Chapter 4

The Curry-Howard isomorphism

Having discussed intuitionistic logic in Chapter 2 and typed lambda-calculus in Chapter 3, we now concentrate on the relationship between the two—the Curry-Howard isomorphism. We have already discovered this amazing analogy between logic and computation in Chapter 3, where we readily accepted lambda terms as an appropriate notation for proofs. Indeed, the Brouwer-Heyting-Kolmogorov interpretation, in particular the functional understanding of constructive implication, strongly supports this choice.

In this chapter we have a closer look at the correspondence between proofs and terms. We soon discover that, depending on the choice of a specific proof formalism, it can be more or less adequate to call it an "isomorphism". In most cases, lambda-terms can be seen as a refinement of proofs rather than an isomorphic image. On the other hand, we also discover that the analogy goes further than the equivalence between non-empty types and provable implicational formulas. It goes further "in width", because it extends to a correspondence between various logical connectives and data types. But it also goes further "in depth". There is a fundamental relationship between term reduction (computation) and proof normalization. This aspect of the formulas-as-types correspondence is perhaps the most important one.

4.1. Proofs and terms

As we have discovered in Chapter 3, the type assignment rules of $\lambda_{\to}$ can be seen as annotated versions of the natural deduction system $\mathrm{NJ}(\to)$. The following fact is an immediate consequence of this similarity.

4.1.1. Proposition (Curry-Howard isomorphism).

(i) *If* $\Gamma \vdash M : \varphi$ *in* $\lambda_{\to}$ *then* $\mathrm{rg}(\Gamma) \vdash \varphi$ *in* $\mathrm{IPC}(\to)$.

(ii) *If* $\Delta \vdash \varphi$ *in* $\mathrm{IPC}(\to)$ *then* $\Gamma \vdash M : \varphi$ *in* $\lambda_{\to}$, *for some* M *and some* Γ *with* $\mathrm{rg}(\Gamma) = \Delta$.

In particular an implicational formula is an intuitionistic theorem if and only if it is an inhabited type.

PROOF. Easy induction with respect to derivations. In part (ii) it is convenient to choose $\Gamma = \{(x_\varphi : \varphi) \mid \varphi \in \Delta\}$, where x_φ are distinct variables. □

The correspondence between logic and lambda-calculus makes it possible to translate results concerning one of these systems into results about the other. For instance we immediately derive that

- There is no combinator of type $((p \to q) \to p) \to p$;
- Formula $((((p \to q) \to p) \to p) \to q) \to q$ is an intuitionistic tautology,

because we know that Peirce's law (Example 2.1.4(i)) is not intuitionistically valid, and because we can construct the term

$$\lambda x^{(((p\to q)\to p)\to p)\to q}.\, x(\lambda y^{(p\to q)\to p}.\, y(\lambda z^{p}.\, x(\lambda u^{(p\to q)\to p}\, z))).$$

APPLICATIONS OF NORMALIZATION. Many properties regarding unprovability can be difficult to obtain directly. In Chapter 2 we used semantics for this purpose. The Curry-Howard isomorphism suggests an alternative approach to such problems, which is based on normalization. Here is an example.

4.1.2. PROPOSITION. *The implicational logic* IPC($\to$) *is consistent, i.e. there are underivable judgements.*

PROOF. Assume that $\vdash p$, where p is a propositional variable. Then $\vdash M : p$ for some closed term M, which by Theorem 3.5.1 can be assumed normal. So $M = \lambda y_1 \ldots y_m.\, x M_1 \ldots M_n$, by Remark 1.3.7, and because of the type of M, we must have $m = 0$. But then x is free in M, a contradiction. □

The reader will perhaps find this result entirely obvious. Indeed, the consistency of intuitionistic propositional calculus is an immediate consequence of soundness. However, other logical calculi and theories are not always so simple. A semantical consistency proof (construction of a model) may then require concepts and tools more sophisticated than normalization. Proposition 4.1.2 demonstrates an important proof method on a simple example.

In the next section (and also in Section 8.8) we will see another application of the Curry-Howard isomorphism and normalization. Questions concerning decidability and complexity of a logical system can easier be resolved if one can identify provability with the existence of normal inhabitants. Other applications of proof normalization are various conservativity results (cf. Remark 7.3.7), program extraction procedures (like in the proof of Theorem 12.7.11), and syntactic proofs of various metamathematical properties (disjunction property, existence property, independence of connectives, etc.)

4.2. Type inhabitation

We now prove that the inhabitation problem for the simply typed lambda-calculus is PSPACE-complete. In particular it is decidable. An immediate consequence is that provability in IPC($\to$) is also decidable and PSPACE-complete. Compare this to the semantic proof obtained in Chapter 2 as a consequence of Theorem 2.4.12.

First we prove that our problem is in PSPACE. It follows from normalization that an inhabited type must have an inhabitant in normal form. Thus it suffices to consider terms in normal form. In fact, it is convenient to make a further restriction.

4.2.1. DEFINITION. We define the notion of a Church-style term in *η-long normal form* (or just *long normal form*) as follows:

- If x is a variable of type $\tau_1 \to \cdots \to \tau_n \to p$, and $M_1^{\tau_1}, \ldots, M_n^{\tau_n}$ are in η-long normal form, then $(xM_1 \ldots M_n)^p$ is in η-long normal form.
- If M^σ is in η-long normal form then so is $(\lambda x{:}\tau.\, M)^{\tau\to\sigma}$.

Equivalently, a long normal form is a term of shape $\lambda x_1^{\tau_1} \ldots x_n^{\tau_n}.\, y\vec{N}$, where $\vec{N}$ are normal forms. Informally speaking, a long normal form is a normal form where all function variables are "fully applied" (all arguments are provided).

4.2.2. LEMMA. *For every Church-style term M^σ in β-normal form, there exists a term L^σ in η-long normal form, such that $L^\sigma \twoheadrightarrow_\eta M^\sigma$.*

PROOF. Easy induction with the help of Remark 1.3.7. □

4.2.3. LEMMA. *There is a polynomial space algorithm to determine whether a given type τ is inhabited in a given environment Γ.*

PROOF. If a type is inhabited then, by Proposition 4.1.1 and Lemma 4.2.2, it is inhabited by a long normal form. To determine whether there exists a long normal form M, satisfying $\Gamma \vdash M : \tau$, we proceed as follows:

- If $\tau = \tau_1 \to \tau_2$, then M must be an abstraction, $M = \lambda x{:}\tau_1.\, M'$. Thus, we look for an M' satisfying $\Gamma, x : \tau_1 \vdash M' : \tau_2$.
- If τ is a type variable, then M is an application of a variable to a sequence of terms. We nondeterministically choose a variable z, declared in Γ to be of type $\tau_1 \to \cdots \to \tau_n \to \tau$. (If there is no such variable, we reject.) If $n = 0$ then we accept. If $n > 0$, we recursively answer all the questions whether τ_i are inhabited in Γ.

