Lecture Notes on
Proofs as Programs

15-816: Modal Logic
Frank Pfenning

Lecture 2
January 14, 2010

1 Introduction

In this lecture we investigate a computational interpretation of intuitionistic proofs and relate it to functional programming. On the propositional fragment of logic this is called the Curry-Howard isomorphism [How80]. From the very outset of the development of constructive logic and mathematics, a central idea has been that proofs ought to represent constructions. The Curry-Howard isomorphism is only a particularly poignant and beautiful realization of this idea. In a highly influential subsequent paper, Martin-Löf [ML80] developed it further into a more expressive calculus called *type theory*.

The computational interpretation of intuitionistic logic and type theory underlies many of the applications of intuitionistic modal logic in computer science. In the context of this course it therefore important to gain a good working knowledge of the interpretation of proofs as programs.

2 Propositions as Types

In order to illustrate the relationship between proofs and programs we introduce a new judgment:

$M : A$ $\quad$ M is a proof term for proposition A

We presuppose that A is a proposition when we write this judgment. We will also interpret $M : A$ as "*M is a program of type A*". These dual interpretations of the same judgment is the core of the Curry-Howard isomorphism. We either think of M as a term that represents the proof of A *true*, or we think of A as the type of the program M. As we discuss each connective, we give both readings of the rules to emphasize the analogy.

We intend that if $M : A$ then A *true*. Conversely, if A *true* then $M : A$. But we want something more: every deduction of $M : A$ should correspond to a deduction of A *true* with an identical structure and vice versa. In other words we annotate the inference rules of natural deduction with proof terms. The property above should then be obvious.

Conjunction. Constructively, we think of a proof of $A \wedge B$ *true* as a pair of proofs: one for A *true* and one for B *true*.

$$\frac{M : A \quad N : B}{\langle M, N \rangle : A \wedge B} \wedge I$$

The elimination rules correspond to the projections from a pair to its first and second elements.

$$\frac{M : A \wedge B}{\pi_1\, M : A} \wedge E_L \qquad \frac{M : A \wedge B}{\pi_2\, M : B} \wedge E_R$$

Hence conjunction $A \wedge B$ corresponds to the product type $A \times B$.

Truth. Constructively, we think of a proof of $\top$ *true* as a unit element that carries now information.

$$\frac{}{\langle\, \rangle : \top} \top I$$

Hence $\top$ corresponds to the unit type **1** with one element. There is no elimination rule and hence no further proof term constructs for truth.

Implication. Constructively, we think of a proof of $A \supset B$ *true* as a function which transforms a proof of A *true* into a proof of B *true*.

In mathematics and many programming languages, we define a function f of a variable x by writing $f(x) = \ldots$ where the right-hand side "…" depends on x. For example, we might write $f(x) = x^2 + x - 1$. In functional programming, we can instead write $f = \lambda x.\, x^2 + x - 1$, that is, we explicitly form a functional object by *λ-abstraction* of a variable (x, in the example).

We now use the notation of λ-abstraction to annotate the rule of implication introduction with proof terms. In the official syntax, we label the abstraction with a proposition (writing $\lambda u{:}A$) in order to specify the domain of a function unambiguously. In practice we will often omit the label to make expressions shorter—usually (but not always!) it can be determined from the context.

$$\cfrac{\begin{array}{c}\overline{u : A}^{\,u}\\ \vdots \\ M : B\end{array}}{\lambda u{:}A.\, M : A \supset B}\;{\supset}I^u$$

The hypothesis label u acts as a variable, and any use of the hypothesis labeled u in the proof of B corresponds to an occurrence of u in M.

As a concrete example, consider the (trivial) proof of $A \supset A$ *true*:

$$\cfrac{\overline{A\ \mathit{true}}^{\,u}}{A \supset A\ \mathit{true}}\;{\supset}I^u$$

If we annotate the deduction with proof terms, we obtain

$$\cfrac{\overline{u : A}^{\,u}}{(\lambda u{:}A.\, u) : A \supset A}\;{\supset}I^u$$

So our proof corresponds to the identity function id at type A which simply returns its argument. It can be defined with $\mathrm{id}(u) = u$ or $\mathrm{id} = (\lambda u{:}A.\, u)$.

The rule for implication elimination corresponds to function application. Following the convention in functional programming, we write $M\,N$ for the application of the function M to argument N, rather than the more verbose $M(N)$.

$$\cfrac{M : A \supset B \quad N : A}{M\,N : B}\;{\supset}E$$

What is the meaning of $A \supset B$ as a type? From the discussion above it should be clear that it can be interpreted as a function type $A \to B$. The introduction and elimination rules for implication can also be viewed as formation rules for functional abstraction $\lambda u{:}A.\, M$ and application $M\,N$.

Note that we obtain the usual introduction and elimination rules for implication if we erase the proof terms. This will continue to be true for all rules in the remainder of this section and is immediate evidence for the soundness of the proof term calculus, that is, if $M : A$ then A *true*.

