{E}$$

The elimination rule follows from this view: if we know $\neg A$ and A then we can conclude any formula C. In the form of a local reduction:

$$\dfrac{\dfrac{\begin{array}{c}\overline{A}\,u \\ \mathcal{D} \\ p\end{array}}{\neg A}\,\neg\mathrm{I}^{p,u} \qquad \begin{array}{c}\mathcal{E} \\ A\end{array}}{C}\,\neg\mathrm{E} \qquad \Longrightarrow_L \qquad \begin{array}{c}\dfrac{\mathcal{E}}{A}\,u \\ {[C/p]\mathcal{D}} \\ C\end{array}$$

The substitution $[C/p]\mathcal{D}$ is valid, since $\mathcal{D}$ is parametric in p.

Truth. There is only an introduction rule for $\top$:

$$\overline{\top}\,\top\mathrm{I}$$

Since we put no information into the proof of $\top$, we know nothing new if we have an assumption $\top$ and therefore we have no elimination rule and no reduction. It may also be helpful to think of $\top$ as a 0-ary conjunction: the introduction rule has 0 premisses instead of 2 and we correspondingly have 0 elimination rules instead of 2.

Falsehood. Since we should not be able to derive falsehood, there is no introduction rule for $\bot$. Therefore, if we can derive falsehood, we can derive everything.

$$\dfrac{\bot}{C}\,\bot\mathrm{E}$$

Note that there is no local reduction rule for $\bot$E. It may be helpful to think of $\bot$ as a 0-ary disjunction: we have 0 instead of 2 introduction rules and we correspondingly have to consider 0 cases instead of 2 in the elimination rule. Even though we postulated that falsehood should not be derivable, falsehood could clearly be a consequence of contradictory assumption. For example, $\vdash A \wedge \neg A \supset \bot$ is derivable.

Universal Quantification. Under which circumstances should we be able to derive $\forall x.\ A$? This clearly depends on the domain of quantification. For example, if we know that x ranges over the natural numbers, then we can conclude $\forall x.\ A$ if we can prove $[0/x]A$, $[1/x]A$, *etc.* Such a rule is not effective, since it has infinitely many premisses. Thus one usually retreats to rules such as induction. However, in a general treatment of predicate logic we would like to prove statements which are true for *all* domains of quantification. Thus we can only say that $\forall x.\ A$ should be provable if $[a/x]A$ is provable for a new parameter a about which we can make no assumption. Conversely, if we know $\forall x.\ A$, we know that $[t/x]A$ for any term t.

$$\frac{[a/x]A}{\forall x.\ A}\,\forall\mathrm{I}^a \qquad\qquad \frac{\forall x.\ A}{[t/x]A}\,\forall\mathrm{E}$$

The superscript on the inference rules is a reminder the parameter a must be "new", that is, it may not occur in any uncancelled assumption in the proof of $[a/x]A$ or in $\forall x.\ A$ itself. In other words, the derivation of the premiss must parametric in a. The local reduction carries out the substitution for the parameter.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ [a/x]A\end{array}}{\forall x.\ A}\,\forall\mathrm{I}}{[t/x]A}\,\forall\mathrm{E} \quad \Longrightarrow_L \quad \begin{array}{c}[t/a]\mathcal{D}\\ [t/x]A\end{array}$$

Here, $[t/a]\mathcal{D}$ is our notation for the result of substituting t for the parameter a throughout the deduction $\mathcal{D}$. For this substitution to preserve the conclusion, we must know that a does not already occur in A. Similarly, we would change the assumptions if a occurred free in any of the undischarged hypotheses of $\mathcal{D}$. This might render a larger proof incorrect. As an example, consider the formula $\forall x.\ \forall y.\ P(x) \supset P(y)$ which should clearly not be derivable for all predicates P. The following is *not* a deduction of this formula.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{P(a)}\,u}{\forall x.\,P(x)}\,\forall\mathrm{I}^a?}{P(b)}\,\forall\mathrm{E}}{P(a)\supset P(b)}\,{\supset}\mathrm{I}^u}{\forall y.\,P(a)\supset P(y)}\,\forall\mathrm{I}^b}{\forall x.\,\forall y.\,P(x)\supset P(y)}\,\forall\mathrm{I}^a$$

The flaw is at the inference marked with "?," where a is free in the assumption u. Applying a local proof reduction to the (incorrect) $\forall$I inference followed by $\forall$E leads to the the assumption $[b/a]P(a)$ which is equal to $P(b)$. The resulting derivation

$$\dfrac{\dfrac{\dfrac{\overline{P(b)}\,u}{P(a)\supset P(b)}\,{\supset}\mathrm{I}^u}{\forall y.\,P(a)\supset P(y)}\,\forall\mathrm{I}^b}{\forall x.\,\forall y.\,P(x)\supset P(y)}\,\forall\mathrm{I}^a$$

is once again incorrect since the hypothesis labelled u should read $P(a)$, not $P(b)$.

Existential Quantification. To conclude that $\exists x.\,A$ is true, we must know that there is a t such that $[t/x]A$ is true. Thus,

$$\dfrac{[t/x]A}{\exists x.\,A}\,\exists\mathrm{I}$$

When we have an assumption $\exists x.\,A$ we do not know for which t it is the case that $[t/x]A$ holds. We can only assume that $[a/x]A$ holds for some parameter a about which we know nothing else. Thus the elimination rule resembles the one for disjunction.

$$\dfrac{\exists x.\,A \qquad \begin{array}{c}\overline{[a/x]A}\,u\\ \vdots\\ C\end{array}}{C}\,\exists\mathrm{E}^{a,u}$$

The restriction is similar to the one for $\forall$I: the parameter a must be new, that is, it must not occur in $\exists x.\,A$, C, or any assumption employed in the derivation of the second premiss. In the reduction rule we have to perform two substitutions: we have to substitute t for the parameter a and we also have to substitute for the hypothesis labelled u.

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ [t/x]A\end{matrix}}{\exists x.\, A}\,\exists\mathrm{I} \qquad \begin{matrix}\overline{[a/x]A}^{\,u}\\ \mathcal{E}\\ C\end{matrix}}{C}\,\exists\mathrm{E}^{a,u} \quad \Longrightarrow_L \quad \begin{matrix}\overline{\;\mathcal{D}\;}^{\,u}\\ [t/x]A\\ [t/a]\mathcal{E}\\ C\end{matrix}$$

The proviso on occurrences of a guarantees that the conclusion and hypotheses of $[t/a]\mathcal{E}$ have the correct form.

Classical Logic. The inference rules so far only model *intuitionistic logic*, and some classically true formulas such as $A \vee \neg A$ (for an arbitrary A) are not derivable (see Exercise 7.10). There are three commonly used ways one can construct a system of *classical natural deduction* by adding one additional rule of inference. $\perp_C$ is called *Proof by Contradiction* or *Rule of Indirect Proof*, $\neg\neg_C$ is the *Double Negation Rule*, and XM is referred to as *Excluded Middle*.

$$\dfrac{\begin{matrix}\overline{\neg A}^{\,u}\\ \vdots\\ \perp\end{matrix}}{A}\,\perp_C{}^{u} \qquad \dfrac{\neg\neg A}{A}\,\neg\neg_C \qquad \overline{A \vee \neg A}\,\mathrm{XM}$$

The rule for classical logic (whichever one chooses to adopt) breaks the pattern of introduction and elimination rules. One can still formulate some reductions for classical inferences, but natural deduction is at heart an intuitionistic calculus. The symmetries of classical logic are much better exhibited in sequent formulations of the logic. In Exercise 7.2 we explore the three ways of extending the intuitionistic proof system and show that they are equivalent.

Here is a simple example of a natural deduction. We attempt to show the process by which such a deduction may have been generated, as well as the final deduction. The three vertical dots indicate a gap in the derivation we are trying to construct, with assumptions and their consequences shown above and the desired conclusion below the gap.

$$\begin{matrix}\vdots\\ A \wedge (A \supset B) \supset B\end{matrix} \quad \leadsto \quad \dfrac{\begin{matrix}\overline{A \wedge (A \supset B)}^{\,u}\\ \vdots\\ B\end{matrix}}{A \wedge (A \supset B) \supset B}\,\supset\mathrm{I}^{u}$$

$$\leadsto \quad \dfrac{\begin{array}{c}\dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A}\,{\wedge}\mathrm{E_L} \\ \vdots \\ B\end{array}}{A \wedge (A \supset B) \supset B}\,{\supset}\mathrm{I}^u \quad \leadsto \quad \dfrac{\begin{array}{c}\dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A}\,{\wedge}\mathrm{E_L} \qquad \dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A \supset B}\,{\wedge}\mathrm{E_R} \\ \vdots \\ B\end{array}}{A \wedge (A \supset B) \supset B}\,{\supset}\mathrm{I}^u$$

$$\leadsto \quad \dfrac{\begin{array}{c}\dfrac{\dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A \supset B}\,{\wedge}\mathrm{E_R} \qquad \dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A}\,{\wedge}\mathrm{E_L}}{B}\,{\supset}\mathrm{E} \\ \vdots \\ B\end{array}}{A \wedge (A \supset B) \supset B}\,{\supset}\mathrm{I}^u$$

$$\leadsto \quad \dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A \supset B}\,{\wedge}\mathrm{E_R} \qquad \dfrac{\dfrac{}{A \wedge (A \supset B)}\,u}{A}\,{\wedge}\mathrm{E_L}}{B}\,{\supset}\mathrm{E}}{A \wedge (A \supset B) \supset B}\,{\supset}\mathrm{I}^u$$

The symbols A and B in this derivation stand for arbitrary formulas; we can thus view the derivation generated below as being parametric in A and B. In other words, every instance of this derivation (replacing A and B by arbitrary formulas) is a valid derivation.

Below is a summary of the rules of intuitionistic natural deduction.