This nondeterministic recursive (or alternating) procedure is repeated as long as there are new questions of the form $\Gamma \vdash ? : \tau$. We can assume that in a successful run of the procedure no such question is raised again (otherwise we can consider a shorter run). Note also that if there are two variables in Γ, say x and y, declared to be of the same type σ, then each term M can be replaced with $M[y := x]$ with no change of type. This means that a type τ is inhabited in Γ iff it is inhabited in $\Gamma - \{y : \sigma\}$, and it suffices to consider contexts with all declared types being different. At each step of our procedure, the environment Γ either stays the same or it expands. And it can remain unchanged for at most as many steps as there are different possible formulas to the right of $\vdash$. Thus the depth of recursion does not exceed the squared number of subformulas of types in Γ, τ, where $\Gamma \vdash ? : \tau$ is the initially posed question. It follows that the space needed to implement our algorithm is polynomial: The stack of procedure activations is of polynomial height, and each record on the stack is of polynomial size. □

The reader who has noticed the game flavour of the algorithm used in the proof will see more of that in Section 4.6. Now we have to show PSPACE-hardness. We define a reduction from the satisfiability problem for classical second-order propositional formulas (or "quantified Boolean formulas"), see Section A.5.

Assume that a quantified Boolean formula Φ is given. Without loss of generality we may assume that the negation symbol $\neg$ does not occur in Φ, except in the context $\neg p$, where p is a propositional variable.

Assume that all bound variables of Φ are different and that no variable occurs both free and bound. For each propositional variable p, occurring in Φ (free or bound), let α_p and $\alpha_{\neg p}$ be fresh type variables. Also, for each subformula φ of Φ, let α_φ be a fresh type variable. We construct an environment Γ_Φ from the following types:

- $(\alpha_p \to \alpha_\psi) \to (\alpha_{\neg p} \to \alpha_\psi) \to \alpha_{\forall p\, \psi}$, for each subformula $\forall p\, \psi$;
- $(\alpha_p \to \alpha_\psi) \to \alpha_{\exists p\, \psi}$, $(\alpha_{\neg p} \to \alpha_\psi) \to \alpha_{\exists p\, \psi}$, for each subformula $\exists p\, \psi$;
- $\alpha_\psi \to \alpha_\vartheta \to \alpha_{\psi \wedge \vartheta}$, for each subformula $\psi \wedge \vartheta$;
- $\alpha_\psi \to \alpha_{\psi \vee \vartheta}$ and $\alpha_\vartheta \to \alpha_{\psi \vee \vartheta}$, for each subformula $\psi \vee \vartheta$.

If v is a zero-one valuation of propositional variables, then Γ_v is Γ_Φ extended with the type variables

- α_p, when $v(p) = 1$;
- $\alpha_{\neg p}$, when $v(p) = 0$.

The following lemma is proved by a routine induction with respect to the length of formulas. Details are left to the reader.

4.2.4. LEMMA. *For every subformula φ of Φ, and every valuation v, defined on the free variables of φ, the type α_φ is inhabited in Γ_v iff $[\![\varphi]\!]_v = 1$. Thus, to verify if Φ is a tautology, one checks if α_Φ is inhabited in Γ_Φ.*

Observe that the number of subformulas of Φ does not exceed the length of Φ. Thus the reduction from Φ to Γ_Φ and α_Φ can be performed in logarithmic space (provided we represent a variable α_ψ using a binary number, rather than ψ itself). This implies the PSPACE-hardness, which together with Lemma 4.2.3 gives the main result of this section.

4.2.5. THEOREM. *The inhabitation problem for simply typed lambda-calculus is complete for polynomial space.* □

An immediate consequence of the above (and Theorem 2.6.2) is that the intuitionistic propositional logic (with all the connectives) is also PSPACE-hard. In Chapter 7 we will define a polynomial space algorithm for arbitrary formulas, thus proving PSPACE-completeness of the full system (Exercise 7.13).

4.3. Not an exact isomorphism

We have just demonstrated that the formulas-as-types analogy can indeed be useful. However, the reader may find Proposition 4.1.1 a little unsatisfactory. If we talk about an "isomorphism" then perhaps the statement of the proposition should have the form of an equivalence? The concluding sentence is indeed of this form, but it only holds on a fairly high level: We must abstract from the proofs and only ask about conclusions. (Or, equivalently, we abstract from terms and only ask which types are non-empty.) To support the idea of an "isomorphism," we would certainly prefer an exact, bijective correspondence between proofs and terms.

Unfortunately, we cannot improve Proposition 4.1.1 in this respect, at least not for free. While it is correct to say that lambda-terms are essentially annotated proofs, the problem is that some proofs can be annotated in more than one way. For instance, the proof

$$\dfrac{\dfrac{p \vdash p}{p \vdash p \to p}}{\vdash p \to p \to p}$$

can be annotated as either $\lambda x^p \lambda y^p\, x$ or $\lambda x^p \lambda y^p\, y$. And the three judgements

$$f : p \to p \to q,\ x : p \vdash fxx : q;$$
$$f : p \to p \to q,\ x : p,\ y : p \vdash fxy : q;$$
$$f : p \to p \to q,\ x : p,\ y : p,\ z : p \vdash fxx : q$$

(and many others) correspond to the same derivation:

$$\dfrac{\dfrac{p \to p \to q, p \vdash p \to p \to q \quad p \to p \to q, p \vdash p}{p \to p \to q, p \vdash p \to q} \qquad p \to p \to q, p \vdash p}{p \to p \to q, p \vdash q}$$

As we can thus see, the difference between terms and proofs is that the former carry more information than the latter. The reason is that in logic, the primary isssue is usually to determine provability of a formula, and in this respect, certain issues are irrelevant. It does not matter whether we use the same assumption twice or if we use two different assumptions about the same formula. In fact, many of us would have objections against the very idea of "two different assumptions about φ." Indeed, in classical or intuitionistic logic, once φ becomes part of our knowledge, it can be asserted as many times as we want. We don't make any distinction between repeated statements of φ, it is just repeating the same information. On the contrary, in lambda-calculus we can have many variables of the same type φ, and this corresponds to making a difference between various assumptions about the same formula.

But when studying proofs as such (rather than mere provability) it may actually be useful to maintain exact bookkeeping of assumptions. A further discovery is that, from a purely logical point of view, it can make a lot of sense to see whether an assumption is used or not in an argument, or how many times it has to be used. In *resource-conscious* logics, a major example of which is *linear logic*, these issues play a fundamental role. If we realize all this, we conclude that our natural deduction proofs are simply too schematic and perhaps a more informative proof system may turn our Curry-Howard correspondence into a real isomorphism.

