3.2 Reduction

In the preceding section, we have introduced the assignment of proof terms to natural deductions. If proofs are programs then we need to explain how proofs are to be executed, and which results may be returned by a computation.

We explain the operational interpretation of proofs in two steps. In the first step we introduce a judgment of *reduction* $M \Longrightarrow_R M'$, read "*M reduces to M'*". A computation then proceeds by a sequence of reductions $M \Longrightarrow_R M_1 \Longrightarrow_R M_2 \ldots$, according to a fixed strategy, until we reach a value which is the result of the computation. In this section we cover reduction; we may return to reduction strategies in a later lecture.

As in the development of propositional logic, we discuss each of the connectives separately, taking care to make sure the explanations are independent. This means we can consider various sublanguages and we can later extend our logic or programming language without invalidating the results from this section. Furthermore, it greatly simplifies the analysis of properties of the reduction rules.

In general, we think of the proof terms corresponding to the introduction rules as the *constructors* and the proof terms corresponding to the elimination rules as the *destructors*.

Conjunction. The constructor forms a pair, while the destructors are the left and right projections. The reduction rules prescribe the actions of the projections.

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

Truth. The constructor just forms the unit element, $\langle\,\rangle$. Since there is no destructor, there is no reduction rule.

Implication. The constructor forms a function by λ-abstraction, while the destructor applies the function to an argument. In general, the application of a function to an argument is computed by *substitution*. As a simple example from mathematics, consider the following equivalent definitions

$$f(x) = x^2 + x - 1 \qquad f = \lambda x.\, x^2 + x - 1$$

and the computation

$$f(3) = (\lambda x.\, x^2 + x - 1)(3) = [3/x](x^2 + x - 1) = 3^2 + 3 - 1 = 11$$

In the second step, we substitute 3 for occurrences of x in $x^2 + x - 1$, the *body of the λ-expression*. We write $[3/x](x^2 + x - 1) = 3^2 + 3 - 1$.

In general, the notation for the substitution of N for occurrences of u in M is $[N/u]M$. We therefore write the reduction rule as

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

We have to be somewhat careful so that substitution behaves correctly. In particular, no variable in N should be bound in M in order to avoid conflict. We can always achieve this by renaming bound variables—an operation which clearly does not change the meaning of a proof term.

Disjunction. The constructors inject into a sum types; the destructor distinguishes cases. We need to use substitution again.

$$\begin{array}{lcl} \mathbf{case}\,\mathbf{inl}^B\,M\,\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow_R & [M/u]N \\ \mathbf{case}\,\mathbf{inr}^A\,M\,\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow_R & [M/w]O \end{array}$$

Falsehood. Since there is no constructor for the empty type there is no reduction rule for falsehood.

This concludes the definition of the reduction judgment. In the next section we will prove some of its properties.

As an example we consider a simple program for the composition of two functions. It takes a pair of two functions, one from A to B and one from B to C and returns their composition which maps A directly to C.

$$\mathsf{comp} \quad : \quad ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$$

We transform the following implicit definition into our notation step-by-step:

$$\begin{array}{rcl} \mathsf{comp}\,\langle f, g\rangle\,(w) & = & g(f(w)) \\ \mathsf{comp}\,\langle f, g\rangle & = & \lambda w.\ g(f(w)) \\ \mathsf{comp}\,u & = & \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)(w)) \\ \mathsf{comp} & = & \lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) \end{array}$$

The final definition represents a correct proof term, as witnessed by the following deduction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{u : (A \supset B) \wedge (B \supset C)}\,u}{\mathbf{snd}\,u : B \supset C}\,\wedge E_R \qquad \dfrac{\dfrac{\dfrac{}{u : (A \supset B) \wedge (B \supset C)}\,u}{\mathbf{fst}\,u : A \supset B}\,\wedge E_L \quad \dfrac{}{w : A}\,w}{(\mathbf{fst}\,u)\,w : B}\,\supset E}{(\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) : C}\,\supset E}{\lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) : A \supset C}\,\supset I^w}{(\lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w)) : ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)}\,\supset I^u$$