Introduction Rules   Elimination Rules

$$\frac{A \qquad B}{A \wedge B}\;\wedge\mathrm{I} \qquad\qquad \frac{A \wedge B}{A}\;\wedge\mathrm{E_L} \qquad \frac{A \wedge B}{B}\;\wedge\mathrm{E_R}$$

$$\frac{A}{A \vee B}\;\vee\mathrm{I_L} \qquad \frac{B}{A \vee B}\;\vee\mathrm{I_R} \qquad\qquad \frac{A \vee B \qquad \begin{array}{c}\overline{A}\,^{u_1}\\ \vdots \\ C\end{array} \qquad \begin{array}{c}\overline{B}\,^{u_2}\\ \vdots \\ C\end{array}}{C}\;\vee\mathrm{E}^{u_1,u_2}$$

$$\frac{\begin{array}{c}\overline{A}\,^{u}\\ \vdots \\ B\end{array}}{A \supset B}\;\supset\mathrm{I}^{u} \qquad\qquad \frac{A \supset B \qquad A}{B}\;\supset\mathrm{E}$$

$$\frac{\begin{array}{c}\overline{A}\,^{u}\\ \vdots \\ p\end{array}}{\neg A}\;\neg\mathrm{I}^{p,u} \qquad\qquad \frac{A \qquad \neg A}{C}\;\neg\mathrm{E}$$

$$\frac{}{\top}\;\top\mathrm{I}$$

$$\frac{\bot}{C}\;\bot\mathrm{E}$$

$$\frac{[a/x]A}{\forall x.\ A}\;\forall\mathrm{I}^{*} \qquad\qquad \frac{\forall x.\ A}{[t/x]A}\;\forall\mathrm{E}$$

$$\frac{[t/x]A}{\exists x.\ A}\;\exists\mathrm{I} \qquad\qquad \frac{\exists x.\ A \qquad \begin{array}{c}\overline{[a/x]A}\,^{u}\\ \vdots \\ C\end{array}}{C}\;\exists\mathrm{E}^{a,u}$$

7.2 Representation in LF

The LF logical framework and its implementation in Elf are ideally suited to the representation of natural deduction, and we will exploit a number of the encoding techniques we have encountered so far.

The representation of terms and formulas employs the idea of higher-order abstract syntax so that substitution and side-conditions on variable occurrences required for the inference rules can be expressed directly. Recall the basic principle of higher-order abstract syntax: represent object-level variables by meta-level variables. As a consequence, object-language constructs that bind variables are represented by meta-level constructs that also bind variables. Parameters play the role of variables in derivations and are thus also represented as variables in the meta-language. As in Section 3.2, we write the representation function as $\ulcorner\cdot\urcorner$. First, we declare the type of individuals (i) and the type of formulas (o).

$$\begin{array}{lcl} \mathrm{i} & : & \mathrm{type} \\ \mathrm{o} & : & \mathrm{type} \end{array}$$

The structure of the domain of individuals is left open in the development of predicate logic. Commitment to, say, natural numbers then leads to a formalization of arithmetic. Our encoding reflects this approach: we do not specify that any particular individuals exist, but we can give the recipe by which function symbols can be added to our encoding. For every function symbol f of arity n, we add a corresponding declaration

$$\mathrm{f} \quad : \quad \underbrace{\mathrm{i} \to \cdots \to \mathrm{i} \to}_{n} \mathrm{i}.$$

The representation of terms is then given by

$$\begin{array}{rcl} \ulcorner x \urcorner & = & x \\ \ulcorner a \urcorner & = & a \\ \ulcorner f(t_1, \ldots, t_n) \urcorner & = & \mathrm{f}\, \ulcorner t_1 \urcorner \ldots \ulcorner t_n \urcorner. \end{array}$$

Note that each parameter a of predicate logic is mapped to an LF variable with the same name.

This kind of encoding takes advantage of the open-ended nature of signatures: we can always add further declarations without invalidating judgments made earlier. Predicate symbols are dealt with in a similar manner: The general recipe is to add a declaration

$$\mathrm{p} \quad : \quad \underbrace{\mathrm{i} \to \cdots \to \mathrm{i} \to}_{n} \mathrm{o}$$

for every predicate symbol P of arity n. The representation function and the remaining declarations are then straightforward.

$$
\begin{array}{rcllcl}
\ulcorner P(t_1,\ldots,t_n)\urcorner & = & \mathrm{p}\,\ulcorner t_1\urcorner\ldots\ulcorner t_n\urcorner & & & \\
\ulcorner A_1 \wedge A_2\urcorner & = & \mathrm{and}\,\ulcorner A_1\urcorner\,\ulcorner A_2\urcorner & \mathrm{and} & : & \mathrm{o}\to\mathrm{o}\to\mathrm{o} \\
\ulcorner A_1 \supset A_2\urcorner & = & \mathrm{imp}\,\ulcorner A_1\urcorner\,\ulcorner A_2\urcorner & \mathrm{imp} & : & \mathrm{o}\to\mathrm{o}\to\mathrm{o} \\
\ulcorner A_1 \vee A_2\urcorner & = & \mathrm{or}\,\ulcorner A_1\urcorner\,\ulcorner A_2\urcorner & \mathrm{or} & : & \mathrm{o}\to\mathrm{o}\to\mathrm{o} \\
\ulcorner \neg A\urcorner & = & \mathrm{not}\,\ulcorner A\urcorner & \mathrm{not} & : & \mathrm{o}\to\mathrm{o} \\
\ulcorner \bot\urcorner & = & \mathrm{false} & \mathrm{false} & : & \mathrm{o} \\
\ulcorner \top\urcorner & = & \mathrm{true} & \mathrm{true} & : & \mathrm{o} \\
\ulcorner \forall x.\,A\urcorner & = & \mathrm{forall}\,(\lambda x{:}\mathrm{i}.\,\ulcorner A\urcorner) & \mathrm{forall} & : & (\mathrm{i}\to\mathrm{o})\to\mathrm{o} \\
\ulcorner \exists x.\,A\urcorner & = & \mathrm{exists}\,(\lambda x{:}\mathrm{i}.\,\ulcorner A\urcorner) & \mathrm{exists} & : & (\mathrm{i}\to\mathrm{o})\to\mathrm{o}
\end{array}
$$

The formulation of an adequacy theorem for this representation is left to the reader (see Exercise 7.4). We only note the substitution property which holds due to the use of higher-order abstract syntax:

$$\ulcorner [t/x]A\urcorner = [\ulcorner t\urcorner/x]\ulcorner A\urcorner \equiv (\lambda x{:}\mathrm{i}.\,\ulcorner A\urcorner)\,\ulcorner t\urcorner.$$

The representation of the derivability judgment of natural deduction follows the schema of Chapter 5, since natural deduction makes essential use of parametric and hypothetical judgments. We introduce a type family nd that is indexed by a formula. The LF type nd $\ulcorner A\urcorner$ is intended to represent the type of natural deductions of the formula A.

$$\mathrm{nd} : \mathrm{o} \to \mathrm{type}$$

Each inference rule is represented by an LF constant which can be thought of as a function from a derivation of the premisses of the rule to a derivation of the conclusion. The constant must further be applied to the representation of the formulas participating in an inference in order to avoid possible ambiguities.

Conjunction.

$$\ulcorner \dfrac{\begin{matrix}\mathcal{D} \\ A\end{matrix} \qquad \begin{matrix}\mathcal{E} \\ B\end{matrix}}{A \wedge B}\wedge\mathrm{I} \urcorner = \mathrm{andi}\,\ulcorner A\urcorner\,\ulcorner B\urcorner\,\ulcorner \mathcal{D}\urcorner\,\ulcorner \mathcal{E}\urcorner$$

from which it follows that we declare

$$\mathrm{andi} \;:\; \Pi A{:}\mathrm{o}.\,\Pi B{:}\mathrm{o}.\,\mathrm{nd}\,A \to \mathrm{nd}\,B \to \mathrm{nd}\,(\mathrm{and}\,A\,B).$$

For derivations ending in one of the two elimination rules we have the similarly obvious representations

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\end{array}}{A}\wedge\mathrm{E_L} \urcorner = \mathrm{andel}\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ \ulcorner \mathcal{D} \urcorner$$

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\end{array}}{B}\wedge\mathrm{E_R} \urcorner = \mathrm{ander}\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ \ulcorner \mathcal{D} \urcorner$$

where

$$\begin{array}{lcl} \mathrm{andel} & : & \Pi A{:}o.\ \Pi B{:}o.\ \mathrm{nd}\ (\mathrm{and}\ A\ B) \to \mathrm{nd}\ A \\ \mathrm{ander} & : & \Pi A{:}o.\ \Pi B{:}o.\ \mathrm{nd}\ (\mathrm{and}\ A\ B) \to \mathrm{nd}\ B \end{array}$$

Implication. The introduction rule for implication is somewhat less straightforward, since it employs a hypothetical judgment. The derivation of the hypothetical judgment in the premiss is represented as a function which, when applied to a derivation of A, yields a derivation of B.

$$\ulcorner \dfrac{\begin{array}{c}\overline{A}\,u\\ \mathcal{D}\\ B\end{array}}{A \supset B}\supset\mathrm{I}^u \urcorner = \mathrm{impi}\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ (\lambda u{:}\mathrm{nd}\ \ulcorner A \urcorner.\ \ulcorner \mathcal{D} \urcorner)$$

The assumption A labelled by u which may be used in the derivaton $\mathcal{D}$ is represented by the LF variable u which ranges over derivations of A.

$$\ulcorner \overline{A}\,u \urcorner = u$$

The elimination rule is simpler, since it does not involve a hypothetical judgment. The representation of a derivation ending in the elimination rule is defined by

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D}\\ A \supset B\end{array} \qquad \begin{array}{c}\mathcal{E}\\ A\end{array}}{B}\supset\mathrm{E} \urcorner = \mathrm{impe}\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ \ulcorner \mathcal{D} \urcorner\ \ulcorner \mathcal{E} \urcorner$$

where

$$\text{impe} \quad : \quad \Pi A{:}\mathrm{o}.\ \Pi B{:}\mathrm{o}.\ \mathrm{nd}\ (\mathrm{imp}\ A\ B) \to \mathrm{nd}\ A \to \mathrm{nd}\ B.$$

