We write $m =_N n$. Otherwise we follow the blueprint of the less-than relation.

$$\frac{\Gamma \vdash m \in \mathbf{nat} \qquad \Gamma \vdash n \in \mathbf{nat}}{\Gamma \vdash m =_N n \ \mathit{prop}} =_N F$$

$$\frac{}{\Gamma \vdash \mathbf{0} =_N \mathbf{0} \ \mathit{true}} =_N I_0 \qquad \frac{\Gamma \vdash m =_N n \ \mathit{true}}{\Gamma \vdash \mathbf{s}(m) =_N \mathbf{s}(n) \ \mathit{true}} =_N I_s$$

$$no =_N E_{00} \ \mathit{elimination\ rule} \qquad \frac{\Gamma \vdash \mathbf{0} =_N \mathbf{s}(n) \ \mathit{true}}{\Gamma \vdash C \ \mathit{true}} =_N E_{0s}$$

$$\frac{\Gamma \vdash \mathbf{s}(m) =_N \mathbf{0} \ \mathit{true}}{\Gamma \vdash C \ \mathit{true}} =_N E_{s0} \qquad \frac{\Gamma \vdash \mathbf{s}(m) =_N \mathbf{s}(n) \ \mathit{true}}{\Gamma \vdash m =_N n \ \mathit{true}} =_N E_{ss}$$

Note the difference between the *function*

$$\mathit{eq} \in \mathbf{nat} \to \mathbf{nat} \to \mathbf{bool}$$

and the *proposition*

$$m =_N n$$

The equality function provides a computation on natural numbers, always returning **true** or **false**. The proposition $m =_N n$ requires *proof*. Using induction, we can later verify a relationship between these two notions, namely that $\mathit{eq}\, n\, m$ reduces to **true** if $m =_N n$ is true, and $\mathit{eq}\, n\, m$ reduces to **false** if $\neg(m =_N n)$.

3.10 Induction

Now that we have introduced the basic propositions regarding order and equality, we can consider induction as a reasoning principle. So far, we have considered the following elimination rule for natural numbers:

$$\frac{\Gamma \vdash t \in \mathbf{nat} \qquad \Gamma \vdash t_0 \in \tau \qquad \Gamma, x \in \mathbf{nat}, f(x) \in \tau \vdash t_s \in \tau}{\Gamma \vdash \mathbf{rec}\ t\ \mathbf{of}\ f(\mathbf{0}) \Rightarrow t_0 \mid f(\mathbf{s}(x)) \Rightarrow t_s \in \tau}\ \mathbf{nat}E$$

This rule can be applied if we can derive $t \in \mathbf{nat}$ from our assumptions and we are trying to construct a term $s \in \tau$. But how do we use a variable or term $t \in \mathbf{nat}$ if the judgment we are trying to prove has the form $M : A$, that is, if we are trying the prove the truth of a proposition? The answer is induction. This is actually very similar to primitive recursion. The only complication is that the proposition A we are trying to prove may depend on t. We indicate this by writing $A(x)$ to mean the proposition A with one or more occurrences of a variable x. $A(t)$ is our notation for the result of substituting t for x in A. We

could also write $[t/x]A$, but this is more difficult to read. Informally, induction says that in order to prove $A(t)$ *true* for arbitrary t we have to prove $A(\mathbf{0})$ *true* (the base case), and that for every $x \in \mathbf{nat}$, if $A(x)$ *true* then $A(\mathbf{s}(x))$ *true*.

Formally this becomes:

$$\frac{\Gamma \vdash t \in \mathbf{nat} \qquad \Gamma \vdash A(\mathbf{0})\ \mathit{true} \qquad \Gamma, x \in \mathbf{nat}, A(x)\ \mathit{true} \vdash A(\mathbf{s}(x))\ \mathit{true}}{\Gamma \vdash A(t)\ \mathit{true}}\ \mathbf{nat}E'$$

Here, $A(x)$ is called the *induction predicate*. If t is a variable (which is frequently the case) it is called the *induction variable*. With this rule, we can now prove some more interesting properties. As a simple example we show that $m < \mathbf{s}(m)$ *true* for any natural number m. Here we use $\mathcal{D}$ to stand for the derivation of the third premise in order to overcome the typesetting difficulties.