We now verify that the composition of two identity functions reduces again to the identity function. First, we verify the typing of this application.

$$(\lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w))\,\langle(\lambda x.\ x), (\lambda y.\ y)\rangle \quad : \quad A \supset A$$

Now we show a possible sequence of reduction steps. This is by no means uniquely determined.

$$\begin{array}{ll} & (\lambda u.\ \lambda w.\ (\mathbf{snd}\, u)\, ((\mathbf{fst}\, u)\, w))\, \langle (\lambda x.\ x), (\lambda y.\ y)\rangle \\ \Longrightarrow_R & \lambda w.\ (\mathbf{snd}\, \langle (\lambda x.\ x), (\lambda y.\ y)\rangle)\, ((\mathbf{fst}\, \langle (\lambda x.\ x), (\lambda y.\ y)\rangle)\, w) \\ \Longrightarrow_R & \lambda w.\ (\lambda y.\ y)\, ((\mathbf{fst}\, \langle (\lambda x.\ x), (\lambda y.\ y)\rangle)\, w) \\ \Longrightarrow_R & \lambda w.\ (\lambda y.\ y)\, ((\lambda x.\ x)\, w) \\ \Longrightarrow_R & \lambda w.\ (\lambda y.\ y)\, w \\ \Longrightarrow_R & \lambda w.\ w \end{array}$$

We see that we may need to apply reduction steps to subterms in order to reduce a proof term to a form in which it can no longer be reduced. We postpone a more detailed discussion of this until we discuss the operational semantics in full.

3.3 Expansion

We saw in the previous section that proof reductions that witness local soundness form the basis for the computational interpretation of proofs. Less relevant to computation are the local expansions. What they tell us, for example, is that if we need to return a pair from a function, we can always construct it as $\langle M, N\rangle$ for some M and N. Another example would be that whenever we need to return a function, we can always construct it as $\lambda u.\ M$ for some M.

We can derive what the local expansion must be by annotating the deduction from Section 2.5 with proof terms. We leave this as an exercise to the reader. The left-hand side of each expansion has the form $M : A$, where M is an arbitrary term and A is a logical connective or constant applied to arbitrary propositions. On the right hand side we have to apply a destructor to M and then reconstruct a term of the original type. The resulting rules can be found in Figure 3.3.

3.4 Summary of Proof Terms

Judgments.

$M : A$	M is a proof term for proposition A, see Figure 3.1
$M \Longrightarrow_R M'$	M reduces to M', see Figure 3.2
$M : A \Longrightarrow_E M'$	M expands to M', see Figure 3.3

Concrete Syntax. The concrete syntax for proof terms used in the mechanical proof checker has some minor differences to the form we presented above.

Constructors	Destructors
$\dfrac{M : A \quad N : B}{\langle M, N\rangle : A \wedge B}\ \wedge I$	$\dfrac{M : A \wedge B}{\mathbf{fst}\, M : A}\ \wedge E_L$ $\dfrac{M : A \wedge B}{\mathbf{snd}\, M : B}\ \wedge E_R$
$\dfrac{}{\langle\,\rangle : \top}\ \top I$	no destructor for $\top$
$\dfrac{\begin{array}{c}\overline{u : A}\ u \\ \vdots \\ M : B\end{array}}{\lambda u{:}A.\ M : A \supset B}\ \supset I^u$	$\dfrac{M : A \supset B \quad N : A}{M\, N : B}\ \supset E$
$\dfrac{M : A}{\mathbf{inl}^B\, M : A \vee B}\ \vee I_L$ $\dfrac{N : B}{\mathbf{inr}^A\, N : A \vee B}\ \vee I_R$	$\dfrac{M : A \vee B \quad \begin{array}{c}\overline{u : A}\ u \\ \vdots \\ N : C\end{array} \quad \begin{array}{c}\overline{w : B}\ w \\ \vdots \\ O : C\end{array}}{\mathbf{case}\, M\, \mathbf{of}\, \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O : C}\ \vee E^{u,w}$
no constructor for $\bot$	$\dfrac{M : \bot}{\mathbf{abort}^C\, M : C}\ \bot E$