As a second example we consider a proof of $(A \wedge B) \supset (B \wedge A)$ *true*.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge B\ \mathit{true}}\,u}{B\ \mathit{true}}\,\wedge E_R \quad \dfrac{\dfrac{}{A \wedge B\ \mathit{true}}\,u}{A\ \mathit{true}}\,\wedge E_L}{B \wedge A\ \mathit{true}}\,\wedge I}{(A \wedge B) \supset (B \wedge A)\ \mathit{true}}\,\supset I^u$$

When we annotate this derivation with proof terms, we obtain a function which takes a pair $\langle M, N\rangle$ and returns the reverse pair $\langle N, M\rangle$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{u : A \wedge B}\,u}{\pi_2\, u : B}\,\wedge E_R \quad \dfrac{\dfrac{}{u : A \wedge B}\,u}{\pi_1\, u : A}\,\wedge E_L}{\langle \pi_2\, u, \pi_1\, u\rangle : B \wedge A}\,\wedge I}{(\lambda u.\, \langle \pi_2\, u, \pi_1\, u\rangle) : (A \wedge B) \supset (B \wedge A)}\,\supset I^u$$

Disjunction. Constructively, we think of a proof of $A \vee B$ *true* as either a proof of A *true* or B *true*. Disjunction therefore corresponds to a disjoint sum type $A + B$, and the two introduction rules correspond to the left and right injection into a sum type.

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

In the official syntax, we have annotated the injections **inl** and **inr** with propositions B and A, again so that a (valid) proof term has an unambiguous type. In writing actual programs we usually omit this annotation. The elimination rule corresponds to a case construct which discriminates between a left and right injection into a sum types.

$$\dfrac{M : A \vee B \qquad \begin{array}{c}\dfrac{}{u : A}\,u \\ \vdots \\ N : C\end{array} \qquad \begin{array}{c}\dfrac{}{w : B}\,w \\ \vdots \\ O : C\end{array}}{\mathbf{case}\, M\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O : C}\,\vee E^{u,w}$$

Recall that the hypothesis labeled u is available only in the proof of the second premise and the hypothesis labeled w only in the proof of the third premise. This means that the scope of the variable u is N, while the scope of the variable w is O.

Falsehood. There is no introduction rule for falsehood ($\bot$). We can therefore view it as the empty type **0**. The corresponding elimination rule allows a term of $\bot$ to stand for an expression of any type when wrapped with **abort**. However, there is no computation rule for it, which means during computation of a valid program we will never try to evaluate a term of the form **abort** M.

$$\dfrac{M : \bot}{\textbf{abort}^C\, M : C}\ \bot E$$

As before, the annotation C which disambiguates the type of **abort** M will often be omitted.

This completes our assignment of proof terms to the logical inference rules. Now we can interpret the interaction laws we introduced early as programming exercises. Consider the following distributivity law:

(L11a) $(A \supset (B \wedge C)) \supset (A \supset B) \wedge (A \supset C)$ *true*

Interpreted constructively, this assignment can be read as:

> Write a function which, when given a function from A to pairs of type $B \wedge C$, returns two functions: one which maps A to B and one which maps A to C.

This is satisfied by the following function:

$$\lambda u.\, \langle (\lambda w.\, \pi_1\,(u\,w)), (\lambda v.\, \pi_2\,(u\,v)) \rangle$$

The following deduction provides the evidence:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{u : A \supset (B \wedge C)}\,u \qquad \dfrac{}{w : A}\,w}{u\,w : B \wedge C}\,\supset E}{\pi_1\,(u\,w) : B}\,\wedge E_L}{\lambda w.\, \pi_1\,(u\,w) : A \supset B}\,\supset I^w \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{u : A \supset (B \wedge C)}\,u \qquad \dfrac{}{v : A}\,v}{u\,v : B \wedge C}\,\supset E}{\pi_2\,(u\,v) : C}\,\wedge E_R}{\lambda v.\, \pi_2\,(u\,v) : A \supset C}\,\supset I^v}{\langle (\lambda w.\, \pi_1\,(u\,w)), (\lambda v.\, \pi_2\,(u\,v)) \rangle : (A \supset B) \wedge (A \supset C)}\,\wedge I}{\lambda u.\, \langle (\lambda w.\, \pi_1\,(u\,w)), (\lambda v.\, \pi_2\,(u\,v)) \rangle : (A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))}\,\supset I^u$$

Programs in constructive propositional logic are somewhat uninteresting in that they do not manipulate basic data types such as natural numbers, integers, lists, trees, etc. These can be added, most elegantly in the context of a type theory where we allow arbitrary inductive types. This extension is somewhat orthogonal to the main aims of this course on modal logic so we will not treat it formally, but use it freely in examples.

To close this section we recall the guiding principles behind the assignment of proof terms to deductions.

1. For every deduction of A *true* there is a proof term M and deduction of $M : A$.

2. For every deduction of $M : A$ there is a deduction of A *true*

3. The correspondence between proof terms M and deductions of A *true* is a bijection.

3 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.

$$\begin{array}{lcl} \pi_1\,\langle M, N\rangle & \Longrightarrow_R & M \\ \pi_2\,\langle M, N\rangle & \Longrightarrow_R & N \end{array}$$

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. A more formal definition is presented in Section 8.