In fact, our formulation of natural deduction is not very convenient for this specific purpose. It can be more informative to write natural deduction proofs in the "traditional" way. We use this occasion to explain it informally. In this style, quite common in the literature on proof theory, one does not associate a set of assumptions (the environment) to every node of a derivation. Instead, one writes all the assumptions at the top, and puts in brackets (or crosses out) those assumptions that have been discharged by the implication introduction rule. To indicate which assumptions are eliminated by which proof steps, one can enhance the notation by using labels:

$$\dfrac{\begin{array}{c}[\varphi]^{(i)} \\ \vdots \\ \psi\end{array}}{\varphi \to \psi}\,(i)$$

In general, in an $\to$-introduction step, we may discharge zero, one, or more occurrences of the assumption. A label referring to an arrow-introduction proof step is then attached to all discharged assumptions.

The arrow elimination is represented as usual:

$$\frac{\varphi \to \psi \quad \varphi}{\psi}$$

and a proof is a tree with all assumptions at the top and the conclusion as a root. It is now apparent that the "traditional" natural deduction is more fine-grained than our simplified notation (although often less convenient in use). Indeed, for instance

$$\dfrac{\dfrac{\dfrac{}{p}(1)}{p \to p}(1)}{\vdash p \to p \to p}(2) \qquad \text{and} \qquad \dfrac{\dfrac{\dfrac{}{p}(1)}{p \to p}(2)}{\vdash p \to p \to p}(1)$$

are two different proofs of $p \to p \to p$, corresponding respectively to terms $\lambda x^p \lambda y^p\, y$ and $\lambda x^p \lambda y^p\, x$. However, this additional information is only available about discharged assumptions, and not about the free ones. Indeed, our other example, proving q from $p \to p \to q$ and p, would be written as

$$\dfrac{\dfrac{p \to p \to q \qquad p}{p \to q} \qquad p}{q}$$

which of course still gives no clue about the possible identification of the two p's, until we discharge one or both of them. Thus, the "traditional" natural deduction allows us to claim an isomorphism[1] for closed terms and proofs with no free assumptions (Exercise 4.8), but not for arbitrary proofs and terms. To extend the isomorphism to arbitrary terms we need a proof system with labeled free assumptions. One can design such a proof system, only to discover that the resulting proofs, up to syntactic sugar, are... another representation of lambda-terms.

4.4. Proof normalization

The correspondence between proofs and lambda-terms suggests that reduction of lambda-terms should be related to a meaningful operation on proofs. Indeed, *proof normalization* is a central issue in proof theory and has been studied by logicians independently. No surprise. Once we agree that proofs are as important in logic as the formulas they derive, it becomes a basic question whether two given proofs of the same formula should be considered essentially identical, or whether they are truly different. In particular it is desirable that an arbitrary proof can be identified with a *normal proof* of

[1] To be precise, we also need alpha-conversion on discharge labels.

a certain simple shape. A natural way to identify two proofs is to rewrite one into another by means of a sequence of proof reductions.[2] As an example, consider the following natural deduction proof:

$$\dfrac{\dfrac{p \vdash p}{\vdash p \to p}\,(\to\mathrm{I}) \qquad \dfrac{\dfrac{\dfrac{p \to p,\ q \vdash p \to p}{p \to p \vdash q \to p \to p}\,(\to\mathrm{I})}{\vdash (p \to p) \to q \to p \to p}\,(\to\mathrm{I})}{\vdash q \to p \to p}\,(\to\mathrm{E})$$

This proof demonstrates that $q \to p \to p$ is derivable. However, it does so in a somehow inefficient way. By $\to$-introduction we derive the formula $(p \to p) \to q \to p \to p$. Since we can *prove* the assumption $p \to p$, we apply the $\to$-elimination rule to obtain the conclusion. That is, we introduce the implication only to immediately eliminate it. Such a pair consisting of an introduction step followed by an elimination step (applied to the formula just introduced) is called a *detour*. Here is a template of a detour.

$$\dfrac{\Gamma \vdash \varphi \qquad \dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\to\mathrm{I})}{\Gamma \vdash \psi}\,(\to\mathrm{E})$$

This resembles of course a beta-redex $(\lambda x{:}\varphi.M^{\psi})N^{\varphi}$ which is reduced by replacing every occurrence of x with a copy of N. In a similar way, to get rid of a detour, one replaces every use of the assumption φ in the right-hand side proof by a copy of the left-hand side proof. This operation turns our proof of $q \to p \to p$ into a simplified one:

$$\dfrac{\dfrac{p \vdash p}{\vdash p \to p}\,(\to\mathrm{I})}{\vdash q \to p \to p}\,(\to\mathrm{I})$$

Our example becomes more transparent if we use the "traditional" notation for natural deduction proofs. Our initial proof is then written as

$$\dfrac{\dfrac{[p]^{(3)}}{p \to p}\,(3) \qquad \dfrac{\dfrac{[p \to p]^{(1)}}{q \to p \to p}\,(2)}{(p \to p) \to q \to p \to p}\,(1)}{q \to p \to p}$$

[2] This was first discovered by Gentzen, who designed *sequent calculus* especially for this purpose (see Chapter 7).

and reduces to

$$\dfrac{\dfrac{[p]^{(3)}}{p \to p}\,(3)}{q \to p \to p}\,(2)$$

as a result of replacing the assumption $[p \to p]^{(1)}$ by the proof of $p \to p$. In general, we have the following reduction rule:

$$\dfrac{\begin{array}{cc}(*) & [\psi]^{(i)} \\ \vdots & \vdots \\ \psi & \varphi \end{array}}{\varphi}\;\dfrac{}{\psi \to \varphi}\,(i) \qquad \Longrightarrow \qquad \begin{array}{c}(*) \\ \vdots \\ \psi \\ \vdots \\ \varphi\end{array}$$

which should be read as follows. Replace each occurrence of the assumption $[\psi]^{(i)}$ in the proof of φ by a separate copy of the proof of ψ marked by $(*)$. Now we have an explicit analogy with a substitution $M[x := N]$.

The process of eliminating proof detours is called *proof normalization,* and a proof tree with no detours is said to be in *normal form,* quite like a term without redexes. An important application of the Curry-Howard isomorphism is that every proof can be turned into a normal form, because every typed term is normalizing. By the strong normalization theorem for $\lambda_{\to}$ (Theorem 3.5.5), the order of eliminating detours is also irrelevant.