As an example which requires only conjunction and implication, consider the derivation of $A \wedge (A \supset B) \supset B$ from Page 215:

$$\dfrac{\dfrac{\dfrac{}{A \wedge (A \supset B)}u}{A \supset B}\wedge\mathrm{E_R} \qquad \dfrac{\dfrac{}{A \wedge (A \supset B)}u}{A}\wedge\mathrm{E_L}}{\dfrac{B}{A \wedge (A \supset B) \supset B}\supset\mathrm{I}^u}\supset\mathrm{E}$$

This derivation is represented by the LF object

$$\begin{array}{l}
\mathrm{impi}\ (\mathrm{and}\ \ulcorner A\urcorner\ (\mathrm{imp}\ \ulcorner A\urcorner\ \ulcorner B\urcorner))\ \ulcorner B\urcorner \\
\quad (\lambda u{:}\mathrm{nd}\ (\mathrm{and}\ \ulcorner A\urcorner\ (\mathrm{imp}\ \ulcorner A\urcorner\ \ulcorner B\urcorner)). \\
\qquad (\mathrm{impe}\ \ulcorner A\urcorner\ \ulcorner B\urcorner \\
\qquad\quad (\mathrm{ander}\ \ulcorner A\urcorner\ (\mathrm{imp}\ \ulcorner A\urcorner\ \ulcorner B\urcorner)\ u) \\
\qquad\quad (\mathrm{andel}\ \ulcorner A\urcorner\ (\mathrm{imp}\ \ulcorner A\urcorner\ \ulcorner B\urcorner)\ u)))
\end{array}$$

which has type

$$\mathrm{nd}\ (\mathrm{imp}\ (\mathrm{and}\ \ulcorner A\urcorner\ (\mathrm{imp}\ \ulcorner A\urcorner\ \ulcorner B\urcorner))\ \ulcorner B\urcorner).$$

This example shows clearly some redundancies in the representation of the deduction (there are many occurrence of $\ulcorner A\urcorner$ and $\ulcorner B\urcorner$). The Elf implementation of natural deduction in Section 7.3 eliminates some of these redundancies.

Disjunction. The representation of the introduction and elimination rules for disjunction employ the same techniques as we have seen above.

$$\dfrac{A}{A \vee B}\vee\mathrm{I_L} \qquad \dfrac{B}{A \vee B}\vee\mathrm{I_R} \qquad \dfrac{A \vee B \quad \begin{array}{c}\overline{A}^{\,u_1} \\ \vdots \\ C\end{array} \quad \begin{array}{c}\overline{B}^{\,u_2} \\ \vdots \\ C\end{array}}{C}\vee\mathrm{E}^{u_1,u_2}$$

The corresponding LF constants:

$$\begin{array}{lcl}
\mathrm{oril} & : & \Pi A{:}\mathrm{o}.\ \Pi B{:}\mathrm{o}.\ \mathrm{nd}\ A \to \mathrm{nd}\ (\mathrm{or}\ A\ B) \\
\mathrm{orir} & : & \Pi A{:}\mathrm{o}.\ \Pi B{:}\mathrm{o}.\ \mathrm{nd}\ B \to \mathrm{nd}\ (\mathrm{or}\ A\ B) \\
\mathrm{ore} & : & \Pi A{:}\mathrm{o}.\ \Pi B{:}\mathrm{o}.\ \Pi C{:}\mathrm{o}. \\
 & & \quad \mathrm{nd}\ (\mathrm{or}\ A\ B) \to (\mathrm{nd}\ A \to \mathrm{nd}\ C) \to (\mathrm{nd}\ B \to \mathrm{nd}\ C) \to \mathrm{nd}\ C.
\end{array}$$

Negation. The introduction and elimination rules for negation and their representation follow the pattern of the rules for implication.

$$\dfrac{\begin{array}{c}\overline{A}\,u\\ \vdots\\ p\end{array}}{\neg A}\,\neg\mathrm{I}^{p,u} \qquad\qquad \dfrac{\neg A \qquad A}{C}\,\neg\mathrm{E}$$

$$\begin{array}{lcl}\mathrm{noti} & : & \Pi A{:}\mathrm{o}.\ (\Pi p{:}\mathrm{o}.\ \mathrm{nd}\ A \to \mathrm{nd}\ p) \to \mathrm{nd}\ (\mathrm{not}\ A)\\ \mathrm{note} & : & \Pi A{:}\mathrm{o}.\ \mathrm{nd}\ (\mathrm{not}\ A) \to \Pi C{:}\mathrm{o}.\ \mathrm{nd}\ A \to \mathrm{nd}\ C\end{array}$$

Truth. There is only an introduction rule, which is easily represented. We have

$$\ulcorner \dfrac{}{\top}\,\top\mathrm{I} \urcorner = \mathrm{truei}$$

where

$$\mathrm{truei} \quad : \quad \mathrm{nd}\ (\mathrm{true}).$$

Falsehood. There is only an elimination rule, which is easily represented. We have

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D}\\ \bot\end{array}}{C}\,\bot\mathrm{E} \urcorner = \mathrm{falsee}\ \ulcorner C \urcorner\ \ulcorner \mathcal{D} \urcorner$$

where

$$\mathrm{falsee} \quad : \quad \Pi C{:}\mathrm{o}.\ \mathrm{nd}\ (\mathrm{false}) \to \mathrm{nd}\ C.$$

Universal Quantification. The introduction rule for the universal quantifier employs a parametric judgment and the elimination rule employs substitution.

$$\dfrac{[a/x]A}{\forall x.\ A}\,\forall\mathrm{I}^{a} \qquad\qquad \dfrac{\forall x.\ A}{[t/x]A}\,\forall\mathrm{E}$$

The side condition on $\forall$I states that the parameter a must be "new". In the spirit of Chapter 5 the derivation of a parametric judgment will be represented as a function of the parameter. Recall that $\ulcorner \forall x.\ A \urcorner = \text{forall}\ (\lambda x{:}\text{i}.\ \ulcorner A \urcorner)$.

$$\ulcorner \begin{array}{c} \mathcal{D} \\ [a/x]A \\ \hline \forall x.\ A \end{array}\ \forall\text{I}^a \urcorner \quad = \text{foralli}\ (\lambda x{:}\text{i}.\ \ulcorner A \urcorner)\ (\lambda a{:}\text{i}.\ \ulcorner \mathcal{D} \urcorner)$$

Note that $\ulcorner A \urcorner$, the representation of A, has a free variable x which must be bound in the meta-language, so that the representing object does not have a free variable x. This representation determines the type of the constant foralli.

$$\text{foralli} \quad : \quad \Pi A{:}\text{i} \to \text{o}.\ (\Pi a{:}\text{i}.\ \text{nd}\ (A\ a)) \to \text{nd}\ (\text{forall}\ A)$$

In an application of this constant, the argument labelled A will be $\lambda x{:}\text{i}.\ \ulcorner A \urcorner$ and $(A\ a)$ will be $(\lambda x{:}\text{i}.\ \ulcorner A \urcorner)\ a$ which is equivalent to $[a/x]\ulcorner A \urcorner$ which in turn is equivalent to $\ulcorner [a/x]A \urcorner$ by the substitution property on Page 218.

The elimination rule does not employ a hypothetical judgment.

$$\ulcorner \begin{array}{c} \mathcal{D} \\ \forall x.\ A \\ \hline [t/x]A \end{array}\ \forall\text{E} \urcorner \quad = \text{foralle}\ (\lambda x{:}\text{i}.\ \ulcorner A \urcorner)\ \ulcorner \mathcal{D} \urcorner\ \ulcorner t \urcorner$$

The substitution of t for x in A is representation by the application of the function $(\lambda x{:}\text{i}.\ \ulcorner A \urcorner)$ (the first argument of foralle) to $\ulcorner t \urcorner$.

$$\text{foralle} \quad : \quad \Pi A{:}\text{i} \to \text{o}.\ \text{nd}\ (\text{forall}\ A) \to \Pi t{:}\text{i}.\ \text{nd}\ (A\ t)$$

We now check that

$$\ulcorner \begin{array}{c} \mathcal{D} \\ \forall x.\ A \\ \hline [t/x]A \end{array}\ \forall\text{E} \urcorner \quad : \text{nd}\ \ulcorner [t/x]A \urcorner,$$

assuming that $\ulcorner \mathcal{D} \urcorner : \text{nd}\ \ulcorner \forall x.\ A \urcorner$. This would be an important part in the proof of adequacy of this representation of natural deductions. First we note that the arguments have the expected types and find that

$$\begin{array}{rcl} \text{foralle} & : & \Pi A{:}\text{i} \to \text{o}.\ \text{nd}\ (\text{forall}\ A) \to \Pi t{:}\text{i}.\ \text{nd}\ (A\ t) \\ \text{foralle}\ (\lambda x{:}\text{i}.\ \ulcorner A \urcorner) & : & \text{nd}\ (\text{forall}\ (\lambda x{:}\text{i}.\ \ulcorner A \urcorner)) \to \Pi t{:}\text{i}.\ \text{nd}\ ((\lambda x{:}\text{i}.\ \ulcorner A \urcorner)\ t) \\ \text{foralle}\ (\lambda x{:}\text{i}.\ \ulcorner A \urcorner)\ \ulcorner \mathcal{D} \urcorner & : & \Pi t{:}\text{i}.\ \text{nd}\ ((\lambda x{:}\text{i}.\ \ulcorner A \urcorner)\ t) \\ \text{foralle}\ (\lambda x{:}\text{i}.\ \ulcorner A \urcorner)\ \ulcorner \mathcal{D} \urcorner\ \ulcorner t \urcorner & : & \text{nd}\ ((\lambda x{:}\text{i}.\ \ulcorner A \urcorner)\ \ulcorner t \urcorner). \end{array}$$