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:

$$\begin{array}{rcl}
\mathsf{comp}\,\langle f, g\rangle\,(w) & = & g(f(w)) \\
\mathsf{comp}\,\langle f, g\rangle & = & \lambda w.\, g(f(w)) \\
\mathsf{comp}\,u & = & \lambda w.\, (\pi_2\,u)\,((\pi_1\,u)(w)) \\
\mathsf{comp} & = & \lambda u.\, \lambda w.\, (\pi_2\,u)\,((\pi_1\,u)\,w)
\end{array}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{u : (A \supset B) \wedge (B \supset C)}\,u}{\pi_2\,u : B \supset C}\,\wedge E_R \qquad \dfrac{\dfrac{\dfrac{}{u : (A \supset B) \wedge (B \supset C)}\,u}{\pi_1\,u : A \supset B}\,\wedge E_L \quad \dfrac{}{w : A}\,w}{(\pi_1\,u)\,w : B}\,\supset E}{\dfrac{(\pi_2\,u)\,((\pi_1\,u)\,w) : C}{\lambda w.\,(\pi_2\,u)\,((\pi_1\,u)\,w) : A \supset C}\,\supset I^w}\,\supset E}{(\lambda u.\,\lambda w.\,(\pi_2\,u)\,((\pi_1\,u)\,w)) : ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)}\,\supset I^u$$

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

$$(\lambda u.\,\lambda w.\,(\pi_2\,u)\,((\pi_1\,u)\,w))\,\langle(\lambda x.\,x), (\lambda y.\,y)\rangle \quad : \quad A \supset A$$

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

$$\begin{array}{rl}
 & (\lambda u.\,\lambda w.\,(\pi_2\,u)\,((\pi_1\,u)\,w))\,\langle(\lambda x.\,x), (\lambda y.\,y)\rangle \\
\Longrightarrow_R & \lambda w.\,(\pi_2\,\langle(\lambda x.\,x), (\lambda y.\,y)\rangle)\,((\pi_1\,\langle(\lambda x.\,x), (\lambda y.\,y)\rangle)\,w) \\
\Longrightarrow_R & \lambda w.\,(\lambda y.\,y)\,((\pi_1\,\langle(\lambda x.\,x), (\lambda y.\,y)\rangle)\,w) \\
\Longrightarrow_R & \lambda w.\,(\lambda y.\,y)\,((\lambda x.\,x)\,w) \\
\Longrightarrow_R & \lambda w.\,(\lambda y.\,y)\,w \\
\Longrightarrow_R & \lambda w.\,w
\end{array}$$

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.

4 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 deductions witnessing local expansions from Lecture 1 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.

5 Summary of Proof Terms

Judgments.

$M : A$	M is a proof term for proposition A, see Figure 1
$M \Longrightarrow_R M'$	M reduces to M', see Figure 2
$M : A \Longrightarrow_E M'$	M expands to M', see Figure 3

6 Hypothetical Judgments in Localized Form

The isomorphism between proofs and programs seems obvious, but it is nonetheless a useful exercise to rigorously prove this relationship as a property of the deductive systems we have developed. When we proceed to a certain level of formality in our analysis, it is beneficial to write hypotheti-

Constructors	Destructors
$\dfrac{M : A \quad N : B}{\langle M, N\rangle : A \wedge B}\,\wedge I$	$\dfrac{M : A \wedge B}{\pi_1\, M : A}\,\wedge E_L$ $\dfrac{M : A \wedge B}{\pi_2\, M : B}\,\wedge E_R$
$\dfrac{}{\langle\,\rangle : \top}\,\top I$	no destructor for $\top$
$\dfrac{\begin{array}{c}\dfrac{}{u : A}\,u \\ \vdots \\ M : B\end{array}}{\lambda u{:}A.\, M : A \supset B}\,\supset I^u$	$\dfrac{M : A \supset B \quad N : A}{M\, N : B}\,\supset E$
$\dfrac{M : A}{\mathbf{inl}^B\, M : A \vee B}\,\vee I_L$ $\dfrac{N : B}{\mathbf{inr}^A\, N : A \vee B}\,\vee I_R$	$\dfrac{M : A \vee B \quad \begin{array}{c}\dfrac{}{u : A}\,u \\ \vdots \\ N : C\end{array} \quad \begin{array}{c}\dfrac{}{w : B}\,w \\ \vdots \\ O : C\end{array}}{\mathbf{case}\, M\ \mathbf{of\ inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O : C}\,\vee E^{u,w}$
no constructor for $\bot$	$\dfrac{M : \bot}{\mathbf{abort}^C\, M : C}\,\bot E$

Figure 1: Proof term assignment for natural deduction

$$
\begin{array}{rcl}
\pi_1 \, \langle M, N \rangle & \Longrightarrow_R & M \\
\pi_2 \, \langle M, N \rangle & \Longrightarrow_R & N \\
\text{no reduction for } \langle\,\rangle & & \\
(\lambda u{:}A.\, M)\, N & \Longrightarrow_R & [N/u]M \\
\textbf{case}\, \textbf{inl}^B\, M \textbf{ of inl}\, u \Rightarrow N \mid \textbf{inr}\, w \Rightarrow O & \Longrightarrow_R & [M/u]N \\
\textbf{case}\, \textbf{inr}^A\, M \textbf{ of inl}\, u \Rightarrow N \mid \textbf{inr}\, w \Rightarrow O & \Longrightarrow_R & [M/w]O \\
\text{no reduction for } \textbf{abort} & &
\end{array}
$$