4.4.1. Proposition. *Each provable judgement of* NJ($\to$) *has a normal proof.*

Proof. Suppose that $\Gamma \vdash \varphi$ is derivable. Then by Proposition 4.1.1(ii) we have $\Delta \vdash M : \varphi$ where $\mathrm{rg}(\Delta) = \Gamma$. By Theorem 3.5.5 we can assume that M is in normal form. We show that $\Gamma \vdash \varphi$ has a normal proof by easy induction with respect to M, using Remark 1.3.7. □

Another form of detour, corresponding to eta-reduction, occurs when an arrow-elimination step is followed by an immediate re-introduction of the same implication. Abusing the formalism slightly, we can represent this situation as

$$\dfrac{\dfrac{\Gamma \vdash \varphi \to \psi \qquad \dfrac{}{\Gamma, \varphi \vdash \varphi}\,(\mathrm{Ax})}{\Gamma, \varphi \vdash \psi}\,(\to\mathrm{E})}{\Gamma \vdash \varphi \to \psi}\,(\mathrm{I}\to)$$

Again, the traditional notation is more illustrative, so we use it to show the eta-rule for proofs:

$$\dfrac{\dfrac{\begin{array}{c}(*)\\ \vdots\\ \varphi \to \psi\end{array} \qquad [\varphi]^{(i)}}{\psi}}{\varphi \to \psi}\,(i) \quad \Longrightarrow \quad \begin{array}{c}(*)\\ \vdots\\ \varphi \to \psi\end{array}$$

where it is assumed that the assumption number i does not occur in the proof represented by $(*)$.

4.5. Sums and products

The correspondence between derivations in NJ$(\to)$ and $\lambda_\to$ can be extended to the whole system of intuitionistic propositional logic if the simply typed λ-calculus is appropriately extended. The BHK interpretation of Chapter 2 gives an explicit guidance on how this extension should be made. First, a construction of a conjunction is a *pair*. That is, the formula $\varphi \wedge \psi$ corresponds to a product type, and could equally well be written as $\varphi \times \psi$. A canonical element of this type is a pair $\langle M, N\rangle$, where M is of type φ and N is of type ψ. The creation of a pair corresponds to $\wedge$-introduction. The two $\wedge$-elimination rules correspond to the two projections.

In type-free λ-calculus, pairs and projections were definable, but this is no longer the case in $\lambda_\to$. Indeed, as we have seen in Chapter 2 (Exercise 2.26), conjunction is not definable by any implicational formula. This means we have to add pairs and projections to the term language.

In the same spirit, $\varphi \vee \psi$ is a *disjoint sum* (or *variant*) type, also called *co-product*. An object of a variant type consists of data of either type φ or ψ together with a flag indicating the variant chosen.

It remains to observe that $\bot$, the formula with no construction, should correspond to the *empty type*.

4.5.1. WARNING. We use the logical symbols $\wedge$ and $\vee$ to denote also the corresponding types. Similar symbols are sometimes used to denote *intersection* and *union* types, which have quite a different meaning (see e.g. [26]), and which correspond to set-theoretic intersections and unions rather than to products and co-products.

4.5.2. DEFINITION.

(i) The syntax of *extended* Church-style simply typed lambda-calculus is defined by the type assignment rules of Figure 4.1. As in Chapter 3, this should be understood so that a "raw" expression becomes a term (in a given environment) only if it can be assigned a type according to these rules. The superscript in $\mathbf{in}_i^{\varphi\vee\psi}(M)$, which ensures type uniqueness, is often omitted for simplicity, and we write just $\mathbf{in}_i(M)$.

(ii) The free variables in a (raw) term are defined as usual, with the following clauses added to Definition 3.3.1(ii):

- $\mathrm{FV}(\langle M, N\rangle) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$;
- $\mathrm{FV}(\pi_i(M)) = \mathrm{FV}(\mathbf{in}_1(M)) = \mathrm{FV}(\mathbf{in}_2(M)) = \mathrm{FV}(\varepsilon_\varphi(M)) = \mathrm{FV}(M)$;
- $\mathrm{FV}(\mathbf{case}\, M\, \mathbf{of}\, [x]P\, \mathbf{or}\, [y]Q) = \mathrm{FV}(M) \cup (\mathrm{FV}(P) - \{x\}) \cup (\mathrm{FV}(Q) - \{y\})$.

That is, a variable in brackets is bound in the corresponding branch of a **case**-expression. Of course, the particular choice of bound variables does not matter; that is, we consider terms up to an (appropriately generalized) alpha-conversion. Substitution respects the bindings, so that for instance $(\mathbf{case}\ M\ \mathbf{of}\ [x]P\ \mathbf{or}\ [y]Q)[z := N]$ is defined as $\mathbf{case}\ M[z := N]\ \mathbf{of}\ [x]P[z := N]\ \mathbf{or}\ [y]Q[z := N]$, for $x, y \notin \mathrm{FV}(N)$.

(iii) The *beta-reduction* relation on extended terms is defined as the smallest compatible relation $\to_\beta$ extending the ordinary beta-reduction by:

$$\begin{array}{lll} \pi_1(\langle M_1, M_2\rangle) & \to_\beta & M_1; \\ \pi_2(\langle M_1, M_2\rangle) & \to_\beta & M_2; \end{array}$$

$$\begin{array}{lll} \mathbf{case}\ \mathbf{in}_1^{\varphi\vee\psi}(N)\ \mathbf{of}\ [x]P\ \mathbf{or}\ [y]Q & \to_\beta & P[x := N]; \\ \mathbf{case}\ \mathbf{in}_2^{\varphi\vee\psi}(N)\ \mathbf{of}\ [x]P\ \mathbf{or}\ [y]Q & \to_\beta & Q[y := N]. \end{array}$$

The intuitive meaning of pairs $\langle M, N\rangle$ and projections $\pi_i(M)$ in the rules of Figure 4.1 should be obvious. Also, the reduction rules (a projection applied to a pair returns a component) are as expected. As far as disjunction is concerned, a term of the form $\mathbf{in}_1^{\varphi\vee\psi}(M)$ or of the form $\mathbf{in}_2^{\varphi\vee\psi}(N)$ represents the conversion of $M : \varphi$ (resp. $N : \psi$) into an element of the disjoint sum. These are the canonical elements of $\varphi \vee \psi$. The actual contents of a given $M : \varphi \vee \psi$ is examined by a **case**-statement $\mathbf{case}\, M\, \mathbf{of}\, [x]P\, \mathbf{or}\, [y]Q$. Depending on the result, either the first branch or the second one is executed.

Finally, the expression $\varepsilon_\varphi(M)$ is a *miracle of type* φ. If $M : \bot$ then M is "the thing which is not". Once we have achieved the impossible, we should be able to do whatever we wish.