Figure 3.1: Proof term assignment for natural deduction

$$
\begin{array}{rcl}
\mathbf{fst}\,\langle M, N\rangle & \Longrightarrow_R & M \\
\mathbf{snd}\,\langle M, N\rangle & \Longrightarrow_R & N \\
\text{no reduction for } \langle\,\rangle & & \\
(\lambda u{:}A.\ M)\,N & \Longrightarrow_R & [N/u]M \\
\mathbf{case}\,\mathbf{inl}^B\,M\ \mathbf{of}\ \mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow_R & [M/u]N \\
\mathbf{case}\,\mathbf{inr}^A\,M\ \mathbf{of}\ \mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow_R & [M/w]O \\
\text{no reduction for } \mathbf{abort} & &
\end{array}
$$

Figure 3.2: Proof term reductions

$$
\begin{array}{lcl}
M : A \wedge B & \Longrightarrow_E & \langle \mathbf{fst}\,M, \mathbf{snd}\,M\rangle \\
M : A \supset B & \Longrightarrow_E & \lambda u{:}A.\ M\,u \quad \text{for } u \text{ not free in } M \\
M : \top & \Longrightarrow_E & \langle\,\rangle \\
M : A \vee B & \Longrightarrow_E & \mathbf{case}\,M\ \mathbf{of}\ \mathbf{inl}\,u \Rightarrow \mathbf{inl}^B\,u \mid \mathbf{inr}\,w \Rightarrow \mathbf{inr}^A\,w \\
M : \bot & \Longrightarrow_E & \mathbf{abort}^{\bot}\,M
\end{array}
$$

Figure 3.3: Proof term expansions

u	`u`	Variable
$\langle M, N\rangle$	`(M,N)`	Pair
$\mathbf{fst}\,M$	`fst M`	First projection
$\mathbf{snd}\,M$	`snd M`	Second projection
$\langle\,\rangle$	`()`	Unit element
$\lambda u{:}A.\ M$	`fn u => M`	Abstraction
$M\,N$	`M N`	Application
$\mathbf{inl}^B\,M$	`inl M`	Left injection
$\mathbf{inr}^A\,N$	`inr N`	Right injection
$\mathbf{case}\,M$	`case M`	Case analysis
$\quad\mathbf{of}\ \mathbf{inl}\,u \Rightarrow N$	`  of inl u => N`	
$\quad\mid \mathbf{inr}\,w \Rightarrow O$	`   | inr w => O`	
	`end`	
$\mathbf{abort}^C\,M$	`abort M`	Abort

Pairs and unit element are delimited by parentheses '`(`' and '`)`' instead of angle brackets $\langle$ and $\rangle$. The `case` constructs requires an `end` token to mark the end of the a sequence of cases.

Type annotations are generally omitted, but a whole term can explicitly be

given a type. The proof checker (which here is also a type checker) infers the missing information. Occasionally, an explicit type ascription `M : A` is necessary as a hint to the type checker.

For rules of operator precedence, the reader is referred to the on-line documentation of the proof checking software available with the course material. Generally, parentheses can be used to disambiguate or override the standard rules.