Figure 2: Proof term reductions

$$
\begin{array}{lcl}
M : A \wedge B & \Longrightarrow_E & \langle \pi_1 \, M, \pi_2 \, M \rangle \\
M : A \supset B & \Longrightarrow_E & \lambda u{:}A.\, M\, u \quad \text{for } u \text{ not free in } M \\
M : \top & \Longrightarrow_E & \langle\,\rangle \\
M : A \vee B & \Longrightarrow_E & \textbf{case}\, M \textbf{ of inl}\, u \Rightarrow \textbf{inl}^B\, u \mid \textbf{inr}\, w \Rightarrow \textbf{inr}^A\, w \\
M : \bot & \Longrightarrow_E & \textbf{abort}^{\bot}\, M
\end{array}
$$

Figure 3: Proof term expansions

cal judgments

$$\begin{array}{ccc} J_1 & \cdots & J_n \\ & \vdots & \\ & J & \end{array}$$

in their localized form

$$J_1, \ldots, J_n \vdash J.$$

When we try to recast the two-dimensional rules, however, some ambiguities in the two-dimensional notation need to be clarified.

The first ambiguity can be seen from considering the question of how many verifications of $P \supset P \supset P$ there are. Clearly, there should be two: one using the first assumption and one using the second assumption. Indeed, after a few steps we arrive at the following situation

$$\dfrac{\dfrac{\begin{array}{c} \dfrac{}{P{\downarrow}}u \quad \dfrac{}{P{\downarrow}}w \\ \vdots \\ P{\uparrow} \end{array}}{P \supset P{\uparrow}} \supset I^w}{P \supset P \supset P{\uparrow}} \supset I^u$$

Now we can use the $\downarrow\uparrow$ rule with the hypothesis labeled u or the hypothesis labeled w. If we look at the hypothetical judgment that we still have to prove and write it as

$$P{\downarrow}, P{\downarrow} \vdash P{\uparrow}$$

then it is not clear if or how we should distinguish the two occurrences of $P{\downarrow}$. We can do this by labeling them in the local notation for hypothetical judgments. But we have already gone through this exercise before, when introducing proof terms! We therefore forego a general analysis of how to disambiguate hypothetical judgments and just work with proof terms.

The localized version of this judgment has the form

$$\underbrace{x_1{:}A_1, \ldots, x_n{:}A_n}_{\Gamma} \vdash M : C$$

where we abbreviate the collection of assumptions by Γ. We often refer to Γ as the *context* for the judgment $M : C$. Following general convention and their use in the proof term M, we now use variables x, y, z for labels of hypotheses. The turnstile '$\vdash$' is the universal symbol indicating a hypothetical judgment, with the hypotheses on the left and the conclusion on the right. Since the goal is to have an unambiguous representation of proofs, we require all variables $x_1, \ldots, x_n$ to be distinct. Of course, some of the propositions $A_1, \ldots, A_n$ may be the same, as they would be in the proof of $A \supset A \supset A$. We write '$\cdot$' for an empty collection of hypotheses when we want to emphasize that there are no assumptions.

The definition of the notion of hypothetical judgment was as an incomplete proof. This means we can always substitute an actual proof for the missing one. That is, any time we use the turnstile ($\vdash$) symbol, we mean that it must satisfy the following principle, now written in localized form.

> **Substitution, v.1**: If $\Gamma_1, x{:}A, \Gamma_3 \vdash N : C$ and $\Gamma_2 \vdash M : A$ then $\Gamma_1, \Gamma_2, \Gamma_3 \vdash [M/x]N : C$.

A second important principle which is implicit in the two-dimensional notation for hypothetical judgments is that hypotheses need not be used. For example, in the proof

$$\dfrac{\dfrac{\dfrac{}{x{:}A, y{:}B \vdash x : A}\,x}{x{:}A \vdash \lambda y.\, x : B \supset A}\,{\supset}I^y}{\cdot \vdash (\lambda x.\, \lambda y.\, x) : A \supset B \supset A}\,{\supset}I^x$$

we find the judgment $x{:}A, y{:}B \vdash x : A$ where y is not used. In two-dimensional notation, this would just be written as

$$\dfrac{}{x : A}\,x$$

which would just be $x{:}A \vdash x : A$ since there is no natural place to display $y{:}B$. Moreover, it might be used later in the context of a larger proof where other assumptions become available, which would also not be shown.

What emerges is a general principle of *weakening*.

> **Weakening**: If $\Gamma_1, \Gamma_2 \vdash N : C$ then $\Gamma_1, x{:}A, \Gamma_2 \vdash N : C$.

It is implicit here that x does not already occur in Γ_1 and Γ_2, which is necessary so that the new context $\Gamma_1, x{:}A, \Gamma_2$ is well-formed.