$$\mathcal{D} = \frac{\dfrac{}{m \in \mathbf{nat}, x \in \mathbf{nat}, x < \mathbf{s}(x)\ \mathit{true} \vdash x < \mathbf{s}(x)\ \mathit{true}}}{m \in \mathbf{nat}, x \in \mathbf{nat}, x < \mathbf{s}(x)\ \mathit{true} \vdash \mathbf{s}(x) < \mathbf{s}(\mathbf{s}(x))}\ {<}I_s$$

$$\frac{\dfrac{}{m \in \mathbf{nat} \vdash m \in \mathbf{nat}} \qquad \dfrac{}{m \in \mathbf{nat} \vdash \mathbf{0} < \mathbf{s}(\mathbf{0})}\ {<}I_0 \qquad \mathcal{D}}{m \in \mathbf{nat} \vdash m < \mathbf{s}(m)}\ \mathbf{nat}E'$$

The property $A(x)$ appearing in the induction principle is $A(x) = x < \mathbf{s}(x)$. So the final conclusion is $A(m) = m < \mathbf{s}(m)$. In the second premise we have to prove $A(\mathbf{0}) = \mathbf{0} < \mathbf{s}(\mathbf{0})$ which follows directly by an introduction rule.

Despite the presence of the induction rule, there are other properties we cannot yet prove easily since the logic does not have quantifiers. An example is the decidability of equality: For any natural numbers m and n, either $m =_{\scriptscriptstyle N} n$ or $\neg(m =_{\scriptscriptstyle N} n)$. This is an example of the practical limitations of *quantifier-free induction*, that is, induction where the induction predicate does not contain any quantifiers.

The topic of this chapter is the interpretation of constructive proofs as programs. So what is the computational meaning of induction? It actually corresponds very closely to primitive recursion.

$$\frac{\Gamma \vdash t \in \mathbf{nat} \qquad \Gamma \vdash M : A(\mathbf{0}) \qquad \Gamma, x \in \mathbf{nat}, u(x){:}A(x) \vdash N : A(\mathbf{s}(x))}{\Gamma \vdash \mathbf{ind}\ t\ \mathbf{of}\ u(\mathbf{0}) \Rightarrow M \mid u(\mathbf{s}(x)) \Rightarrow N : A(t)}\ \mathbf{nat}E'$$

Here, $u(x)$ is just the notation for a variable which may occur in N. Note that u cannot occur in M or in N in any other form. The reduction rules are precisely the same as for primitive recursion.

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

We see that primitive recursion and induction are almost identical. The only difference is that primitive recursion returns an element of a type, while induction generates a proof of a proposition. Thus one could say that they are related by an extension of the Curry-Howard correspondence. However, not every type τ can be naturally interpreted as a proposition (which proposition, for example, is expressed by **nat**?), so we no longer speak of an isomorphism.

We close this section by the version of the rules for the basic relations between natural numbers that carry proof terms. This annotation of the rules is straightforward.

$$\frac{\Gamma \vdash n \in \mathbf{nat} \quad \Gamma \vdash m \in \mathbf{nat}}{\Gamma \vdash n < m \ prop} <F$$

$$\frac{}{\Gamma \vdash \mathbf{lt}_0 : \mathbf{0} < \mathbf{s}(n)} <I_0 \qquad \frac{\Gamma \vdash M : m < n}{\Gamma \vdash \mathbf{lt}_s(M) : \mathbf{s}(m) < \mathbf{s}(n)} <I_s$$

$$\frac{\Gamma \vdash M : m < \mathbf{0}}{\Gamma \vdash \mathbf{ltE}_0(M) : C} <E_0$$

$$\textit{no rule for } \mathbf{0} < \mathbf{s}(n') \qquad \frac{\Gamma \vdash M : \mathbf{s}(m') < \mathbf{s}(n')}{\Gamma \vdash \mathbf{ltE}_s(M) : m' < n'} <E_s$$

$$\frac{\Gamma \vdash m \in \mathbf{nat} \qquad \Gamma \vdash n \in \mathbf{nat}}{\Gamma \vdash m =_{\mathbf{N}} n \ prop} =_{\mathbf{N}} F$$