As an example, we show the proof term implementing function composition.

```
term comp : (A => B) & (B => C) => (A => C) =
fn u => fn x => (snd u) ((fst u) x);
```

We also allow annotated deductions, where each line is annotated with a proof term. This is a direct transcription of deduction for judgments of the form $M : A$. As an example, we show the proof that $A \vee B \supset B \vee A$, first in the pure form.

```
proof orcomm : A | B => B | A =
begin
[ A | B;
  [ A;
    B | A];
  [ B;
    B | A];
  B | A ];
A | B => B | A
end;
```

Now we systematically annotate each line and obtain

```
annotated proof orcomm : A | B => B | A =
begin
[ u : A | B;
  [ v : A;
    inr v : B | A];
  [ w : B;
    inl w : B | A];
  case u
    of inl v => inr v
     | inr w => inl w
   end : B | A ];
fn u => case u
          of inl v => inr v
           | inr w => inl w
         end : A | B => B | A
end;
```

3.5 Properties of Proof Terms

In this section we analyze and verify various properties of proof terms. Rather than concentrate on reasoning within the logical calculi we introduced, we now want to reason about them. The techniques are very similar—they echo the ones we have introduced so far in natural deduction. This should not be surprising. After all, natural deduction was introduced to model mathematical reasoning, and we now engage in some mathematical reasoning about proof terms, propositions, and deductions. We refer to this as *metalogical reasoning*.

First, we need some more formal definitions for certain operations on proof terms, to be used in our meta-logical analysis. One rather intuitive property of is that variable names should not matter. For example, the identity function at type A can be written as $\lambda u{:}A.\ u$ or $\lambda w{:}A.\ w$ or $\lambda u'{:}A.\ u'$, etc. They all denote the same function and the same proof. We therefore identify terms which differ only in the names of variables (here called u) bound in $\lambda u{:}A.\ M$, $\mathbf{inl}\, u \Rightarrow M$ or $\mathbf{inr}\, u \Rightarrow O$. But there are pitfalls with this convention: variables have to be renamed *consistently* so that every variable refers to the same binder before and after the renaming. For example (omitting type labels for brevity):

$$\begin{array}{rcl} \lambda u.\ u & = & \lambda w.\ w \\ \lambda u.\ \lambda w.\ u & = & \lambda u'.\ \lambda w.\ u' \\ \lambda u.\ \lambda w.\ u & \neq & \lambda u.\ \lambda w.\ w \\ \lambda u.\ \lambda w.\ u & \neq & \lambda w.\ \lambda w.\ w \\ \lambda u.\ \lambda w.\ w & = & \lambda w.\ \lambda w.\ w \end{array}$$

The convention to identify terms which differ only in the naming of their bound variables goes back to the first papers on the λ-calculus by Church and Rosser [CR36], is called the "*variable name convention*" and is pervasive in the literature on programming languages and λ-calculi. The term λ-calculus typically refers to a pure calculus of functions formed with λ-abstraction. Our proof term calculus is called a *typed λ-calculus* because of the presence of propositions (which an be viewed as types).

Following the variable name convention, we may silently rename when convenient. A particular instance where this is helpful is substitution. Consider

$$[u/w](\lambda u.\ w\, u)$$

that is, we substitute u for w in $\lambda u.\ w\, u$. Note that u is a variable visible on the outside, but also bound by λu. By the variable name convention we have

$$[u/w](\lambda u.\ w\, u) = [u/w](\lambda u'.\ w\, u') = \lambda u'.\ u\, u'$$

which is correct. But we cannot substitute without renaming, since

$$[u/w](\lambda u.\ w\, u) \neq \lambda u.\ u\, u$$

In fact, the right hand side below is invalid, while the left-hand side makes perfect sense. We say that u is *captured* by the binder λu. If we assume a hypothesis $u{:}\top \supset A$ then

$$[u/w](\lambda u{:}\top.\ w\, u) : A$$

but

$$\lambda u{:}\top.\ u\,u$$

is not well-typed since the first occurrence of u would have to be of type $\top \supset A$ but instead has type $\top$.

So when we carry out substitution $[M/u]N$ we need to make sure that no variable in M is *captured* by a binder in N, leading to an incorrect result. Fortunately we can always achieve that by renaming some bound variables in N if necessary. We could now write down a formal definition of substitution, based on the cases for the term we are substituting into. However, we hope that the notion is sufficiently clear that this is not necessary.