Now we have to recall the rule of type conversion for LF (see Section 3.8)

$$\frac{\Gamma \vdash_\Sigma M : A \qquad A \equiv B \qquad \Gamma \vdash_\Sigma B : \text{type}}{\Gamma \vdash_\Sigma M : B}\,\mathsf{conv}$$

and note that

$$(\lambda x{:}\mathrm{i}.\ \ulcorner A \urcorner)\ \ulcorner t \urcorner \equiv [\ulcorner t \urcorner / x]\ulcorner A \urcorner$$

by β-conversion. Furthermore, by the substitution property for the representation we have

$$[\ulcorner t \urcorner / x]\ulcorner A \urcorner = \ulcorner [t/x]A \urcorner$$

which yields the desired

$$\text{foralle}\ (\lambda x{:}\mathrm{i}.\ \ulcorner A \urcorner)\ \ulcorner \mathcal{D} \urcorner\ \ulcorner t \urcorner : \text{nd}\ (\ulcorner [t/x]A \urcorner).$$

Existential Quantification. The representation techniques are the same we used for universal quantification: parametric and hypothetical derivations are represented as LF functions.

$$\ulcorner \dfrac{\begin{array}{c}\mathcal{D}\\ {[t/x]A}\end{array}}{\exists x.\ A}\,\exists\mathrm{I} \urcorner = \text{existsi}\ (\lambda x{:}\mathrm{i}.\ \ulcorner A \urcorner)\ \ulcorner t \urcorner\ \ulcorner \mathcal{D} \urcorner$$

$$\ulcorner \dfrac{\exists x.\ A \qquad \begin{array}{c}\overline{[a/x]A}^{\,u}\\ \mathcal{D}\\ C\end{array}}{C}\,\exists\mathrm{E}^{a,u} \urcorner = \text{existse}\ (\lambda x{:}\mathrm{i}\ \ulcorner A \urcorner)\ \ulcorner C \urcorner\ (\lambda a{:}\mathrm{i}.\ \lambda u{:}\text{nd}\ \ulcorner [a/x]A \urcorner.\ \ulcorner \mathcal{D} \urcorner)$$

where

$$\begin{array}{lcl}
\text{existsi} & : & \Pi A{:}\mathrm{i} \to \mathrm{o}.\ \Pi t{:}\mathrm{i}.\ \text{nd}\ (A\ t) \to \text{nd}\ (\text{exists}\ A)\\
\text{existse} & : & \Pi A{:}\mathrm{i} \to \mathrm{o}.\ \Pi C{:}\mathrm{o}.\ \text{nd}\ (\text{exists}\ A) \to (\Pi a{:}\mathrm{i}.\ \text{nd}\ (A\ a) \to \text{nd}\ C) \to \text{nd}\ C.
\end{array}$$

Once again, we will not formally state or proof the adequacy theorem for this encoding. We only mention the three substitution properties which will be important in the formalization of reduction in the next section.

$$\ulcorner [t/a]\mathcal{D} \urcorner = [\ulcorner t \urcorner / a]\ulcorner \mathcal{D} \urcorner \quad \text{and} \quad \ulcorner [C/p]\mathcal{D} \urcorner = [\ulcorner C \urcorner / p]\ulcorner \mathcal{D} \urcorner \quad \text{and} \quad \ulcorner [\mathcal{E}/u]\mathcal{D} \urcorner = [\ulcorner \mathcal{E} \urcorner / u]\ulcorner \mathcal{D} \urcorner.$$

Each of the rules that may be added to obtain classical logic can be easily represented with the techniques from above. They are left as Exercise 7.7.

7.3 Implementation in Elf

In this section we summarize the LF encoding of natural deduction from the previous section as an Elf signature, and also give the representation of the local reduction rules from Section 7.1. We will make a few cosmetic changes that reflect common Elf programming practice. The first is the use of infix and prefix notation in the use of the logical connectives. According to our conventions, conjunction, disjunction, and implication are all right associative, and conjunction and disjunction bind stronger than implication. Negation is treated as a prefix operator binding tighter than the binary connectives. For example,

$$A \wedge (A \supset B) \supset \neg B \supset C$$

is the same formula as

$$(A \wedge (A \supset B)) \supset ((\neg B) \supset C).$$

As an arbitrary baseline in the pragmas below we pick a binding strength of 10 for implication.

```
i : type.  % individuals
o : type.  % formulas
%name i T S
%name o A B C

and    : o -> o -> o.  %infix right 11 and
imp    : o -> o -> o.  %infix right 10 imp
or     : o -> o -> o.  %infix right 11 or
not    : o -> o.       %prefix 12 not
true   : o.
false  : o.
forall : (i -> o) -> o.
exists : (i -> o) -> o.
```

The `%name` declarations instruct Elf's printing routines to prefer names `T` and `S` for unnamed variables of type `i`, and names `A`, `B`, and `C` for unnamed variables of type `o`. This serves to improve the readability of Elf's output.

The second simplification in the concrete presentation is to leave some Π-quantifiers implicit. The type reconstruction algorithm always interprets free variables in a declaration as a schematic variables (which are therefore implicitly Π-quantified) and determines their type from the context in which they appear.[1]

[1] [*pointer to full discussion*]

```
nd : o -> type.  % proofs
%name nd D E

andi    : nd A -> nd B -> nd (A and B).
andel   : nd (A and B) -> nd A.
ander   : nd (A and B) -> nd B.

impi    : (nd A -> nd B) -> nd (A imp B).
impe    : nd (A imp B) -> nd A -> nd B.

oril    : nd A -> nd (A or B).
orir    : nd B -> nd (A or B).
ore     : nd (A or B) -> (nd A -> nd C) -> (nd B -> nd C) -> nd C.

noti    : ({p:o} nd A -> nd p) -> nd (not A).
note    : nd (not A) -> {C:o} nd A -> nd C.

truei   : nd (true).

falsee  : nd (false) -> nd C.

foralli : ({a:i} nd (A a)) -> nd (forall A).
foralle : nd (forall A) -> {T:i} nd (A T).

existsi : {T:i} nd (A T) -> nd (exists A).
existse : nd (exists A) -> ({a:i} nd (A a) -> nd C) -> nd C.
```

As a consequence of omitting the quantifiers on some variables in these declarations, the corresponding arguments to the constants also have to be omitted. For example, in the input language the constant `andi` now appears to take only two arguments (the representation of the derivations of `A` and `B`), rather than four like the LF constant

$$\mathrm{andi} : \Pi A{:}\mathrm{o}.\ \Pi B{:}\mathrm{o}.\ \mathrm{nd}\ A \to \mathrm{nd}\ B \to \mathrm{nd}\ (\mathrm{and}\ A\ B).$$

The type reconstruction algorithm will determine the two remaining implicit arguments from context. The derivation of $A \wedge (A \supset B) \supset B$ from Page 220 has this very concise Elf representation:

```
impi [u:nd (A and (A imp B))] (impe (ander u) (andel u))
```

where `A` and `B` are free variables of type `o`. The use of variable `A` and `B` indicates the generic nature of this derivation: we can substitute any two objects of type `o` for `A` and `B` and still obtain the representation of a valid derivation. Incidentally, in this example the type of `u` is also redundant and could also have been omitted.

Next we turn to the local reduction judgment

$$\mathcal{D} :: \vdash A \quad \Longrightarrow_L \quad \mathcal{D}' :: \vdash A.$$

We used this judgment to check that the introduction and elimination rules for each logical connective and quantifier match up. This higher-level judgment relates derivations of the same formula A. We have already seen such judgments in Section 3.7 and subsequent representations of meta-theoretic proofs. The representing LF type family would be declared as

$$\mathrm{redl} \quad : \quad \Pi A{:}\mathrm{o}.\ \mathrm{nd}\ A \to \mathrm{nd}\ A \to \mathrm{type}.$$

We will not show this representation in pure LF, but immediately give its concrete syntax in Elf.

```
==>L : nd A -> nd A -> type.  %infix none 14 ==>L
%name ==>L L
```

Note that the quantifier over `A` is once again implicit and that `==>L` must be read as one symbol.

Conjunction. The local reductions have the form

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D} & \mathcal{E}\\ A & B\end{array}}{A \wedge B}\,\wedge\mathrm{I}}{A}\,\wedge\mathrm{E_L} \quad \Longrightarrow_L \quad \begin{array}{c}\mathcal{D}\\ A\end{array}$$

and symmetrically

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D} & \mathcal{E}\\ A & B\end{array}}{A \wedge B}\,\wedge\mathrm{I}}{B}\,\wedge\mathrm{E_R} \quad \Longrightarrow_L \quad \begin{array}{c}\mathcal{E}\\ B\end{array}.$$

Because of type reconstruction, we can omit the formulas entirely from the straightforward representations of these two rules.

```
redl_andl   : (andel (andi D E)) ==>L D.
redl_andr   : (ander (andi D E)) ==>L E.
```

Implication. This reduction involves a substitution of a derivation for an assumption.

$$\dfrac{\dfrac{\begin{matrix}\overline{\;A\;}\,u\\ \mathcal{D}\\ B\end{matrix}}{A \supset B}\,{\supset}\mathrm{I}^u \qquad \begin{matrix}\mathcal{E}\\ A\end{matrix}}{B}\,{\supset}\mathrm{E} \qquad \Longrightarrow_L \qquad \begin{matrix}\mathcal{E}\\ \overline{\;A\;}\,u\\ \mathcal{D}\\ B\end{matrix}$$

The representation of the left-hand side in Elf is

```
(impe (impi D) E)
```

where E $= \ulcorner\mathcal{E}\urcorner$ and D $= \lambda u{:}\mathrm{nd}\,\ulcorner A\urcorner.\,\ulcorner\mathcal{D}\urcorner$. The derivation on the right-hand side can be written more succinctly as $[\mathcal{E}/u]\mathcal{D}$. The substitution property for derivations (see Page 7.2) yields

$$\ulcorner[\mathcal{E}/u]\mathcal{D}\urcorner = [\ulcorner\mathcal{E}\urcorner/u]\ulcorner\mathcal{D}\urcorner \equiv (\lambda u{:}\mathrm{nd}\,\ulcorner A\urcorner.\,\ulcorner\mathcal{D}\urcorner)\,\ulcorner\mathcal{E}\urcorner.$$