4.5.3. Proposition (Curry-Howard isomorphism).

(i) *If* $\Gamma \vdash M : \varphi$ *then* $\mathrm{rg}(\Gamma) \vdash \varphi$.

(ii) *If* $\Delta \vdash \varphi$, *then there are* M *and* Γ *with* $\Gamma \vdash M : \varphi$ *and* $\mathrm{rg}(\Gamma) = \Delta$.

There is a general pattern in the above rules, related to the duality between introduction and elimination in natural deduction. Each data type (function space, product, sum) comes with its own *constructors*. The constructors are operators *creating* canonical objects of the particular types and the destructors *use* these objects in computation. It may happen that

$$\Gamma, x:\psi \vdash x:\psi$$

$$\frac{\Gamma,\, x:\varphi \vdash M:\psi}{\Gamma \vdash (\lambda x{:}\varphi\, M):\varphi\to\psi} \qquad \frac{\Gamma \vdash M:\varphi\to\psi \quad \Gamma \vdash N:\varphi}{\Gamma \vdash (MN):\psi}$$

$$\frac{\Gamma \vdash M:\psi \quad \Gamma \vdash N:\varphi}{\Gamma \vdash \langle M,N\rangle:\psi\wedge\varphi} \qquad \frac{\Gamma \vdash M:\psi\wedge\varphi}{\Gamma \vdash \pi_1(M):\psi} \qquad \frac{\Gamma \vdash M:\psi\wedge\varphi}{\Gamma \vdash \pi_2(M):\varphi}$$

$$\frac{\Gamma \vdash M:\varphi}{\Gamma \vdash \mathbf{in}_1^{\varphi\vee\psi}(M):\psi\vee\varphi} \qquad \frac{\Gamma \vdash M:\psi}{\Gamma \vdash \mathbf{in}_2^{\varphi\vee\psi}(M):\psi\vee\varphi}$$

$$\frac{\Gamma \vdash L:\varphi\vee\psi \quad \Gamma,\, x:\varphi \vdash M:\rho \quad \Gamma,\, y:\psi \vdash N:\rho}{\Gamma \vdash (\mathbf{case}\ L\ \mathbf{of}\ [x]M\ \mathbf{or}\ [y]N):\rho}$$

$$\frac{\Gamma \vdash M:\bot}{\Gamma \vdash \varepsilon_\psi(M):\psi}$$

FIGURE 4.1: EXTENDED SIMPLY TYPED LAMBDA-CALCULUS

a destructor decomposes a complex object and returns a simpler object of a component type. This is how a projection works, but notice that the result of applying a **case**-statement to an object of a sum type may be of an arbitrary type. However there is always an analogy between a proof detour and a beta-redex. The introduction-elimination detour corresponds to a constructor directly passed to a destructor. A projection applied to a pair, an abstraction given an argument, or a **case**-dispatch on an explicitly indicated variant—all these expressions are ready for evaluation and may be subject to a reduction step.

We omit the proof of the following theorem, because it is a corollary of stronger results we consider later in the book. See Exercises 6.16 and 11.29.

4.5.4. THEOREM. *The extended simply typed lambda-calculus is strongly normalizing.*

A consequence of the above is the Church-Rosser property, which can be obtained with help of Newman's lemma, after an easy verification of WCR (cf. Section 3.6).

A small summary. Various other aspects and components of proofs also correspond to computational phenomena. For instance, the natural deduction axiom (a free assumption) is reflected by the use of a variable in a term. The provability problem translates to the inhabitation problem. The subject reduction property means that removing detours in a proof yields a proof of the same formula, and the Church-Rosser theorem states that the order of proof normalization is immaterial. Here is a little glossary:

logic	lambda-calculus
formula	type
propositional variable	type variable
connective	type constructor
implication	function space
conjunction	product
disjunction	disjoint sum
absurdity	empty type
proof	term
assumption	object variable
introduction	constructor
elimination	destructor
proof detour	redex
normalization	reduction
normal proof	normal form
provability	inhabitation

In the remainder of this book the concepts corresponding to one another under the isomorphism are often used interchangeably. Systems to be introduced in the following chapters will be considered from these two points of view: as typed lambda-calculi and as logics. And our glossary will expand.

4.6. Prover-skeptic dialogues

Recall the BHK-interpretation from Chapter 2. We can imagine the development of a construction for a formula as a dialogue between a *prover*, who is to produce the construction, and a *skeptic*, who doubts that it exists.

Prover 1: I assert that $((p \to p) \to (q \to r \to q) \to s) \to s$ holds.

Skeptic 1: Really? Let us suppose that I provide you with a construction of $(p \to p) \to (q \to r \to q) \to s$. Can you then give me one for s?

Prover 2: I got it by applying your construction to a construction of $p \to p$ and then by applying the result to a construction of $q \to r \to q$.

Skeptic 2: Hmm... you are making two new assertions. I doubt the first one the most. Are you sure you have a construction of $p \to p$? Suppose I give you a construction of p, can you give me one back for p?

Prover 3: I can just use the same construction!

A *dialogue* starts with the prover making an *assertion.* Based on the prover's assertion, the skeptic presents a list of *offers* to the prover and then presents a *challenge.* The offers do not consist of *actual* constructions; rather, the skeptic is telling the prover to continue under the assumption that she has such constructions. The prover must meet the challenge using these offers as well as previous offers. In doing so, the prover is allowed to introduce new assertions. The skeptic then reacts to each of these assertions, etc.

Here is another dialogue starting from the same assertion (the first three steps are identical to the preceding dialogue):

Skeptic 2': Hmm... you are making two new assertions. I doubt the second one the most. Are you sure you have a construction of $q \to r \to q$? If I give you constructions of q and r, can you give me one for q?

Prover 3': I can use the one you just gave me!

If the prover, in a given step, introduces several new assertions, then the skeptic may challenge any one of them—but only one, and always one from the latest prover step. Conversely, the prover must always respond to the latest challenge, but she may use any previous offer that the skeptic has made, not just those from the latest step.

A dialogue *ends* when the player who is up cannot respond, in which case the other player *wins;* here we think of the dialogues as a sort of game or debate. The skeptic has no response, if the prover's last step did not introduce new assertions to address the skeptic's last challenge. The prover has no response, if the skeptic's last step introduced a challenge that cannot be addressed using any of the skeptic's offers.

A dialogue can continue infinitely, which is regarded as a win for the skeptic. For example, the prover starts by asserting $(a \to b) \to (b \to a) \to a$, the skeptic offers $a \to b$ and $b \to a$ and challenges the prover to produce a construction of a. She can do this by applying the second offer to an alleged construction of b, which the skeptic will then challenge the prover to construct. Now the prover can respond by applying the first offer to an alleged construction of a, etc.