$$\frac{}{\Gamma \vdash \mathbf{eq}_0 : \mathbf{0} =_{\mathbf{N}} \mathbf{0}} =_{\mathbf{N}} I_0 \qquad \frac{\Gamma \vdash M : m =_{\mathbf{N}} n}{\Gamma \vdash \mathbf{eq}_s(M) : \mathbf{s}(m) =_{\mathbf{N}} \mathbf{s}(n)} =_{\mathbf{N}} I_s$$

$$\textit{no } =_{\mathbf{N}} E_{00} \textit{ elimination rule} \qquad \frac{\Gamma \vdash M : \mathbf{0} =_{\mathbf{N}} \mathbf{s}(n)}{\Gamma \vdash \mathbf{eqE}_{0s}(M) : C} =_{\mathbf{N}} E_{0s}$$

$$\frac{\Gamma \vdash M : \mathbf{s}(m) =_{\mathbf{N}} \mathbf{0}}{\Gamma \vdash \mathbf{eqE}_{s0}(M) : C} =_{\mathbf{N}} E_{s0} \qquad \frac{\Gamma \vdash M : \mathbf{s}(m) =_{\mathbf{N}} \mathbf{s}(n)}{\Gamma \vdash \mathbf{eqE}_{ss}(M) : m =_{\mathbf{N}} n} =_{\mathbf{N}} E_{ss}$$

Chapter 4

First-Order Logic and Type Theory

In the first chapter we developed the logic of pure propositions without reference to data types such as natural numbers. In the second chapter we explained the computational interpretation of proofs, and, separately, introduced several data types and ways to compute with them using primitive recursion.

In this chapter we will put these together, which allows us to reason about data and programs manipulating data. In other words, we will be able to prove our programs correct with respect to their expected behavior on data. The principal means for this is induction, introduced at the end of the last chapter. There are several ways to employ the machinery we will develop. For example, we can execute proofs directly, using their interpretation as programs. Or we can *extract* functions, ignoring some proof objects that have are irrelevant with respect to the data our programs return. That is, we can contract proofs to programs. Or we can simply write our programs and use the logical machinery we have developed to prove them correct.

In practice, there are situations in which each of them is appropriate. However, we note that in practice we rarely formally prove our programs to be correct. This is because there is no mechanical procedure to establish if a given programs satisfies its specification. Moreover, we often have to deal with input or output, with mutable state or concurrency, or with complex systems where the specification itself could be as difficult to develop as the implementation. Instead, we typically convince ourselves that central parts of our program and the critical algorithms are correct. Even if proofs are never formalized, this chapter will help you in reasoning about programs and their correctness.

There is another way in which the material of this chapter is directly relevant to computing practice. In the absence of practical methods for verifying full correctness, we can be less ambitious by limiting ourselves to program properties that can indeed be mechanically verified. The most pervasive application of this idea in programming is the idea of *type systems*. By checking the type

correctness of a program we fall far short of verifying it, but we establish a kind of consistency statement. Since languages satisfy (or are supposed to satisfy) type preservation, we know that, if a result is returned, it is a value of the right type. Moreover, during the execution of a program (modelled here by reduction), all intermediate states are well-typed which prevents certain absurd situations, such as adding a natural number to a function. This is often summarized in the slogan that "*well-typed programs cannot go wrong*". Well-typed programs are *safe* in this respect. In terms of machine language, assuming a correct compiler, this guards against irrecoverable faults such as jumping to an address that does not contain valid code, or attempting to write to inaccessible memory location.

There is some room for exploring the continuum between types, as present in current programming languages, and full specifications, the domain of *type theory*. By presenting these elements in a unified framework, we have the basis for such an exploration.

We begin this chapter with a discussion of the universal and existential quantifiers, followed by a number of examples of inductive reasoning with data types.