Thus the representation of the right-hand side will be β-equivalent to `(D E)` and we formulate the rule as

```
redl_imp    : (impe (impi D) E) ==>L (D E).
```

Disjunction. The two reductions for disjunction introduction followed by elimination are

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ A\end{matrix}}{A \vee B}\,{\vee}\mathrm{I_L} \qquad \begin{matrix}\overline{\;A\;}\,u_1\\ \mathcal{E}_1\\ C\end{matrix} \qquad \begin{matrix}\overline{\;B\;}\,u_2\\ \mathcal{E}_2\\ C\end{matrix}}{C}\,{\vee}\mathrm{E}^{u_1,u_2} \qquad \Longrightarrow_L \qquad \begin{matrix}\mathcal{D}\\ \overline{\;A\;}\,u_1\\ \mathcal{E}_1\\ C\end{matrix}$$

and

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ B\end{matrix}}{A \vee B}\,{\vee}\mathrm{I_R} \qquad \begin{matrix}\overline{\;A\;}\,u_1\\ \mathcal{E}_1\\ C\end{matrix} \qquad \begin{matrix}\overline{\;B\;}\,u_2\\ \mathcal{E}_2\\ C\end{matrix}}{C}\,{\vee}\mathrm{E}^{u_1,u_2} \qquad \Longrightarrow_L \qquad \begin{matrix}\mathcal{D}\\ \overline{\;B\;}\,u_2\\ \mathcal{E}_2\\ C\end{matrix}.$$

Their representation follows the pattern from the previous case to model the substitution of derivations.

```
redl_orl    : (ore (oril D) E1 E2) ==>L (E1 D).
redl_orr    : (ore (orir D) E1 E2) ==>L (E2 D).
```

Negation. This is similar to implication.

$$\dfrac{\dfrac{\begin{array}{c}\overline{A}^{\,u}\\ \mathcal{D}\\ p\end{array}}{\neg A}\,\neg\mathrm{I}^{p,u} \qquad \begin{array}{c}\mathcal{E}\\ A\end{array}}{C}\,\neg\mathrm{E} \quad\Longrightarrow_L\quad \begin{array}{c}\dfrac{\mathcal{E}}{A}\,u\\ {[C/p]\mathcal{D}}\\ C\end{array}$$

```
redl_not    : (note (noti D) C E) ==>L (D C E).
```

Universal Quantification. The universal introduction rule involves a parametric judgment. Consequently, the substitution to be carried out during reduction replaces a parameter by a term.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ {[a/x]A}\end{array}}{\forall x.\ A}\,\forall\mathrm{I}^a}{[t/x]A}\,\forall\mathrm{E} \quad\Longrightarrow_L\quad \begin{array}{c}{[t/a]\mathcal{D}}\\ {[t/x]A}\end{array}$$

In the representation we exploit the first part of the substitution property for derivations (see page 223):

$$\ulcorner[t/a]\mathcal{D}\urcorner = [\ulcorner t\urcorner/a]\ulcorner\mathcal{D}\urcorner \equiv (\lambda a{:}\mathrm{i}.\ \ulcorner\mathcal{D}\urcorner)\ \ulcorner t\urcorner.$$

This gives rise to the declaration

```
redl_forall : (foralle (foralli D) T) ==>L (D T).
```

Existential Quantification. This involves both a parametric and hypothetical judgments, and we combine the techniques used for implication and universal quantification.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ {[t/x]A}\end{array}}{\exists x.\ A}\,\exists\mathrm{I} \qquad \begin{array}{c}\overline{[a/x]A}^{\,u}\\ \mathcal{E}\\ C\end{array}}{C}\,\exists\mathrm{E}^{a,u} \quad\Longrightarrow_L\quad \begin{array}{c}\dfrac{\mathcal{D}}{[t/x]A}\,u\\ {[t/a]\mathcal{E}}\\ C\end{array}$$

The crucial equations for the adequacy of the encoding below are

$$\ulcorner[\mathcal{D}/u][t/a]\mathcal{E}\urcorner = [\ulcorner\mathcal{D}\urcorner/u][\ulcorner t\urcorner/a]\ulcorner\mathcal{E}\urcorner \equiv (\lambda a{:}\mathrm{i}.\ \lambda u{:}\mathrm{nd}\ \ulcorner[a/x]A\urcorner.\ \ulcorner\mathcal{E}\urcorner)\ \ulcorner t\urcorner\ \ulcorner\mathcal{D}\urcorner.$$

```
redl_exists : (existse (existsi T D) E) ==>L (E T D).
```

7.4 The Curry-Howard Isomorphism

The basic judgment of the system of natural deduction is the derivability of a formula A, written as $\vdash A$. If we wish to make the derivation explicit we write $\mathcal{D} :: \vdash A$. It has been noted by Howard [How80] that there is a strong correspondence between the (intuitionistic) derivations $\mathcal{D}$ and λ-terms. The formulas A then act as types classifying those λ-terms. In the propositional case, this correspondence is an isomorphism: formulas are isomorphic to types and derivations are isomorphic to simply-typed λ-terms. These isomorphisms are often called the *propositions-as-types* and *proofs-as-programs* paradigms.

If we stopped at this observation, we would have obtained only a fresh interpretation of familiar deductive systems, but we would not be any closer to the goal of providing a language for reasoning about properties of programs. However, the correspondences can be extended to first-order and higher-order logics. Interpreting first-order (or higher-order) formulas as types yields a significant increase in expressive power of the type system. However, maintaining an isomorphism during the generalization to first-order logic is somewhat unnatural and cumbersome. One might expect that a proof contains more information than the corresponding program. Thus the literature often talks about *extracting programs from proofs* or *contracting proofs to programs.*

The first step will be to introduce a notation for derivations to be carried along in deductions. For example, if M represents a proof of A and N represents a proof of B, then the pair $\langle M, N\rangle$ can be seen as a representation of the proof of $A \wedge B$ by $\wedge$-introduction. We write $M :\cdot A$ to express the judgment M *is a proof term for* A. We also repeat the local reductions from the previous section in the new notation.

Conjunction. The proof term for a conjunction is simply the pair of proofs of the premisses.

$$\frac{M :\cdot A \qquad N :\cdot B}{\langle M, N\rangle :\cdot A \wedge B}\wedge\mathrm{I} \qquad\qquad \frac{M :\cdot A \wedge B}{\mathbf{fst}\, M :\cdot A}\wedge\mathrm{E_L} \qquad \frac{M :\cdot A \wedge B}{\mathbf{snd}\, M :\cdot B}\wedge\mathrm{E_R}$$

The local reductions now lead to two obvious local reductions of the proof terms.

$$\begin{array}{rcl} \mathbf{fst}\,\langle M, N\rangle & \longrightarrow_L & M \\ \mathbf{snd}\,\langle M, N\rangle & \longrightarrow_L & N \end{array}$$

Implication. The proof of an implication $A \supset B$ will be represented by a function which maps proofs of A to proofs of B. The introduction rule explicitly forms such a function by λ-abstraction and the elimination rule applies the function to an argument.

$$\dfrac{\dfrac{\dfrac{}{u :\cdot A}u'}{\vdots}\\ M :\cdot B}{(\lambda u{:}A.\ M) :\cdot A \supset B}\supset\!\mathrm{I}^{u,u'} \qquad \dfrac{M :\cdot A \supset B \qquad N :\cdot A}{M\,N :\cdot B}\supset\!\mathrm{E}$$

The binding of the variable u in the conclusion of $\supset$I correctly models the intuition that the hypothesis is discharged and not available outside deduction of the premiss. The abstraction is labelled with the proposition A so that we can later show that the proof term uniquely determines a natural deduction. If A were not given then, for example, $\lambda u.\ u$ would be ambigous and serve as a proof term for $A \supset A$ for any formula A. The local reduction rule is β-reduction.

$$(\lambda u{:}A.\ M)\,N \quad \longrightarrow_L \quad [N/u]M$$

Here bound variables in M that are free in N must be renamed in order to avoid variable capture.

Disjunction. The proof term for disjunction introduction is the proof of the premiss together with an indication whether it was inferred by introduction on the left or on the right. We also annotate the proof term with the formula which did not occur in the premiss so that a proof terms always proves exactly one proposition.

$$\dfrac{M :\cdot A}{\mathbf{inl}^B\ M :\cdot A \vee B}\vee\mathrm{I_L} \qquad \dfrac{N :\cdot B}{\mathbf{inr}^A\ N :\cdot A \vee B}\vee\mathrm{I_R}$$

The elimination rule corresponds to a case construction.

$$\dfrac{M :\cdot A \vee B \qquad \begin{array}{c}\dfrac{}{u_1 :\cdot A}u_1' \\ \vdots \\ N_1 :\cdot C\end{array} \qquad \begin{array}{c}\dfrac{}{u_2 :\cdot B}u_2' \\ \vdots \\ N_2 :\cdot C\end{array}}{(\mathbf{case}\ M\ \mathbf{of}\ \mathbf{inl}\,u_1 \Rightarrow N_1 \mid \mathbf{inr}\,u_2 \Rightarrow N_2) :\cdot C}\vee\mathrm{E}^{u_1,u_2,u_1',u_2'}$$

Since the variables u_1 and u_2 label assumptions, the corresponding proof term variables are *bound* in N_1 and N_2, respectively. The two reduction rules now also look like rules of computation in a λ-calculus.

$$\begin{array}{lcl}\mathbf{case}\ \mathbf{inl}^B\ M\ \mathbf{of}\ \mathbf{inl}\,u_1 \Rightarrow N_1 \mid \mathbf{inr}\,u_2 \Rightarrow N_2 & \longrightarrow_L & [M/u_1]N_1 \\ \mathbf{case}\ \mathbf{inr}^A\ M\ \mathbf{of}\ \mathbf{inl}\,u_1 \Rightarrow N_1 \mid \mathbf{inr}\,u_2 \Rightarrow N_2 & \longrightarrow_L & [M/u_2]N_2\end{array}$$

The substitution of a deduction for a hypothesis is represented by the substitution of a proof term for a variable.