A *prover strategy* is a technique for arguing against the skeptic. When the skeptic has a choice—when the prover introduces more than one new assertion—the prover has to anticipate all the different choices the skeptic can make and prepare a reaction to each of them. Such a strategy can be represented as a tree where every path from the root is a dialogue, and where

every node labeled with a prover step has as many children as the number of assertions introduced by the prover in her step; each of these children is labeled with a skeptic step challenging an assertion. For instance, the above two dialogues can be collected into the prover strategy in Figure 4.2. A prover strategy is *winning* if all its dialogues are won by the prover.

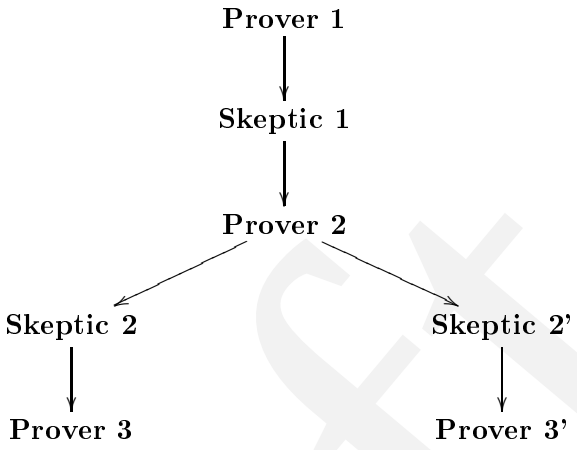


Figure 4.2: Prover strategy

One can extract the construction for a formula from a winning prover strategy. Indeed, if we accept simply typed λ-terms of type φ as constructions of φ, then the construction is obtained by viewing the skeptic's steps as lambda-abstractions and the prover's steps as applications, as outlined in Figure 4.3.

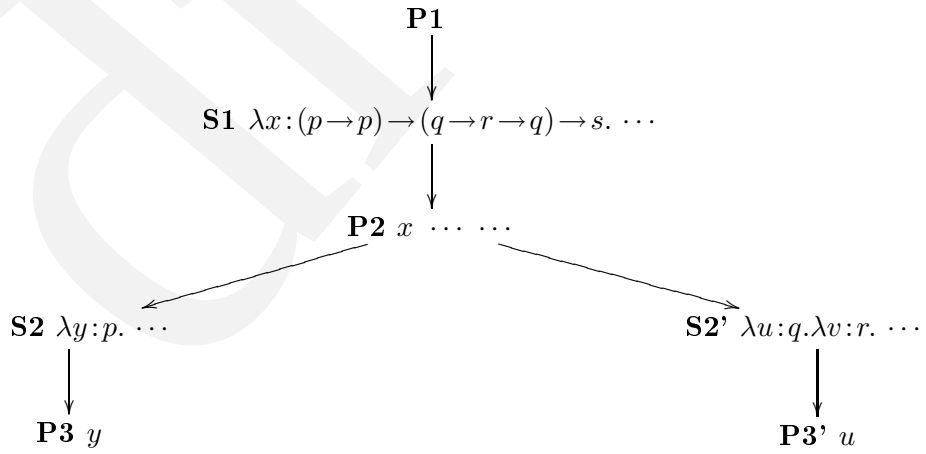


Figure 4.3: Constructions from strategies

In fact, we can think of the extracted λ-term, which is an η-long normal form, as a convenient representation of the strategy. And the construction of long forms in the Ben-Yelles algorithm (see Lemma 4.2.3) can be seen as the search for a winning prover strategy.

We now define these concepts rigorously for minimal implicational logic. For technical reasons it is convenient to consider dialogues (about some formula φ) relative to a collection Γ of offers provided in advance.

WARNING. The dialogues considered in this chapter are somewhat different than the ones originating with Lorenzen—see Chapter 6.

4.6.1. DEFINITION. A *dialogue* over (Γ, φ) is a possibly infinite sequence $(\Sigma_1, \alpha_1), (\Pi_1, \beta_1), (\Sigma_2, \alpha_2), (\Pi_2, \beta_2) \ldots$, where α_i, β_j are propositional variables and Σ_i, Π_j are finite or empty sequences of formulas, satisfying:

(i) $\Sigma_1 = \varphi$ and $\alpha_1 = \top$. (Prover begins.)[3]

(ii) $\alpha_{i+1} = \beta_i$. (Prover meets the challenge of the preceding step.)

(iii) $\Sigma_{i+1} \to \alpha_{i+1} \in \Gamma \cup \Pi_1 \cup \ldots \cup \Pi_i$. (Prover uses an available offer.)[4]

(iv) $\Pi_i \to \beta_i \in \Sigma_i$. (Skeptic challenges a formula from the preceding step.)

A *winning prover strategy for* (Γ, φ) (or just "for φ" when Γ is empty) is a labeled tree where:

- Every branch is a dialogue over (Γ, φ).
- Every node labeled with a prover step with n assertions has n children labeled with distinct steps.
- Every node labeled with a skeptic step has one child.
- All branches are finite and end in a leaf labeled with a prover step with no assertions.

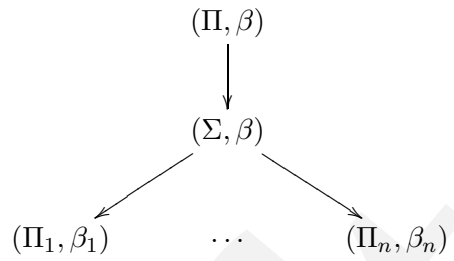
A prover step (Σ, α) means meeting challenge α by introducing new assertions Σ. A skeptic step (Π, β) means a challenge to prove β with permission to use offers Π. An assertion (e.g. the initial formula) may be atomic; in this case, the prover will be challenged to prove it without any additional offers.

4.6.2. THEOREM. *There is a winning prover strategy for φ iff φ is a theorem in minimal implicational logic*

[3] The choice $\alpha_1 = \top$ in this case is an arbitrary convention.

[4] $(\psi_1, \ldots, \psi_n) \to \varphi$ means $\psi_1 \to \cdots \to \psi_n \to \varphi$. For $n = 0$, this boils down to just φ.

