Chapter 3

Proofs as Programs

In this chapter we investigate a computational interpretation of constructive proofs and relate it to functional programming. On the propositional fragment of logic this is referred to as 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*.

3.1 Propositions as Types

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

$M : A$ 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*.

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

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

$$\frac{M : A \wedge B}{\mathbf{fst}\, M : A}\wedge E_L \qquad \frac{M : A \wedge B}{\mathbf{snd}\, M : B}\wedge E_R$$

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 $\mathbf{1}$ 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.

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

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*:

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

If we annotate the deduction with proof terms, we obtain

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

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 $M(N)$.

$$\frac{M : A \supset B \qquad N : A}{M\,N : B}\supset\! E$$

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*.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{A \wedge B\ \mathit{true}}\,u}{B\ \mathit{true}}\,\wedge E_R \qquad \cfrac{\cfrac{}{A \wedge B\ \mathit{true}}\,u}{A\ \mathit{true}}\,\wedge E_L}{B \wedge A\ \mathit{true}}\,\wedge I}{(A \wedge B) \supset (B \wedge A)\ \mathit{true}}\,\supset\! I^u$$

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 $\langle N, M\rangle$.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{u : A \wedge B}\,u}{\mathbf{snd}\,u : B}\,\wedge E_R \qquad \cfrac{\cfrac{}{u : A \wedge B}\,u}{\mathbf{fst}\,u : A}\,\wedge E_L}{\langle \mathbf{snd}\,u, \mathbf{fst}\,u\rangle : B \wedge A}\,\wedge I}{(\lambda u.\ \langle \mathbf{snd}\,u, \mathbf{fst}\,u\rangle) : (A \wedge B) \supset (B \wedge A)}\,\supset\! I^u$$

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.

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

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.

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

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 ($\perp$). We can therefore view it as the empty type $\mathbf{0}$. The corresponding elimination rule allows a term of $\perp$ 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.

$$\dfrac{M : \perp}{\mathbf{abort}^C\, M : C}\perp E$$

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 left-to-right direction of (L11)

(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:

$$\lambda u.\ \langle (\lambda w.\ \mathbf{fst}\,(u\, w)), (\lambda v.\ \mathbf{snd}\,(u\, v))\rangle$$

The following deduction provides the evidence:

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

$$
\dfrac{\langle(\lambda w.\ \mathbf{fst}\,(u\,w)), (\lambda v.\ \mathbf{snd}\,(u\,v))\rangle : (A \supset B) \wedge (A \supset C)}{\lambda u.\ \langle(\lambda w.\ \mathbf{fst}\,(u\,w)), (\lambda v.\ \mathbf{snd}\,(u\,v))\rangle : (A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))}{\supset}I^u
$$

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. We introduce such data types in Section **??**, following the same method we have used in the development of logic.

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.

We will prove these in Section **??**.

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* $M \Longrightarrow M'$, read "M *reduces to* M'". A computation then proceeds by a sequence of reductions $M \Longrightarrow M_1 \Longrightarrow M_2 \ldots$, according to a fixed strategy, until we reach a value which is the result of the computation. In this section we cover reduction; we return to reduction strategies in Section **??**.

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.

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

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

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

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.

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

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.

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

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

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

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

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

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.

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

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 Summary of Proof Terms

Judgments.

$M : A$	M is a proof term for proposition A
$M \Longrightarrow M'$	M reduces to M'

Proof Term Assignment.

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

Reductions.

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

Bibliography

[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.