Negation. This is similar to implication. Since the premise of the rule is parametric in p the corresponding proof constructor must bind a propositional variable p, indicated by μ^p. Similar, the elimination construct must record the formula so we can substitute for p in the reduction. This is indicated as a subscript in $\cdot_C$.

$$\dfrac{\begin{array}{c}\overline{u :\cdot A}\ u' \\ \vdots \\ M :\cdot p\end{array}}{\mu^p u{:}A.\ M :\cdot \neg A}\ \neg\mathrm{I}^{p,u,u'} \qquad\qquad \dfrac{M :\cdot \neg A \qquad N :\cdot A}{M \cdot_C N :\cdot C}\ \neg\mathrm{E}$$

The reduction performs formula and proof term substitutions.

$$(\mu^p u{:}A.\ M) \cdot_C N \quad \longrightarrow_L \quad [N/u][C/p]M$$

Truth. The proof term for $\top$I is written $\langle\,\rangle$.

$$\dfrac{}{\langle\,\rangle :\cdot \top}\ \top\mathrm{I}$$

Of course, there is no reduction rule.

Absurdity. Here we need to annotate the proof term **abort** with the formula being proved to avoid ambiguity.

$$\dfrac{M :\cdot \bot}{\mathbf{abort}^C\ M :\cdot C}\ \bot\mathrm{E}$$

Again, there is no reduction rule.

In summary, we have

Terms	M	$::=$	u	*Hypotheses*
			$\mid \langle M_1, M_2\rangle \mid \mathbf{fst}\, M \mid \mathbf{snd}\, M$	*Conjunction*
			$\mid \lambda u{:}A.\ M \mid M_1\ M_2$	*Implication*
			$\mid \mathbf{inl}^A\ M \mid \mathbf{inr}^A\ M$	*Disjunction*
			$\mid (\mathbf{case}\ M_1\ \mathbf{of}\ \mathbf{inl}\, u_1 \Rightarrow M_2 \mid \mathbf{inr}\, u_1 \Rightarrow M_3)$	
			$\mid \mu^p u{:}A.\ M \mid M \cdot_C N$	*Negation*
			$\mid \langle\,\rangle$	*Truth*
			$\mid \mathbf{abort}^A\ M$	*Falsehood*

and the reduction rules

$$
\begin{array}{lrcl}
\pi_1 & \mathbf{fst}\,\langle M, N\rangle & \longrightarrow_L & M \\
\pi_2 & \mathbf{snd}\,\langle M, N\rangle & \longrightarrow_L & N \\
\beta & (\lambda u{:}A.\ M)\,N & \longrightarrow_L & [N/u]M \\
\rho_1 & \mathbf{case}\,\mathbf{inl}^B\,M\,\mathbf{of}\,\mathbf{inl}\,u_1 \Rightarrow N_1 \mid \mathbf{inr}\,u_2 \Rightarrow N_2 & \longrightarrow_L & [M/u_1]N_1 \\
\rho_2 & \mathbf{case}\,\mathbf{inr}^A\,M\,\mathbf{of}\,\mathbf{inl}\,u_1 \Rightarrow N_1 \mid \mathbf{inr}\,u_2 \Rightarrow N_2 & \longrightarrow_L & [M/u_2]N_2 \\
\mu & (\mu^p u{:}A.\ M) \cdot_C N & \longrightarrow_L & [N/u][C/p]M
\end{array}
$$

We can now see that the formulas act as types for proof terms. Shifting to the usual presentation of the typed λ-calculus we use τ and σ as symbols for types, and $\tau \times \sigma$ for the product type, $\tau \to \sigma$ for the function type, $\tau + \sigma$ for the disjoint sum type, 1 for the unit type and 0 for the empty or void type. Base types b remain unspecified, just as the basic propositions of the propositional calculus remain unspecified. Types and propositions then correspond to each other as indicated below.

$$
\begin{array}{rccccccccccccc}
\text{Types} & \tau & ::= & b & \mid & \tau_1 \times \tau_2 & \mid & \tau_1 \to \tau_2 & \mid & \tau_1 + \tau_2 & \mid & 1 & \mid & 0 \\
\text{Propositions} & A & ::= & p & \mid & A_1 \wedge A_2 & \mid & A_1 \supset A_2 & \mid & A_1 \vee A_2 & \mid & \top & \mid & \bot
\end{array}
$$

We omit here the negation type which is typically not used in functional programming and thus does not have a well-known counterpart. We can $\neg A$ as corresponding to $\tau \to 0$, where τ corresponds to A (see Exercise **??**). In addition to the terms, shown above, the current set of hypotheses which are available in a subdeduction are usually made explicit in a *context* Γ. These are simply a list of variables with their types. We assume that no variable is declared twice in a context.

$$\text{Contexts} \quad \Gamma \quad ::= \quad \cdot \mid \Gamma, u{:}A$$

We omit the $\cdot$ at the beginning of a context or to the left of the typing judgment. The typing rules for the λ-calculus are the rules for natural deduction under a shift of notation and with explicit contexts. The typing judgment has the form

$$\Gamma \triangleright M : \tau \quad \textit{M has type } \tau \textit{ in context } \Gamma$$

$$\frac{\Gamma \triangleright M : \tau \qquad \Gamma \triangleright N : \sigma}{\Gamma \triangleright \langle M, N\rangle : \tau \times \sigma}\,\mathsf{pair}$$

$$\frac{\Gamma \triangleright M : \tau \times \sigma}{\Gamma \triangleright \mathbf{fst}\, M : \tau}\,\mathsf{fst} \qquad \frac{\Gamma \triangleright M : \tau \times \sigma}{\Gamma \triangleright \mathbf{snd}\, M : \sigma}\,\mathsf{snd}$$

$$\frac{\Gamma, u{:}\tau \triangleright M : \sigma}{\Gamma \triangleright (\lambda u{:}\tau.\ M) : \tau \to \sigma}\,\mathsf{lam} \qquad \frac{u : \tau \text{ in } \Gamma}{\Gamma \triangleright u : \tau}\,\mathsf{var}$$

$$\frac{\Gamma \triangleright M : \tau \to \sigma \qquad \Gamma \triangleright N : \tau}{\Gamma \triangleright M\, N : \sigma}\,\mathsf{app}$$

$$\frac{\Gamma \triangleright M : \tau}{\Gamma \triangleright \mathbf{inl}^{\sigma}\, M : \tau + \sigma}\,\mathsf{inl} \qquad \frac{\Gamma \triangleright N : \sigma}{\Gamma \triangleright \mathbf{inr}^{\tau}\, N : \tau + \sigma}\,\mathsf{inr}$$

$$\frac{\Gamma \triangleright M : \tau + \sigma \qquad \Gamma, u{:}\tau \triangleright N_1 : \nu \qquad \Gamma, u{:}\sigma \triangleright N_2 : \nu}{\Gamma \triangleright (\mathbf{case}\, M\, \mathbf{of}\, \mathbf{inl}\, u_1 \Rightarrow N_1 \mid \mathbf{inr}\, u_2 \Rightarrow N_2) : \nu}\,\mathsf{case}$$

$$\frac{}{\Gamma \triangleright \langle\,\rangle : 1}\,\mathsf{unit} \qquad \frac{\Gamma \triangleright M : 0}{\Gamma \triangleright \mathbf{abort}^{\nu}\, M : \nu}\,\mathsf{abort}$$

7.5 Generalization to First-Order Arithmetic

We have not yet made the connection between local reduction and computation in a functional programming language. Before we examine this relationship, we investigate how the Curry-Howard isomorphism might be generalized to first-order arithmetic. This involves two principal steps: one to account for quantifiers, and one to account for natural numbers and proofs by induction. The natural numbers here stand in for arbitrary *inductively defined datatypes*, which are beyond the scope of these notes. We begin by generalizing to first-order logic, that is, without any particular built-in datatype such as the natural numbers.

Terms and Atomic Formulas. A well-formed first-order term $f(t_1, \ldots, t_n)$ where f is an n-ary function symbol corresponds to the application $\mathrm{f}\, t_1 \ldots t_n$, where f has been declared to be a constant of type $\mathrm{i} \to \cdots \to \mathrm{i}$. In the propositional case, atomic formulas are drawn from some basic propositions and propositional variables. In first-order logic, well-formed atomic formulas have the form $P(t_1, \ldots, t_n)$, where P is an n-ary predicate. This corresponds directly to the familiar notion of a *type family* p indexed by terms $t_1, \ldots, t_n$, all of type i. In summary: under the Curry-Howard isomorphism, predicates correspond to type families.