4.1 Quantification

In this section, we introduce universal and existential quantification. As usual, we follow the method of using introduction and elimination rules to explain the meaning of the connectives. First, universal quantification, written as $\forall x{\in}\tau.\ A(x)$. For this to be well-formed, the body must be well-formed under the assumption that x is a variable of type τ.

$$\frac{\tau\ \mathit{type} \qquad \Gamma, x{\in}\tau \vdash A(x)\ \mathit{prop}}{\Gamma \vdash \forall x{\in}\tau.\ A(x)\ \mathit{prop}}\,\forall F$$

For the introduction rule we require that $A(x)$ be valid for arbitrary x. In other words, the premise contains a parametric judgment.

$$\frac{\Gamma, x{\in}\tau \vdash A(x)\ \mathit{true}}{\Gamma \vdash \forall x{\in}\tau.\ A(x)\ \mathit{true}}\,\forall I$$

If we think of this as the defining property of universal quantification, then a verification of $\forall x{\in}\tau.\ A(x)$ describes a construction by which an arbitrary $t \in \tau$ can be transformed into a proof of $A(t)\ \mathit{true}$.

$$\frac{\Gamma \vdash \forall x{\in}\tau.\ A(x)\ \mathit{true} \qquad \Gamma \vdash t \in \tau}{\Gamma \vdash A(t)\ \mathit{true}}\,\forall E$$

We must verify that $t \in \tau$ so that $A(t)$ is a proposition. We can see that the computational meaning of a proof of $\forall x{\in}\tau.\ A(x)\ \mathit{true}$ is a function which, when

given an argument t of type τ, returns a proof of $A(t)$. If we don't mind overloading application, the proof term assignment for the universal introduction and elimination rule is

$$\frac{\Gamma, x{\in}\tau \vdash M : A(x)}{\Gamma \vdash \lambda x{\in}\tau.\ M : \forall x{\in}\tau.\ A(x)}\,\forall I$$

$$\frac{\Gamma \vdash M : \forall x{\in}\tau.\ A(x) \qquad \Gamma \vdash t \in \tau}{\Gamma \vdash M\,t : A(t)}\,\forall E$$

The computation rule simply performs the required substitution.

$$(\lambda x{\in}\tau.\ M)\,t \implies [t/x]M$$

The existential quantifier $\exists x{\in}\tau.\ A(x)$ lies at the heart of constructive mathematics. This should be a proposition if $A(x)$ is a proposition under the assumption that x has type τ.

$$\frac{\tau\ \mathit{type} \qquad \Gamma, x{\in}\tau \vdash A(x)\ \mathit{prop}}{\Gamma \vdash \exists x{\in}\tau.\ A(x)\ \mathit{prop}}\,\exists F$$

The introduction rule requires that we have a *witness* term t and a proof that t satisfies property A.

$$\frac{\Gamma \vdash t \in \tau \qquad \Gamma \vdash A(t)\ \mathit{true}}{\Gamma \vdash \exists x{\in}\tau.\ A(x)\ \mathit{true}}\,\exists I$$

The elimination rule bears some resemblance to disjunction: if we know that we have a verification of $\exists x{\in}\tau.\ A(x)$ we do not know the witness t. As a result we cannot simply write a rule of the form

$$\frac{\Gamma \vdash \exists x{\in}\tau.\ A(x)\ \mathit{true}}{\Gamma \vdash t \in \tau}\,\exists E?$$

since we have no way of referring to the proper t. Instead we reason as follows: If $\exists x{\in}\tau.\ A(x)$ is true, then there is some element of τ for which A holds. Call this element x and assume $A(x)$. Whatever we derive from this assumption must be true, as long as it does not depend on x itself.

$$\frac{\Gamma \vdash \exists x{\in}\tau.\ A(x)\ \mathit{true} \qquad \Gamma, x{\in}\tau, A(x)\ \mathit{true} \vdash C\ \mathit{true}}{\Gamma \vdash C\ \mathit{true}}\,\exists E$$

The derivation of the second premise is parametric in x and hypothetical in $A(x)$, that is, x may not occur in Γ or C.

The proof term assignment and computational contents of these rules is not particularly difficult. The proof term for an existential introduction is a pair

consisting of the witness t and the proof that t satisfies the stated property. The elimination rule destructs the pair, making the components accessible.

$$\frac{\Gamma \vdash t \in \tau \qquad \Gamma \vdash M : A(t)}{\Gamma \vdash \langle t, M\rangle : \exists x{\in}\tau.\ A(x)}\,\exists I$$

$$\frac{\Gamma \vdash M : \exists x{\in}\tau.\ A(x) \qquad \Gamma, x{\in}\tau, u{:}A(x) \vdash N : C}{\Gamma \vdash \mathbf{let}\ \langle x, u\rangle = M\ \mathbf{in}\ N : C}\,\exists E$$

The reduction rule is straightforward, substituting both the witness and the proof term certifying its correctness.

$$\mathbf{let}\langle x, u\rangle = \langle t, M\rangle\ \mathbf{in}\ N \quad\Longrightarrow\quad [M/u]\,[t/x]\,N$$

As in the case of the propositional connectives, we now consider various interactions between quantifiers and connectives to obtain an intuition regarding their properties. We continue to denote a proposition A that depends on a variable x by $A(x)$.