Instead we revisit the substitution principle for hypothetical judgments. It states that if we have a hypothetical proof of C *true* from A *true* and we have a proof of A *true*, we can substitute the proof of A *true* for uses of the hypothesis A *true* and obtain a (non-hypothetical) proof of A *true*. In order to state this more precisely in the presence of several hypotheses, we recall that

$$\begin{array}{c} A_1\ \mathit{true} \ldots A_n\ \mathit{true} \\ \vdots \\ C\ \mathit{true} \end{array}$$

can be written as

$$\underbrace{A_1\ \mathit{true}, \ldots, A_n\ \mathit{true}}_{\Delta} \vdash C\ \mathit{true}$$

Generally we abbreviate several hypotheses by Δ. We then have the following properties, evident from the very definition of hypothetical judgments and hypothetical proofs

Weakening: If $\Delta \vdash C\ \mathit{true}$ then $\Delta, \Delta' \vdash C\ \mathit{true}$.

Substitution: If $\Delta, A\ \mathit{true}, \Delta' \vdash C\ \mathit{true}$ and $\Delta \vdash A\ \mathit{true}$ then $\Delta, \Delta' \vdash C\ \mathit{true}$.

As indicated above, weakening is realized by adjoining unused hypotheses, substitutions is realized by substitution of proofs for hypotheses.

For the proof term judgment, $M : A$, we use the same notation and write

$$\begin{array}{c} u_1{:}A_1\ \ldots\ u_n{:}A_n \\ \vdots \\ N : C \end{array}$$

as

$$\underbrace{u_1{:}A_1, \ldots, u_n{:}A_n}_{\Gamma} \vdash N : C$$

We use Γ to refer to collections of hypotheses $u_i{:}A_i$. In the deduction of $N : C$, each u_i stands for an unknown proof term for A_i, simply assumed to exist. If we actually find a proof $M_i{:}A_i$ we can eliminate this assumption, again by substitution. However, this time, the substitution has to perform two operations:

we have to substitute M_i for u_i (the unknown proof term variable), and the deduction of $M_i : A_i$ for uses of the hypothesis $u_i{:}A_i$. More precisely, we have the following two properties:

Weakening: If $\Gamma, \Gamma' \vdash N : C$ then $\Gamma, u{:}A, \Gamma' \vdash N : C$.

Substitution: If $\Gamma, u{:}A, \Gamma' \vdash N : C$ and $\Gamma \vdash M : A$ then $\Gamma, \Gamma' \vdash [M/u]N : C$.

Now we are in a position to state and prove our second meta-theorem, that is, a theorem about the logic under consideration. The theorem is called *subject reduction* because is concerns the *subject* M of the judgment $M : A$. It states that reduction preserves the type of an object. We make the hypotheses explicit as we have done in the explanations above.

Theorem 3.1 (Subject Reduction)
If $\Gamma \vdash M : A$ *and* $M \Longrightarrow_R M'$ *then* $\Gamma \vdash M' : A$.

Proof: We consider each case in the definition of $M \Longrightarrow_R M'$ in turn and show that the property holds. This is simply an instance of *proof by cases*.

Case: $\mathbf{fst}\,\langle M_1, M_2\rangle \Longrightarrow_R M_1$. By assumption we also know that

$$\Gamma \vdash \mathbf{fst}\,\langle M_1, M_2\rangle : A.$$

We need to show that $\Gamma \vdash M_1 : A$.

Now we inspect all inference rules for the judgment $M : A$ and we see that there is only one way how the judgment above could have been inferred: by $\wedge E_L$ from

$$\Gamma \vdash \langle M_1, M_2\rangle : A \wedge A_2$$

for some A_2. This step is called *inversion*, since we infer the premises from the conclusion of the rule. But we have to be extremely careful to inspect all possibilities for derivations so that we do not forget any cases.