We also have the principle of *contraction*, whose justification can be seen by going back to the original definition of hypothetical judgments. Does the hypothetical proof

$$\dfrac{\dfrac{A \wedge B\ \mathit{true}}{B\ \mathit{true}}\,{\wedge}E_R \quad \dfrac{A \wedge B\ \mathit{true}}{A\ \mathit{true}}\,{\wedge}E_L}{B \wedge A\ \mathit{true}}\,{\wedge}I$$

correspond to $A \wedge B$ *true* $\vdash B \wedge A$ *true* or $A \wedge B$ *true*, $A \wedge B$ *true* $\vdash B \wedge A$ *true*? Depending on the situation in which we use this, both are possible. For example, during the proof of $A \wedge B \supset B \wedge A$ we would be taking the first interpretation, while the second might come out of a proof of $A \wedge B \supset A \wedge B \supset B \wedge A$.

In general, we can always contract two assumptions by identifying the variables that label them: we simply replace any use by either x or y by z.

> **Contraction**: If $\Gamma_1, x{:}A, \Gamma_2, y{:}A, \Gamma_3 \vdash N : C$ then $\Gamma_1, z{:}A, \Gamma_2, \Gamma_3 \vdash [z/x, z/y]N : C$.

Again, it is implicit that z be chosen distinct from the variables in $\Gamma_1, \Gamma_2, \Gamma_3$.

Finally, we may note that the order of the hypotheses in Γ is irrelevant.

> **Exchange**: If $\Gamma_1, x{:}A, y{:}B, \Gamma_2 \vdash N : C$ then $\Gamma_1, y{:}B, x{:}A, \Gamma_2 \vdash N : C$.

We will have occasion to consider systems where some of the principles of weakening, contraction, or exchange are violated. For example, in *linear logic*, weakening and contraction are restricted to specific circumstances, in *ordered logic* exchange is also repudiated. Also, in *dependent type theory* we have dependencies between assumptions so that some cannot be exchanged, even if weakening and contraction do hold.

The one invariant, however, is the substitution principle since we view is as the defining property of hypothetical judgments.However, even the substitution principle has different forms, depending on how we employ it. Consider the local reduction of $(\lambda x{:}A.\, N)\, M \Longrightarrow_R [M/x]N$. We would like to demonstrate that if the left-hand side is a valid proof term, then the right-hand side is as well. This property is called *subject reduction*. It verifies that the local reduction, when written on proof terms, really is a proper local reduction when considered as an operation on proofs. By inspection of the rules (which we will develop below), we see that if $\Gamma \vdash (\lambda x{:}A.\, N)\, M : C$ then $\Gamma, x{:}A \vdash N : C$ and $\Gamma \vdash M : A$. Now the assumptions of the substitution principle above do not quite apply, so it is convenient to have an alternative formulation that matches this particular use.

> **Substitution, v.2**: If $\Gamma_1, x{:}A, \Gamma_2 \vdash N : C$ and $\Gamma_1 \vdash M : A$ then $\Gamma_1, \Gamma_2 \vdash [M/x]N : C$.

This now matches the property we need for $\Gamma_2 = (\cdot)$. Under the right circumstances, the two versions of the substitution principle coincide (see Exercise 5).

7 Natural Deduction in Localized Form

We now revisit the rules for natural deduction, using the local form of hypothetical judgments, thereby making the contexts of each judgment explicit. We read these rules from the conclusion to the premises. For example (eliding proof terms), in order to prove $\Gamma \vdash A \wedge B$ *true* we have to prove $\Gamma \vdash A$ and $\Gamma \vdash B$, making all current assumptions Γ available for both subproofs. The rules in this form are summarized in Figure 4. There is an alternative possibility, reading the rules from premises to conclusion which is pursued in Exercise 6.

The rules $\supset I$ and $\vee E$ that introduce new hypotheses, are now usually no longer labeled, because we track the assumption and its name locally, in the context. We have to be careful to maintain the assumption that all variables in a context are distinct. For example, would we consider $\vdash \lambda x{:}A.\, \lambda x{:}B.\, x : A \supset B \supset B$ to be a correct judgment? It appears we might be stuck at the point

$$x{:}A \vdash \lambda x{:}B.\, x : B \supset B$$

but we are not, because we can always silently rename bound variables. In this case, this is indistinguishable from

$$x{:}A \vdash \lambda y{:}B.\, y : B \supset B$$

which is now easily checked.

We also see that we acquired one new rule when writing the judgment in its localized form which makes the use of a hypothesis explicit:

$$\frac{}{\Gamma_1, x{:}A, \Gamma_2 \vdash x : A}\ \mathsf{hyp}$$

which could also be written as

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash x : A}\ \mathsf{hyp}$$

This rule is not connected to any particular logical connective, but is derived from the nature of the hypothetical judgment. We often refer to such rules as *judgmental rules* because they are concerned with judgments rather than any specific propositions.