PROOF. For the direction from left to right, let D be a winning prover strategy for φ. For each skeptic node (Π, β), we consider the accumulated set of offers Γ, i.e. the union of all offers made in the dialogue from the root to this node. For any subtree D' with root (Π, β) that has accumulated offers Γ, we now show that $\Gamma \vdash \beta$ in minimal implicational logic. The form of D' must be ($n \geq 0$):



Each child has accumulated offers $\Gamma \cup \Pi_i$ (viewing Π_i as the set of its elements). By the induction hypothesis $\Gamma, \Pi_i \vdash \beta_i$, so $\Gamma \vdash \Pi_i \to \beta_i$. Since $\Sigma = \Pi_1 \to \beta_1, \ldots, \Pi_n \to \beta_n$, also $\Gamma \vdash (\Pi_1 \to \beta_1) \to \cdots \to (\Pi_n \to \beta_n) \to \beta$ (an equivalent offer must have been stated) so $\Gamma \vdash \beta$ by *modus ponens*.

In particular, D has root label $((\varphi), \top)$ and single child labeled (Π, β), where $\varphi = \Pi \to \beta$, so $\Pi \vdash \beta$, i.e. $\vdash \varphi$.

For the opposite direction, assume $\vdash \varphi$. There is a closed λ-term of type φ. By normalization and Lemma 4.2.2 we can assume that the term is in η-long normal form, so it suffices to show that if

$$x_1 : \psi_1, \ldots, x_n : \psi_n \vdash M : \varphi$$

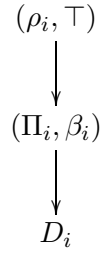
for η-long normal form M, then there is a winning prover strategy for $(\{\psi_1, \ldots, \psi_n\}, \varphi)$. We proceed by induction on the size of M. Let

$$M = \lambda x_{n+1}^{\psi_{n+1}} \ \ldots \ x_{n+m}^{\psi_{n+m}} . \, x_k \, N_1 \ldots N_l,$$

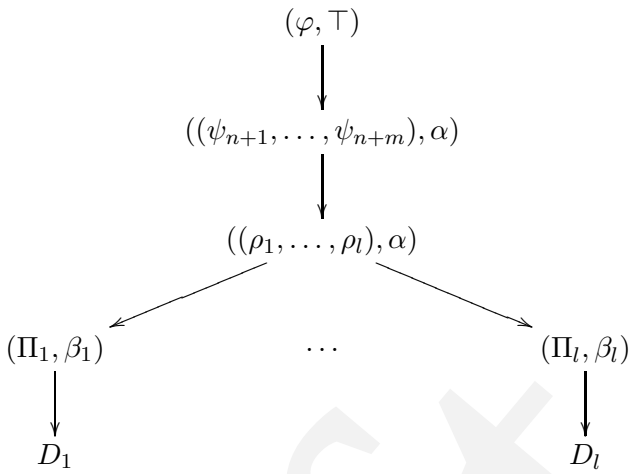
where $1 \leq k \leq n + m$. Then

$$x_1 : \psi_1, \ldots, x_{n+m} : \psi_{n+m} \vdash N_i : \rho_i$$

for each i, where $\psi_k = \rho_1 \to \ldots \rho_l \to \alpha$ and $\varphi = \psi_{n+1} \to \ldots \to \psi_{n+m} \to \alpha$. By the induction hypothesis, there is a winning prover strategy for each $(\{\psi_1, \ldots, \psi_{n+m}\}, \rho_i)$. It must have form



Then we have the following strategy for $(\{\psi_1, \ldots, \psi_n\}, \varphi)$:



This concludes the proof. □

In the direction from left to right we could construct λ-terms of type β rather than prove $\Gamma \vdash \beta$, and it is not difficult to see that these are actually η-long normal forms. Thus we can state the following identification:

$$\text{Winning prover strategy} \quad \sim \quad \eta\text{-long normal form.}$$

More precisely, in a term

$$\lambda x_1^{\psi_1} \ldots x_n^{\psi_n}.\, x_i\, M_1 \ldots M_m,$$

the abstractions correspond to offers made by the skeptic, and the type of $x_i M_1 \ldots M_m$ is the challenge by the skeptic. The variable x_i represents the prover's choice of offer, and the terms $M_1, \ldots, M_m$ represent the continued dialogue in each branch of the winning prover strategy.

4.7. Prover-skeptic dialogues with absurdity

What was said above concerns minimal logic. What happens when we add absurdity and the rule *ex falso sequitur quodlibet*? The BHK interpretation states that "there is no construction of $\bot$." If the skeptic actually makes $\bot$ as an offer, or makes an offer from which $\bot$ can be inferred, he is creating false assumptions. In this case the prover should be permitted to abort the current challenge—the skeptic is not playing by the book. The prover can provoke such circumstances by making assertions that will force the skeptic to provide contradictory information, as in the following example.

Prover 1: I assert that $p \to \neg p \to q$ holds!

Skeptic 1: Really? Suppose I give you constructions of p and $\neg p$, can you give me a construction of q?

Prover 2: I do not believe you can provide such constructions. By applying your second offer to a construction of p, I get absurdity, and that's impossible!

Skeptic 2: Sure, but I doubt you have a construction of p.

Prover 3: I can just use the one you gave me earlier on. I win!

4.7.1. Definition. Dialogues and strategies are as in Definition 4.6.1 except that α, β now range over all atomic propositions, i.e. not only propositional variables, but also $\bot$. (This will be the convention for the rest of this section.) Moreover, condition (ii) is replaced by:

(ii) $\alpha_{i+1} = \beta_i$ (prover meets challenge of preceding step) or $\alpha_{i+1} = \bot$ (prover aborts preceding challenge).

Note that if the prover has to make new assertions to derive absurdity, then the skeptic will challenge these, so it is not the dialogue as such that is aborted, only the latest challenge.

In order to represent strategies as terms, we use the extended simply typed λ-calculus from Section 4.5 without pairs and sums, but with ε, here denoted $\lambda\varepsilon$-calculus. The following reduction rules are needed.

4.7.2. Definition. The relation $\to_\varepsilon$ is the smallest compatible relation containing the following rules:

$$\varepsilon_{\varphi\to\psi}(M)N \to_\varepsilon \varepsilon_\psi(M) \qquad (\varepsilon_1)$$

$$\varepsilon_\psi(\varepsilon_\bot(M)) \to_\varepsilon \varepsilon_\psi(M) \qquad (\varepsilon_2)$$

The term $\varepsilon_\varphi(M)$ expresses the fact that the prover has inferred absurdity from a skeptic offer, and that she is aborting the attempt to meet the skeptic's challenge. We can view this as though the prover has produced an imaginary construction of φ. The reduction rule (ε_1) captures the fact that rather than producing an imaginary construction of an implication $\varphi \to \psi$ and then applying this construction to a construction of φ to obtain a construction of ψ, we might as well "abort" the argument and produce an imaginary construction of ψ directly.

In order to relate the revised prover-skeptic games to provability in intuitionistic, implicational logic we need some preparations.