Next, we apply inversion again: the judgment above could only have been inferred by $\wedge I$ from the two premises

$$\Gamma \vdash M_1 : A$$

and

$$\Gamma \vdash M_2 : A_2$$

But the first of these is what we had to prove in this case and we are done.

Case: $\mathbf{snd}\,\langle M_1, M_2\rangle \Longrightarrow_R M_2$. This is symmetric to the previous case. We write it an abbreviated form.

$\Gamma \vdash \mathbf{snd}\,\langle M_1, M_2\rangle : A$	Assumption
$\Gamma \vdash \langle M_1, M_2\rangle : A_1 \wedge A$ for some A_1	By inversion
$\Gamma \vdash M_1 : A_1$ and	
$\Gamma \vdash M_2 : A$	By inversion

Here the last judgment is what we were trying to prove.

Case: There is no reduction for $\top$ since there is no elimination rule and hence no destructor.

Case: $(\lambda u{:}A_1.\ M_2)\, M_1 \Longrightarrow_R [M_1/u]M_2$. By assumption we also know that

$$\Gamma \vdash (\lambda u{:}A_1.\ M_2)\, M_1 : A.$$

We need to show that $\Gamma \vdash [M_1/u]M_2 : A$.

Since there is only one inference rule for function application, namely implication elimination ($\supset E$), we can apply inversion and find that

$$\Gamma \vdash (\lambda u{:}A_1.\ M_2) : A_1' \supset A$$

and

$$\Gamma \vdash M_1 : A_1'$$

for some A_1'. Now we repeat inversion on the first of these and conclude that

$$\Gamma, u{:}A_1 \vdash M_2 : A$$

and, moreover, that $A_1 = A_1'$. Hence

$$\Gamma \vdash M_1 : A_1$$

Now we can apply the substitution property to these to judgments to conclude

$$\Gamma \vdash [M_1/u]M_2 : A$$

which is what we needed to show.

Case: $(\mathbf{case}\,\mathbf{inl}^C\, M_1\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) \Longrightarrow_R [M_1/u]N$. By assumption we also know that

$$\Gamma \vdash (\mathbf{case}\,\mathbf{inl}^C\, M_1\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) : A$$

Again we apply inversion and obtain three judgments

$$\begin{array}{l}\Gamma \vdash \mathbf{inl}^C\, M_1 : B' \vee C' \\ \Gamma, u{:}B' \vdash N : A \\ \Gamma, w{:}C' \vdash O : A\end{array}$$

for some B' and C'.

Again by inversion on the first of these, we find

$$\Gamma \vdash M_1 : B'$$

and also $C' = C$. Hence we can apply the substitution property to get

$$\Gamma \vdash [M_1/u]N : A$$

which is what we needed to show.

Case: $(\mathbf{case}\,\mathbf{inr}^B\,M_1\ \mathbf{of}\ \mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O) \Longrightarrow_R [M_1/u]N$. This is symmetric to the previous case and left as an exercise.

Case: There is no introduction rule for $\bot$ and hence no reduction rule.

□

The important techniques introduced in the proof above are *proof by cases* and *inversion*. In a proof by cases we simply consider all possibilities for why a judgment could be evident and show the property we want to establish in each case. Inversion is very similar: from the shape of the judgment we see it could have been inferred only in one possible way, so we know the premises of this rule must also be evident. We see that these are just two slightly different forms of the same kind of reasoning.

If we look back at our early example computation, we saw that the reduction step does not always take place at the top level, but that the redex may be embedded in the term. In order to allow this, we need to introduce some additional ways to establish that $M \Longrightarrow_R M'$ when the actual reduction takes place *inside* M. This is accomplished by so-called *congruence rules*.

Bibliography

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

[Dum91] Michael Dummett. *The Logical Basis of Metaphysics*. Harvard University Press, Cambridge, Massachusetts, 1991. The William James Lectures, 1976.

[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.

[Pra65] Dag Prawitz. *Natural Deduction*. Almquist & Wiksell, Stockholm, 1965.