$$\frac{}{\Gamma_1, x{:}A, \Gamma_2 \vdash x : A}\ \mathsf{hyp}$$

Constructors

Destructors

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N\rangle : A \wedge B}\ \wedge I$$

$$\frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \pi_1\, M : A}\ \wedge E_L$$

$$\frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \pi_2\, M : B}\ \wedge E_R$$

$$\frac{}{\Gamma \vdash \langle\,\rangle : \top}\ \top I$$

no destructor for $\top$

$$\frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.\, M : A \supset B}\ \supset I$$

$$\frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B}\ \supset E$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathbf{inl}^B\, M : A \vee B}\ \vee I_L$$

$$\frac{\Gamma \vdash N : B}{\Gamma \vdash \mathbf{inr}^A\, N : A \vee B}\ \vee I_R$$

$$\frac{\Gamma \vdash M : A \vee B \quad \Gamma, x{:}A \vdash N : C \qquad \Gamma, y{:}B \vdash O : C}{\Gamma \vdash \mathbf{case}\, M\ \mathbf{of}\ \mathbf{inl}\, x \Rightarrow N \mid \mathbf{inr}\, y \Rightarrow O : C}\ \vee E$$

no constructor for $\bot$

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathbf{abort}^C\, M : C}\ \bot E$$

Figure 4: Proof term assignment for natural deduction with contexts

8 Subject Reduction

We would like to prove that local reduction, as expressed on proof terms, preserves types. In some sense the idea is already contained in the display of local reduction on proofs themselves, since the judgment A true remains unchanged. Now we reexamine this in a more formal setting, where proof terms are explicit and hypotheses are listed explicitly as part of hypothetical judgments.

The reduction for implication, $(\lambda x{:}A.\, N)\, N \Longrightarrow_R [M/x]N$, suggests that we will first need a property of substitution, already stated in Section 6. We take for granted the property of weakening, which just adds an unused hypothesis but otherwise leaves the proof unchanged. Before we undertake the proof, we should formally define the substitution as an operation on terms. For that we define the notion of a *free variable* in a term. We say x is free in M if x occurs in M outside the scope of a binder for x. Note that if $\Gamma \vdash M : A$ then any free variable in M must be declared in Γ. The point of the terminology is to be able to define certain operations without requiring terms to be a priori well-typed or carrying around explicit contexts at all times.

$$
\begin{array}{lcll}
[M/x]x & = & M & \\
[M/x]y & = & y & \text{for } x \neq y \\[1ex]
[M/x](\langle N_1, N_2\rangle) & = & ([M/x]N_1)\,([M/x]N_2) & \\
[M/x](\pi_1\, N) & = & \pi_1\,([M/x]N) & \\
[M/x](\pi_2\, N) & = & \pi_2\,([M/x]N) & \\[2ex]
[M/x](\langle\,\rangle) & = & \langle\,\rangle & \\[2ex]
[M/x](\lambda y{:}A.\, N) & = & \lambda y{:}A.\, [M/x]N & \text{provided } x \neq y \text{ and } y \text{ not free in } M \\
[M/x](N_1\, N_2) & = & ([M/x]N_1)\,([M/x]N_2) & \\[2ex]
[M/x](\mathbf{inl}^B\, N) & = & \mathbf{inl}^B\,([M/x]N) & \\
[M/x](\mathbf{inr}^A\, N) & = & \mathbf{inr}^A\,([M/x]N) & \\
\multicolumn{4}{l}{[M/x](\mathbf{case}\, N_1\, \mathbf{of}\, \mathbf{inl}\, y \Rightarrow N_2 \mid \mathbf{inr}\, z \Rightarrow N_3)} \\
 & = & \mathbf{case}\,([M/x]N_1)\, \mathbf{of}\, \mathbf{inl}\, y \Rightarrow ([M/x]N_2) \mid \mathbf{inr}\, z \Rightarrow ([M/x]N_3) & \\
 & & & \text{provided } x \neq y,\, x \neq z \\
 & & & \text{and } y \text{ and } z \text{ not free in } M \\[2ex]
[M/x](\mathbf{abort}^C\, N) & = & \mathbf{abort}^C\,([M/x]N) &
\end{array}
$$

We call the definition of substitution *compositional* because it pushes the substitution through to all subterms and recomposes the results with the same constructor or destructor. This is a key observation, because it tells us that the definition is modular: if we add a new logical connective or operator to the language, substitution is always extended compositionally. This should have been clear from the beginning, given the intuitive picture of replacing an unproven assumption in a proof with an actualy proof.

We also say that the definition of $[M/x]N$ is by induction on the structure of N, because on the right-hand side the substitution is always applied to subterms. Since we also have a clause for every possible case, this guarantees that substitution is always defined. To see this we just have to observe that the provisos on the clauses that bind variables (λ and **case**) can always be satisfied by renaming, which we do silently.

Theorem 1 (Substitution) *If* $\Gamma \vdash M : A$ *and* $\Gamma, x{:}A, \Gamma' \vdash N : C$ *then* $\Gamma, \Gamma' \vdash [M/x]N : C$

Proof: By induction on N. This is suggested by the fact that the substitution operation itself is defined by induction on N. We show a few representative cases.

Case: $N = x$.

$\Gamma, x{:}A, \Gamma' \vdash x : C$	Given
$A = C$	By inversion, since var. decls. are unique
$\Gamma \vdash M : A$	Given
$\Gamma, \Gamma' \vdash M : A$	By weakening
$\Gamma, \Gamma' \vdash [M/x]x : A$	Since $[M/x]x = M$
$\Gamma, \Gamma' \vdash [M/x]x : C$	Since $A = C$

Case: $N = y$ for $y \neq x$.