Universal Quantification. Recall the introduction and elimination rules for universal quantification:

$$\frac{[a/x]A}{\forall x.\ A}\,\forall\mathrm{I}^a \qquad\qquad \frac{\forall x.\ A}{[t/x]A}\,\forall\mathrm{E}$$

where a is a new parameter. This suggests that the proof term should be a function which, when given a well-formed first-order term t returns a proof of $[t/x]A$. It is exactly this reading of the introduction rule for $\forall$ that motivated its representation in LF. The elimination rule then applies this function to the argument t. This reasoning yields the following formulation of the rules with explicit proof terms.

$$\frac{[a/x]M :\cdot [a/x]A}{(\lambda x{:}i.\ M) :\cdot \forall x.\ A}\,\forall\mathrm{I} \qquad\qquad \frac{M :\cdot \forall x.\ A}{M\,t :\cdot [t/x]A}\,\forall\mathrm{E}$$

This formulation glosses over that we ought to check that t is actually well-formed, that is, has type i. Similarly, we may need the assumption that a is of type i while deriving the premiss of the introduction rule. This leads to the following versions.

$$\frac{\begin{array}{c}\overline{a : i}^{\,u}\\ \vdots\\ [a/x]M :\cdot [a/x]A\end{array}}{(\lambda x{:}i.\ M) :\cdot \forall x.\ A}\,\forall\mathrm{I}^{a,u} \qquad\qquad \frac{M :\cdot \forall x.\ A \qquad t : i}{M\,t :\cdot [t/x]A}\,\forall\mathrm{E}$$

It has now become apparent, that we have *not* unified the notions of propositions and types: these rules employ a typing judgment $t : i$ as well as a proof term judgment $M :\cdot A$. The restriction of the universal quantifier to terms of type i is quite artificial from the point of view of the Curry-Howard isomorphism. If we drop this distinction and further eliminate the distinction between variables and parameters we obtain the rules for abstraction and application in the LF type theory, written in the style of natural deduction rather than with an explicit context.

$$\frac{\begin{array}{c}\overline{x : B}^{\,u}\\ \vdots\\ M : A\end{array}}{(\lambda x{:}B.\ M) : \forall x{:}B.\ A}\,\forall\mathrm{I}^{x,u} \qquad\qquad \frac{M : \forall x{:}B.\ A \qquad N : B}{M\,N : [N/x]A}\,\forall\mathrm{E}$$

Thus the universal quantifier corresponds to a dependent function type constructor Π under a modified Curry-Howard isomorphism. Following this line of development to its logical conclusion leads to a type theory in the tradition of Martin-Löf. Thompson [Tho91] provides a good introduction to this complex subject. We will only return to it briefly when discussing the existential quantifier below; instead we pursue an alternative whereby we *contract* proofs to programs rather than insisting on an isomorphism.

The local reduction rule for the universal quantifier

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ [a/x]A\end{matrix}}{\forall x.\ A}\,\forall\mathrm{I}}{[t/x]A}\,\forall\mathrm{E} \quad \Longrightarrow_L \quad \begin{matrix}[t/a]\mathcal{D}\\ [t/x]A\end{matrix}$$

corresponds to β-reduction on the proof terms.

$$(\lambda x{:}i.\ M)\,t \quad \longrightarrow_L \quad [t/x]M$$

The language of proof terms now has two forms of abstraction: abstraction over individuals (which yields proof terms for universal formulas) and abstraction over proof terms (which yields proof terms for implications). In a type theory such as LF, these are identified and the latter is seen as a special case of the former where there is no dependency (see Exercise 3.9).

Existential Quantification. The proof term for existential introduction must record both the *witness* term t and the proof term for $[t/x]A$. That is, we have

$$\dfrac{[t/x]A}{\exists x.\ A}\,\exists\mathrm{I} \qquad\qquad \dfrac{M :\cdot [t/x]A}{\langle t, M\rangle :\cdot \exists x.\ A}\,\exists\mathrm{I}$$

In the type-theoretic version of this rule, we generalize the existential quantifier to range over arbitrary types, and we must check that the witness has the requested type.

$$\dfrac{N : B \qquad M : [N/x]A}{\langle N, M\rangle : \exists x{:}B.\ A}\,\exists\mathrm{I}$$

In analogy to the dependent function type, the existential forms a type for dependent pairs: the *type* of the second component M depends on the first component N. This is refered to as a *sum type* or Σ-type, since it is most commonly written as $\Sigma x{:}A.\ B$. There are some variations on the sum type, depending on how one can access the components of its elements, that is, depending on the form of the elimination rule. If we directly equip the logical elimination rule

$$\dfrac{\exists x.\ A \qquad \begin{matrix}\overline{[a/x]A}^{\,u}\\ \vdots\\ C\end{matrix}}{C}\,\exists\mathrm{E}^{a,u}$$

with proof objects we obtain a **case** construct for pairs (with only one branch).

$$\dfrac{M :\cdot \exists x.\ A \qquad \begin{array}{c} \overline{u :\cdot [a/x]A}^{\,u'} \\ \vdots \\ [a/x]N :\cdot C \end{array}}{(\mathbf{case}\ M\ \mathbf{of}\ \langle x, u\rangle \Rightarrow N) :\cdot C}\ \exists\mathrm{E}^{a,u,u'}$$

Here, x and u are bound in N which reflects that the right premiss of the existential elimination rule is parametric in a and u. In type theory, this construct is sometimes called **split** or **spread**. The local proof reduction

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ [t/x]A\end{array}}{\exists x.\ A}\ \exists\mathrm{I} \qquad \begin{array}{c} \overline{[a/x]A}^{\,u} \\ \mathcal{E} \\ C \end{array}}{C}\ \exists\mathrm{E}^{a,u} \quad \Longrightarrow_L \quad \begin{array}{c} \overline{\quad\mathcal{D}\quad}^{\,u} \\ [t/x]A \\ [t/a]\mathcal{E} \\ C \end{array}$$

translates to the expected reduction rule on proof terms

$$\mathbf{case}\ \langle t, M\rangle\ \mathbf{of}\ \langle x, u\rangle \Rightarrow N \quad \longrightarrow_L \quad [t/x][M/u]N.$$

The language of proof terms now has two different forms of pairs: those that pair a first-order term with a proof term (which form proof terms of existential formulas) and those that pair two proof terms (which form proof terms of conjunctions). In type theory these are identified, and the latter is seen as a special case of the former where there is no dependency.

Certain subtle problems arise in connection with the existential type. Because of the difference in the elimination rules for conjunction and existential, we cannot apply **fst** and **snd** to pairs of the form $\langle t, M\rangle$. Moreover, **snd** cannot be defined in general from **case** for dependently typed pairs (see Exercise 7.8). Perhaps even more significantly, proof terms no longer uniquely determine the formula they prove (see Exercise 7.9). In the theory of programming languages, existential and sum types can be used to model abstract data types and modules; the problems mentioned above must be addressed carefully in each such application.

Natural Numbers. If we fix the domain of individuals to be the natural numbers, we arrive at first-order arithmetic. We write nat instead of i and we have a constant z (denoting 0) and a unary function symbol s (denoting the successor function). The fact that these are the only ways to construct natural numbers is usual captured by Peano's axiom schema of induction:

For any formula A with free variable x,

$$[z/x]A \wedge (\forall x.\ A \supset [s(x)/x]A) \supset \forall x.\ A.$$

Our approach has been to explain the semantics of language constructs by inference rules, rather than axioms in a logic. If we look at the rules

$$\frac{}{z : \mathsf{nat}}\,\mathrm{NI}_z \qquad\qquad \frac{t : \mathsf{nat}}{s(t) : \mathsf{nat}}\,\mathrm{NI}_s$$

as introduction rules for elements of the type nat, then we arrive at the following elimination rule.

$$\frac{t : \mathsf{nat} \qquad [z/x]A \qquad \begin{array}{c}\overline{[a/x]A}^{\,u}\\ \vdots \\ [s(a)/x]A\end{array}}{[t/x]A}\,\mathrm{NE}^{a,u}$$

Here, the judgment of the third premiss is parametric in a and hypothetical in u. The local reductions follow from this view. If t was inferred by NI_z (and thus $t = z$), the conclusion $[z/x]A$ is proved directly by the second premiss.

$$\frac{\dfrac{}{z : \mathsf{nat}}\,\mathrm{NI}_z \qquad \begin{array}{c}\mathcal{E}_1\\ [z/x]A\end{array} \qquad \begin{array}{c}\overline{[a/x]A}^{\,u}\\ \mathcal{E}_2 \\ [s(a)/x]A\end{array}}{[z/x]A}\,\mathrm{NE}^{a,u} \quad \Longrightarrow_L \quad \begin{array}{c}\mathcal{E}_1\\ [z/x]A\end{array}$$

If t was inferred by NI_s (and thus $t = s(t')$), then we can obtain a derivation $\mathcal{E}'$ of $[t'/x]A$ by NE. To obtain a derivation of $[s(t')/x]A$ we use the parametric and hypothetical derivation of the rightmost premiss: we substitute t' for a and $\mathcal{D}'$ for the hypothesis u.

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}'\\ t' : \mathsf{nat}\end{array}}{s(t') : \mathsf{nat}}\,\mathrm{NI}_s \qquad \begin{array}{c}\mathcal{E}_1\\ [z/x]A\end{array} \qquad \begin{array}{c}\overline{[a/x]A}^{\,u}\\ \mathcal{E}_2 \\ [s(a)/x]A\end{array}}{[s(t')/x]A}\,\mathrm{NE}^{a,u}$$

$$\Longrightarrow_L \quad \begin{array}{c}\dfrac{\begin{array}{c}\mathcal{D}'\\ t' : \mathsf{nat}\end{array} \qquad \begin{array}{c}\mathcal{E}_1\\ [z/x]A\end{array} \qquad \begin{array}{c}\overline{[a/x]A}^{\,u}\\ \mathcal{E}_2 \\ [s(a)/x]A\end{array}}{[t'/x]A}\,\mathrm{NE}^{a,u}\\ [t'/a]\mathcal{E}_2 \\ [s(t')/x]A\end{array}$$

In order to write out proof terms for the NE rule, we need a new proof term constructor **prim**.

$$\cfrac{t : \mathsf{nat} \qquad N_1 :\cdot [z/x]A \qquad \begin{array}{c}\overline{u :\cdot [a/x]A}\,u' \\ \vdots \\ [a/x]N_2 :\cdot [s(a)/x]A\end{array}}{\mathbf{prim}\; t\; N_1\; (x, u.\; N_2) :\cdot [t/x]A}\,\mathrm{NE}^{a,u,u'}$$

Here, x and u are bound in N_2. The local reductions can then be written as

$$\begin{array}{rcl}\mathbf{prim}\; z\; N_1\; (x, u.\; N_2) & \longrightarrow_L & N_1 \\ \mathbf{prim}\; (s(t'))\; N_1\; (x, u.\; N_2) & \longrightarrow_L & [t/x][(\mathbf{prim}\; t'\; N_1\; (x, u.\; N_2))/u]N_2\end{array}$$

An analogous construct of *primitive recursion* at the level of first-order terms (rather than proof terms) is also usually assumed in the treatment of arithmetic in order to *define* new functions such as addition, multiplication, exponentiation, *etc.* From the point of view of type theory, this is a special case of the above, where A does not depend on x. The typing rule has the form

$$\cfrac{t : \mathsf{nat} \qquad t_1 : \tau \qquad \begin{array}{c}\overline{x : \tau}\,u_1, \quad \overline{y : \tau}\,u_2 \\ \vdots \\ t_2 : \tau\end{array}}{\mathbf{prim}\; t\; t_1\; (x, y.\; t_2) : \tau}\,\mathrm{NE}^{x,y,u_1,u_2}$$

and the local reduction are

$$\begin{array}{rcl}\mathbf{prim}\; z\; t_1\; (x, y.\; t_2) & \longrightarrow_L & t_1 \\ \mathbf{prim}\; (s(t'))\; t_1\; (x, y.\; t_2) & \longrightarrow_L & [t'/x][(\mathbf{prim}\; t'\; t_1\; (x, y.\; t_2))/y]t_2\end{array}$$

Here we followed the convention for type systems where parameters are not used explicitly, but the same name is used for a bound variable and its corresponding parameter.