Our first example states that universal quantification distributes over conjunction. In order to make it fit on the page, we have abbreviated $u{:}\forall x{\in}\tau.\ A(x){\wedge}B(x)$ by $u{:}-$. Furthermore, we named the parameter introduced into the derivation a (rather than x), to emphasize the distinction between a bound variable in a proposition and a parameter which is bound in a derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a{\in}\tau, u{:}- \vdash \forall x{\in}\tau.\ A(x)\wedge B(x)\ \mathit{true}}\,u \qquad \dfrac{}{a{\in}\tau, u{:}- \vdash a \in \tau}\,a}{u{:}-, a{\in}\tau \vdash A(a)\wedge B(a)\ \mathit{true}}\,\forall E}{u{:}-, a{\in}\tau \vdash A(a)\ \mathit{true}}\,\wedge E_L}{u{:}- \vdash \forall x{\in}\tau.\ A(x)\ \mathit{true}}\,\forall I^a}{\vdash (\forall x{\in}\tau.\ A(x)\wedge B(x)) \supset (\forall x{\in}\tau.\ A(x))\ \mathit{true}}\,\supset I^u$$

The lists of hypotheses of the form $x{\in}\tau$ and $u{:}A$ in each line of a natural deduction can be reconstructed, so we will use the following abbreviated form familiar from the early development of propositional logic.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall x{\in}\tau.\ A(x)\wedge B(x)\ \mathit{true}}\,u \qquad \dfrac{}{a \in \tau}\,a}{A(a)\wedge B(a)\ \mathit{true}}\,\forall E}{A(a)\ \mathit{true}}\,\wedge E_L}{\forall x{\in}\tau.\ A(x)\ \mathit{true}}\,\forall I^a}{(\forall x{\in}\tau.\ A(x)\wedge B(x)) \supset (\forall x{\in}\tau.\ A(x))\ \mathit{true}}\,\supset I^u$$

From this deduction it is easy to see that

$$(\forall x{\in}\tau.\ A(x)\wedge B(x)) \supset (\forall x{\in}\tau.\ A(x)) \wedge (\forall x{\in}\tau.\ B(x))\ \mathit{true}$$

By annotating the derivation above we can construct the following proof term for this judgment (omitting some labels):

$$\begin{array}{ll} \vdash & \lambda u.\ \langle \lambda x{\in}\tau.\ \mathbf{fst}\,(u\,x), \lambda x{\in}\tau.\ \mathbf{snd}\,(u\,x)\rangle \\ : & (\forall x{\in}\tau.\ A(x) \wedge B(x)) \supset (\forall x{\in}\tau.\ A(x)) \wedge (\forall x{\in}\tau.\ B(x)) \end{array}$$

The opposite direction also holds, which means that we can freely move the universal quantifier over conjunctions and vice versa. This judgment (and also the proof above) are parametric in τ. Any instance by a concrete type for τ will be an evident judgment. We show here only the proof term (again omitting some labels):

$$\begin{array}{ll} \vdash & \lambda p.\ \lambda x{\in}\tau.\ \langle (\mathbf{fst}\ p)\ x, (\mathbf{snd}\ p)\ x\rangle \\ : & (\forall x{\in}\tau.\ A(x)) \wedge (\forall x{\in}\tau.\ B(x)) \supset (\forall x{\in}\tau.\ A(x) \wedge B(x)) \end{array}$$

The corresponding property for the existential quantifier allows distributing the existential quantifier over disjunction.

$$(\exists x{\in}\tau.\ A(x) \vee B(x)) \equiv (\exists x{\in}\tau.\ A(x)) \vee (\exists x{\in}\tau.\ B(x))$$

We verify one direction.