$\Gamma \vdash M : A$	Given
$\Gamma, x{:}A, \Gamma' \vdash y : C$	Given
$y{:}C \in (\Gamma, \Gamma')$	By inversion and $y \neq x$
$\Gamma, \Gamma' \vdash y : C$	By rule hyp

Case: $N = N_1\, N_2$.

$\Gamma, x{:}A, \Gamma' \vdash N_1\, N_2 : C$	Given
$\Gamma, x{:}A, \Gamma' \vdash N_1 : C_2 \supset C$ and	
$\Gamma, x{:}A, \Gamma' \vdash N_2 : C_2$	By inversion

$\Gamma, \Gamma' \vdash [M/x]N_1 : C_2 \supset C$	By i.h. on N_1
$\Gamma, \Gamma' \vdash [M/x]N_2 : C_2$	By i.h. on N_2
$\Gamma, \Gamma' \vdash ([M/x]N_1)\,([M/x]N_2) : C$	By rule $\supset E$
$\Gamma, \Gamma' \vdash [M/x](N_1\,N_2) : C$	By defn. of $[M/x](N_1\,N_2)$

Case: $N = (\lambda y.\, N_1)$.

$\Gamma, x{:}A, \Gamma' \vdash (\lambda y.\, N_1) : C$	Given
$\Gamma, x{:}A, \Gamma', y{:}C_2 \vdash N_1 : C_1$ and	
$C = C_2 \supset C_1$ for some C_1, C_2	By inversion
$\Gamma, \Gamma', y{:}C_2 \vdash [M/x]N_1 : C_1$	by i.h. on N_1
$\Gamma, \Gamma' \vdash \lambda y{:}C_2.\, [M/x]N_1 : C_1$	By rule $\supset I$
$\Gamma, \Gamma' \vdash [M/x](\lambda y{:}C_2.\, N_1) : C_1$	By definition of $[M/x](\lambda y{:}C_2.\, N_1)$

In this last case worth verifying that the provisos in the definition of $[M/x](\lambda y{:}C_2.\, N_1)$ are satisfied to see that it is equal to $\lambda y{:}C_2.\, [M/x]N_1$. When we applied inversion, we formed $\Gamma, x{:}A, \Gamma', y{:}C_2$, possibly applying some implicit renaming to make sure y was different from x and any variable in Γ and Γ'. Since the free variables of M are contained in Γ, the proviso must be satisfied.

□

Now we return to subject reduction proper. We only prove it for a top-level (local) reduction. It should be clear by a simple additional argument that if the reduction takes place anywhere inside a term, the type of the overall term also is preserved.

Theorem 2 (Subject Reduction) *If* $\Gamma \vdash M : A$ *and* $M \Longrightarrow_R M'$ *then* $\Gamma \vdash M' : A$.

Proof: By cases on $M \Longrightarrow_R M'$, using inversion on the typing derivation in each case. We show a few cases.

Case: $\pi_1\langle M_1, M_2\rangle \Longrightarrow_R M_1$.

$\Gamma \vdash \pi_1\langle M_1, M_2\rangle : A$	Given
$\Gamma \vdash \langle M_1, M_2\rangle : A \wedge B$ for some B	By inversion
$\Gamma \vdash M_1 : A$ and $\Gamma \vdash M_2 : B$	By inversion

Case: $\pi_2\langle M_1, M_2\rangle \Longrightarrow_R M_2$. Symmetric to the previous case.

Case: $(\lambda x{:}A_2.\, M_1)\, M_2 \Longrightarrow_R [M_2/x]M_1$.

$\Gamma \vdash (\lambda x{:}A_2.\, M_1)\, M_2 : A$	Given
$\Gamma \vdash \lambda x{:}A_2.\, M_1 : A_2 \supset A$ and	
$\Gamma \vdash M_2 : A_2$	By inversion
$\Gamma, x{:}A_2 \vdash M_1 : A$	By inversion
$\Gamma \vdash [M_2/x]M_1 : A$	By substitution (Theorem 1)

$\square$

An analogous theorem holds for local expansion; see Exercise 11.

The material so far is rather abstract, in that it does not commit to whether we are primarily interested in logical reasoning or programming. If we want to take this further, the two views which are perfectly unified at this point, will start to drift apart. From the logical point of view, we need to investigate how to prove global soundness and completeness as introduced at the end of Lecture 1.

From a programming language perspective, we are interested in defining an operational semantics which applies local reductions in a precisely specified manner to subterms. With respect to such a semantics we can prove *progress* (either we have a value, or we can perform a reduction step) on well-typed program and *totality* (all well-typed programs have a value). Some progress has been made to understand the choices that still remain in the definition of the operational semantics from a logical point of view, but in the end the interpretation of proofs as programs remains as an important guide to the design of the type structure of a language, but is not the final word. We will often return to these questions of the operational semantics, because much of our investigation of various intuitionistic modal logics and their computational interpretations will be at this interface.