In ordinary primitive recursion we also have λ-abstraction and application, but the result type τ of the **prim** constructor is restricted to be nat. The more general definition above is the basis for the language of *primitive recursive functionals*, which is at the core of Gödel's system T [Göd90]. Strictly more functions can be defined by using primitive recursion at higher types. A famous example is the Ackermann function, which is not primitive recursive, but lies within Gödel's system T. The surface syntax of terms defined by **prim** can be rather opaque, their properties are illustrated by the following formulation. Define

$$\begin{array}{rcl}f & = & \lambda x.\; \mathbf{prim}\; x\; t_1\; (x, y.\; t_2) \\ g & = & \lambda x.\; \lambda y.\; t_2.\end{array}$$

Then f satisfies (using a notion of equivalence $\equiv$ not made precise here)

$$\begin{array}{rcl} f(z) & \equiv & t_1 \\ f(s(x)) & \equiv & g\,x\,(f(x)). \end{array}$$

If we think in terms of evaluation, then x will be bound to the predecessor of the argument to f, and y will be bound to the result of the recursive call to f on x when evaluating t_2.

In order to make arithmetic useful as a specification language, we also need some primitive predicates such as equality or inequality between natural numbers. An equality should satisfy the usual axioms (reflexivity, symmetry, transitivity, congruence), but it should also allow computation with functions defined by primitive recursion. Furthermore, we need to consider Peano's third and fourth axioms.[2]

$$\begin{array}{c} \forall x.\, \neg s(x) \doteq z \\ \forall x.\, \forall y.\, s(x) \doteq s(y) \supset x \doteq y \end{array}$$

These axioms insure that we can prove that different natural numbers are in fact distinct; all the other axioms would also hold if, for example, we interpreted the type nat as the natural numbers modulo 5. We can formulate these additional axioms as inference rules and write out appropriate proof terms. We postpone the detailed treatment until the next section, since some of these rules will have no computational significance.

In the realm of Martin-Löf type theories, the best treatment of equality is also a difficult and controversial subject. Built into the theory is an *equality judgment* that relates objects based on the rules of computation. We have an *equality type* which may or may not be extensional. For further discussion, the interested reader is referred to [Tho91].

[*insert equality rules here, but which formulation?*]

7.6 Contracting Proofs to Programs*

In this section we presume that we are only interested in data values as results of computations, rather than proof terms. Then a proof may contain significantly more information than the program extracted from it. This phenomenon should not come unexpectedly: it usually requires much more effort to prove a program correct than to write it. As a first approximation toward program extraction, we postulate that the type extracted from an equality formula should be the unit type 1, and that the proof object extracted from *any* proof of an equality should be the

[2] [*a note on negation*]

unit element $\langle\,\rangle$. The extraction function $|\cdot|$ for types then has the following shape.

$$
\begin{array}{rcl}
|t_1 \doteq t_2| & = & 1 \\
|A \wedge B| & = & |A| \times |B| \\
|A \vee B| & = & |A| + |B| \\
|A \supset B| & = & |A| \rightarrow |B| \\
|\top| & = & 1 \\
|\bot| & = & 0 \\
|\forall x.\, A| & = & \mathsf{nat} \rightarrow |A| \\
|\exists x.\, A| & = & \mathsf{nat} \times |A|
\end{array}
$$

As examples we consider two simple specifications: one for the predecessor function on natural numbers, and one for the integer division of a number by 2. For the first example, recall that $\neg A$ is merely an abbreviation for $A \supset \bot$.[3]

$$
\begin{array}{rcl}
\mathit{Pred} & = & \forall x.\, \exists y.\, \neg x \doteq z \supset x \doteq s(y) \\
|\mathit{Pred}| & = & \mathsf{nat} \rightarrow (\mathsf{nat} \times ((1 \rightarrow 0) \rightarrow 1)) \\
\\
\mathit{double} & = & \mathbf{lam}\, x.\, \mathbf{prim}\, x\, \mathbf{z}\, (x', y.\, \mathbf{s}\,(\mathbf{s}\, y)) \\
\mathit{Half} & = & \forall x.\, \exists y.\, x \doteq \mathit{double}\, y \vee x \doteq s\,(\mathit{double}\, y) \\
|\mathit{Half}| & = & \mathsf{nat} \rightarrow (\mathsf{nat} \times (1 + 1))
\end{array}
$$

These types may be surprising. For example we would expect the predecessor function to yield just a natural number. At this point we observe that there is only one value of the unit type 1. Thus, for example, the function $((1 \rightarrow 0) \rightarrow 1)$ can only ever return the unit element $\langle\,\rangle$. Therefore it carries no new information can can be contracted to 1. We reason further that a value of type $\mathsf{nat} \times 1$ must be a pair of a natural number and the unit element and carries no more information that nat itself. In general we are taking advantage of the following intuitive isomorphisms between types.[4]

$$
\begin{array}{cc}
\tau \times 1 \cong \tau & 1 \times \tau \cong \tau \\
\tau \rightarrow 1 \cong 1 & 1 \rightarrow \tau \cong \tau
\end{array}
$$

Some other isomorphisms do *not* hold. In particular, the type $1 + 1$ cannot be simplified: It contains two values ($\mathbf{inl}\,\langle\,\rangle$ and $\mathbf{inr}\,\langle\,\rangle$), while 1 contains only one. In order to make programs more legible, we will abbreviate

$$
\begin{array}{rcl}
1 + 1 & = & \mathsf{bool} \\
\mathbf{inl}\,\langle\,\rangle & = & \mathsf{true} \\
\mathbf{inr}\,\langle\,\rangle & = & \mathsf{false} \\
\mathbf{if}\; e_1\; \mathbf{then}\; e_2\; \mathbf{else}\; e_3 & = & \mathbf{case}\; e_1\; \mathbf{of}\; \mathbf{inl}\, x \Rightarrow e_2 \mid \mathbf{inr}\, y \Rightarrow e_3
\end{array}
$$

---

[3] [*this must be fixed for proper treatment of negation*]

[4] Some further isomorphism may be observed regarding the void type 0, but we do not consider it here. For a detailed treatment, see [And93].

In the modified type extraction, we use the isomorphisms above in order to concentrate on computational contents.

[*The remainder of this section and chapter is under construction.*]

7.7 Proof Reduction and Computation*

7.8 Termination*

7.9 Exercises

Exercise 7.1 Prove the following by natural deduction using only intuitionistic rules when possible. We use the convention that $\supset$, $\wedge$, and $\vee$ associate to the right, that is, $A \supset B \supset C$ stands for $A \supset (B \supset C)$. $A \equiv B$ is a syntactic abbreviation for $(A \supset B) \wedge (B \supset A)$. Also, we assume that $\wedge$ and $\vee$ bind more tightly than $\supset$, that is, $A \wedge B \supset C$ stands for $(A \wedge B) \supset C$. The scope of a quantifier extends as far to the right as consistent with the present parentheses. For example, $(\forall x.\, P(x) \supset C) \wedge \neg C$ would be disambiguated to $(\forall x.\, (P(x) \supset C)) \wedge (\neg C)$.

1. $A \supset B \supset A$.
2. $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$.
3. (Peirce's Law). $((A \supset B) \supset A) \supset A$.
4. $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$.
5. $A \supset (A \wedge B) \vee (A \wedge \neg B)$.
6. $(A \supset \exists x.\, P(x)) \equiv \exists x.\, (A \supset P(x))$.
7. $((\forall x.\, P(x)) \supset C) \equiv \exists x.\, (P(x) \supset C)$.
8. $\exists x.\, \forall y.\, (P(x) \supset P(y))$.

Exercise 7.2 Show that the three ways of extending the intuitionistic proof system are equivalent, that is, the same formulas are deducible in all three systems.

Exercise 7.3 Assume we had omitted disjunction and existential quantification and their introduction and elimination rules from the list of logical primitives. In the classical system, give a definition of disjunction and existential quantification (in terms of other logical constants) and show that the introduction and elimination rules now become *admissible rules of inference*. A rule of inference is *admissible* if any deduction using the rule can be transformed into one without using the rule.

Exercise 7.4 Carefully state and prove adequacy of the given representation for for terms, formulas, and natural deductions in LF.

Exercise 7.5 Give an interpretation of the classical calculus in the intuitionistic calculus, that is, define a function $\overline{()}$ from formulas to formulas such that A is deducible in the classical calculus if and only if $\overline{A}$ is deducible in the intuitionistic calculus.

Exercise 7.6 Give the representation of the natural deductions in Exercise 7.1 in Elf.

Exercise 7.7 Give the LF representations of the rules of indirect proof, double negation, and excluded middle from Page 214.

Exercise 7.8 [*on problems with typing of* **snd** *on dependently typed pairs*]

Exercise 7.9 [*failure of uniqueness of types with existential or* Σ *types*]

Exercise 7.10 Using the normalization theorem for intuitionistic natural deduction (Theorem **??**) prove that the general law of excluded middle is not derivable in intuitionistic logic.