$$\dfrac{\dfrac{\dfrac{}{\exists x{\in}\tau.\ A(x) \vee B(x)\ \mathit{true}}\,u \qquad \begin{array}{c}\mathcal{D}\\ (\exists x{\in}\tau.\ A(x)) \vee (\exists x{\in}\tau.\ B(x))\ \mathit{true}\end{array}}{(\exists x{\in}\tau.\ A(x)) \vee (\exists x{\in}\tau.\ B(x))\ \mathit{true}}\,\exists E^{a,w}}{(\exists x{\in}\tau.\ A(x) \vee B(x)) \supset (\exists x{\in}\tau.\ A(x)) \vee (\exists x{\in}\tau.\ B(x))\ \mathit{true}}\,\supset I^u$$

where the deduction $\mathcal{D}$ is the following

$$\dfrac{\dfrac{}{A(a) \vee B(a)\ \mathit{true}}\,w \qquad \dfrac{\dfrac{\dfrac{}{a \in \tau}\,a \qquad \dfrac{}{A(a)\ \mathit{true}}\,v_1}{\exists x{\in}\tau.\ A(x)\ \mathit{true}}\,\exists I}{(\exists x{\in}\tau.\ A(x)) \vee (\exists x{\in}\tau.\ B(x))\ \mathit{true}}\,\vee I_L \qquad \vdots}{(\exists x{\in}\tau.\ A(x)) \vee (\exists x{\in}\tau.\ B(x))\ \mathit{true}}\,\vee E^{v_1,v_2}$$

The omitted derivation of the second case in the disjunction elimination is symmetric to the given case and ends in $\vee I_R$.

It is important to keep in mind the restriction on the existential elimination rule, namely that the parameter must be new in the second premise. The following is an incorrect derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a \in \mathbf{nat}}\,a?}{\mathbf{s}(a) \in \mathbf{nat}}\,\mathbf{nat}I_s \qquad \dfrac{\dfrac{}{\exists x{\in}\mathbf{nat}.\ A(\mathbf{s}(x))\ \mathit{true}}\,u \qquad \dfrac{}{A(\mathbf{s}(a))\ \mathit{true}}\,w}{A(\mathbf{s}(a))\ \mathit{true}}\,\exists E^{a,w}?}{\exists y{\in}\mathbf{nat}.\ A(y)\ \mathit{true}}\,\exists I}{(\exists x{\in}\mathbf{nat}.\ A(\mathbf{s}(x))) \supset \exists y{\in}\mathbf{nat}.\ A(y)\ \mathit{true}}\,\supset I^u$$

The problem can be seen in the two questionable rules. In the existential introduction, the term a has not yet been introduced into the derivation and its use can therefore not be justified. Related is the incorrect application of the $\exists E$ rule. It is supposed to introduce a new parameter a and a new assumption w. However, a occurs in the conclusion, invalidating this inference.

In this case, the flaw can be repaired by moving the existential elimination downward, in effect introducing the parameter into the derivation earlier (when viewed from the perspective of normal proof construction).

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\exists x{\in}\mathbf{nat}.\ A(\mathbf{s}(x))\ true}\,u \qquad \dfrac{\dfrac{\dfrac{}{a \in \mathbf{nat}}\,a}{\mathbf{s}(a) \in \mathbf{nat}}\,\mathbf{nat}I_s \qquad \dfrac{}{A(\mathbf{s}(a))\ true}\,w}{\exists y{\in}\mathbf{nat}.\ A(y)\ true}\,\exists I}{\exists y{\in}\mathbf{nat}.\ A(y)\ true}\,\exists E^{a,w}}{(\exists x{\in}\mathbf{nat}.\ A(\mathbf{s}(x))) \supset \exists y{\in}\mathbf{nat}.\ A(y)\ true}\,\supset I^u$$

Of course there are other cases where the flawed rule cannot be repaired. For example, it is easy to construct an incorrect derivation of $(\exists x{\in}\tau.\ A(x)) \supset \forall x{\in}\tau.\ A(x)$.

Bibliography

[CR36] Alonzo Church and J.B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, May 1936.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980. Hitherto unpublished note of 1969, rearranged, corrected, and annotated by Howard.

[ML80] Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, pages 153–175. North-Holland, 1980.

[ML96] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.