Exercises

Exercise 1 *Write proof terms for each direction of the interaction laws (L13), (L19), and (L20). Abuse the type inference engine of your favorite functional programming language (ML, Haskell, . . .) to check the correctness of your proof terms.*

Exercise 2 *Two types A and B are* isomorphic *if there are functions $M : A \supset B$ and $N : B \supset A$ such that their compositions $\lambda x{:}A.\, N\ (M\ x) : A \supset A$ and $\lambda y{:}B.\, M\ (N\ y) : B \supset B$ are both identity functions.*

For example, $A \wedge B$ and $B \wedge A$ are isomorphic, because

$$\lambda y{:}A \wedge B.\, \langle \pi_2\, y, \pi_1\, y\rangle : (A \wedge B) \supset (B \wedge A)$$

and

$$\lambda z{:}B \wedge A.\, \langle \pi_2\, y, \pi_1\, y\rangle : (B \wedge A) \supset (A \wedge B)$$

and

$$\begin{array}{l} \lambda x.\, (\lambda y.\, \langle \pi_2\, y, \pi_1\, y\rangle)\ ((\lambda z.\, \langle \pi_2\, z, \pi_1\, z\rangle)\ x) \\ \equiv \lambda x.\, (\lambda y.\, \langle \pi_2\, y, \pi_1\, y\rangle)\ \langle \pi_2\, x, \pi_1\, x\rangle \\ \equiv \lambda x.\, \langle \pi_2\, \langle \pi_2\, x, \pi_1\, x\rangle, \pi_1\, \langle \pi_2\, x, \pi_1\, x\rangle\rangle \\ \equiv \lambda x.\, \langle \pi_1\, x, \pi_1\, \langle \pi_2\, x, \pi_1\, x\rangle\rangle \\ \equiv \lambda x.\, \langle \pi_1\, x, \pi_2\, x\rangle \\ \equiv \lambda x.\, x \end{array}$$

and symmetrically for the other direction. Here we used the congruence ($\equiv$) generated by reductions and expansions, used in either direction and applied to arbitrary subterms.

Check which among, (L13), (L19), and (L20) are type isomorphisms, using only local reductions and expansions, in either direction, on any subterm, in order to show that the compositions are the identity.

For those that fail, is it the case that the types are simply not isomorphic, or does the failure suggest additional equivalences between proof terms that might be justified?

Exercise 3 *Proceed as in Exercises 1 and 2, but for laws (L4) and (L6).*

Exercise 4 *We could try to add a second rule for uses of falsehood, namely*

$$\frac{\bot\!\downarrow}{C\!\downarrow}\ \bot E$$

Give arguments for an against such a rule, using examples and counterexamples.

Exercise 5 *Carefully state the relationship between the two versions of the substitution principle and prove that it is satisfied.*

Exercise 6 *We can write a system with localized hypotheses where each rule is naturally read from the premises to the conclusion and only records hypotheses that are actually used. For example, we can rewrite* $\wedge I$ *and* hyp *as*

$$\frac{\Gamma_1 \vdash M : A \quad \Gamma_2 \vdash N : B}{\Gamma_1, \Gamma_2 \vdash \langle M, N \rangle : A \wedge B} \wedge I \qquad\qquad \frac{}{x{:}A \vdash x : A}\,\mathsf{hyp}$$

(i) Complete the system of rules.

(ii) State the precise relationship with the system in Figure 4.

(iii) Prove that relationship.

Exercise 7 *Show the cases concerning pairs in the proof of the substitution property (Theorem 1).*

Exercise 8 *Show the cases concerning disjunction in the proof of the substitution property (Theorem 1).*

Exercise 9 *As pragmatists, we can define a new connective* $A \otimes B$ *by the following elimination rule.*

$$\frac{A \otimes B\ \mathit{true} \qquad \begin{array}{c} \overline{A\ \mathit{true}}^{\,u} \quad \overline{B\ \mathit{true}}^{\,w} \\ \vdots \\ C\ \mathit{true} \end{array}}{C\ \mathit{true}} \otimes E^{u,w}$$

(i) Write down corresponding introduction rules.

(ii) Show local reductions to verify local soundness.

(iii) Show local expansion to verify local completeness.

(iv) Define verifications and uses of $A \otimes B$.

(v) Devise a notation for proof terms and show versions of the rules with local hypotheses and proof terms.

(vi) Reiterate local reduction and expansions on proof terms.

(vii) Extend the definition of substitution to include the new terms.

(viii) Extend the proof of the substitution property to encompass the new terms.

(ix) Show the new cases in subject reduction.

(x) Can we express $A \otimes B$ as an equivalent proposition in variables A and B using only conjunction, implication, disjunction, truth, and falsehood? If so, prove the logical equivalence. If not, explain why not.

(xi) Can you relate the proof terms for $A \otimes B$ to constructs in existing functional languages such as Haskell or ML?

Exercise 10 *As verificationists, we can define a new connective $A \wedge_L B$ with the following introduction rule.*

$$\dfrac{A\ \mathit{true} \qquad \begin{array}{c}\overline{A\ \mathit{true}}^{\,u} \\ \vdots \\ B\ \mathit{true}\end{array}}{A \wedge_L B\ \mathit{true}}\ \wedge_L I^u$$

(i) Write down the corresponding elimination rules.

(ii)–(xi) as in Exercise 9, where in part (x), try to define $A \wedge_L B$ in terms of the existing connectives.

Exercise 11 *Prove* subject expansion*: If $\Gamma \vdash M : A$ and $M : A \Longrightarrow_E M'$ then $\Gamma \vdash M' : A$.*

References

[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.