4.7.3. Theorem. *The reduction $\to_{\beta\varepsilon}$ is normalizing.*

PROOF. First reduce a term to β-normal form. From the point of view of β-reductions, there is no difference between $\varepsilon_\varphi(M)$ and $x^{\bot\to\varphi}M$, where x is a fresh variable, so by the normalization theorem, such a β-normal form exists. Then reduce to ε-normal form, which is possible since each reduction decreases the term size. Finally note that ε-reduction steps cannot create β-redexes, so the resulting term is still in β-normal form. □

4.7.4. DEFINITION. The set of η-long normal forms of $\lambda\varepsilon$-calculus is the smallest set closed under the following rules (where $m \geq 0$):

- If y has type $\tau_1 \to \cdots \to \tau_n \to \alpha$ and M_i is an η-long normal form of type τ_i, then the following term is also an η-long normal form:
$$\lambda y_1^{\sigma_1} \ldots y_m^{\sigma_m}.\, yM_1 \ldots M_n.$$
- If y has type $\tau_1 \to \cdots \to \tau_n \to \bot$ and M_i is an η-long normal form of type τ_i, then the following term is also an η-long normal form:
$$\lambda y_1^{\sigma_1} \ldots y_m^{\sigma_m}.\, \varepsilon_\alpha(yM_1 \ldots M_n).$$

4.7.5. LEMMA. *For every simply typed $\lambda\varepsilon$-term, there is another simply typed $\lambda\varepsilon$-term of the same type in η-long normal form.*

PROOF. Given a term of type $\varphi = \sigma_1 \to \cdots \to \sigma_m \to \alpha$, we proceed by induction on the term. We can assume it is in $\beta\varepsilon$-normal form, i.e.

$$\lambda y_1^{\sigma_1} \ldots y_k^{\sigma_k}.\, \varepsilon_\psi(yM_1 \ldots M_n) \quad \text{or} \quad \lambda y_1^{\sigma_1} \ldots y_k^{\sigma_k}.\, yM_1 \ldots M_n,$$

where $k \leq m$ and $\psi = \sigma_{k+1} \to \cdots \to \sigma_m \to \alpha$. By the induction hypothesis, we can let N_i be an η-long normal form of M_i. Then the desired term is, respectively:

$$\lambda y_1^{\sigma_1} \ldots y_m^{\sigma_m}.\, \varepsilon_\alpha(yN_1 \ldots N_n) \quad \text{or} \quad \lambda y_1^{\sigma_1} \ldots y_m^{\sigma_m}.\, yN_1 \ldots N_n\, y_{k+1} \ldots y_m. \quad \square$$

4.7.6. THEOREM. *There is a winning prover strategy for φ iff φ is a theorem in intuitionistic implicational logic.*

PROOF. Similar to the proof of Theorem 4.6.2. □

4.8. Notes

Surprisingly, the ancient Greeks probably did not know about the Curry-Howard isomorphism. But some authors trace the roots of the idea back to Schönfinkel, Brouwer, Heyting, or even Frege and Peano.

Although it is hard to deny that the proof interpretation of Brouwer-Heyting-Kolmogorov is fundamental for the propositions-as-types correspondence, we would not go so far as to claim that the BHK interpretation and Curry-Howard isomorphism are the same. The formulas-as-types paradigm is one possible way of understanding BHK. And e.g. realizability (see Chapter 9) can be seen as another.

FORMULAS-AS-TYPES. To our knowledge, the first explicit statement that may be considered the beginning of the subject occurs in Curry's [102], as footnote 28:

Note the similarity of the postulates for F and those for P. If in any of the former postulates we change F to P and drop the combinator we have the corresponding postulate for P.

Here P stands for implication and F for a "functionality" combinator, such that $FABf$ essentially means as much as $f : A \to B$. In fact, Curry must have realized the similarity several years before. As pointed out by Hindley [237], a remark on page 588 of [100] is a hint of that. In the same paper, and also in [101], there is another hint. Properties of $\supset$ are named PB, PC, PW and PK, after the combinators B, C, W and K. From [439] we know that Curry used this notation as early as 1930. Years later, Carew Meredith independently observed the analogy between proofs and combinators, cf. [237, 339].

The correspondence was made precise (for typed combinatory logic) by Curry and Feys in [107, Chapter 9] by means of two theorems. Theorem 1 (p. 313) states that inhabited types are provable, and Theorem 2 (p. 314) states the converse.

Before Howard, various authors, e.g. Kreisel, Goodman, and Scott, contributed to the understanding of the relationship between formulas and types. In the paper of Läuchli [298], preceded by the quarter-page abstract [297], formulas are assigned sets and provability corresponds to non-emptiness. Stenlund [459], who probably first used the word "isomorphism", points out the inspiration coming from the *Dialectica* interpretation, so that also Gödel should be counted as one of the fathers of the paradigm. However, all these works are relevant to only one aspect of what we call Curry-Howard isomorphism: the essentially "static" correspondence between provable theorems and inhabited types.

REDUCTION AS COMPUTATION. The other important aspect is the relationship between proof normalization and computation (term reduction). The paper often credited for the discovery of this is Tait's [466]. For instance, Howard himself declares in [247] that he was inspired by Tait. Instead of *proving* that an expression of type "number" is equal to a certain number, Tait *reduces* the expression to normal form. Thus, in a sense, term reduction is used instead of proof normalization. But the statement of the propositions-as-types paradigm can hardly be considered explicit in [466]. Also Curry and Feys, who used cut-elimination in [107] to prove the normalization theorem for simply typed terms, seem not to pay real attention to the relationship between proof reduction and computation. An explicit statement of the "dynamic" aspect of the isomorphism did not occur in the literature until 1969, when the famous paper of Howard began to be "privately circulated".

Howard did not publish his manuscript [247] until 1980 when it became part of Curry's Festschrift [441]. By then, the paper was already well-known, and the idea of formulas-as-types had gained considerable popularity. (As early as 1969, Martin-Löf applied Howard's approach in a strong normalization proof.)

The dozen pages of [247] explain the fundamentals of the proofs-as-terms paradigm on the level of implicational logic, with explicit emphasis on the relationship between normalization of terms and normalization of proofs. The approach is then extended to other propositional connnectives, and finally a term language for Heyting Arithmetic (first-order intuitionistic arithmetic) is introduced and discussed.

PROOFS AS OBJECTS. While Curry and Howard are given fair credit for their discovery, the contribution of Nicolaas Govert de Bruijn is not always properly recognized. De Bruijn not only made an independent statement of the correspondence

between proofs and objects, but he was also the first to make practical use of it. His "mathematical language" Automath [58, 60, 57, 360], developed in Eindhoven from 1967 onward (at the same time when Howard's paper was written), was a tool for writing mathematical proofs in a precise way so that they could be verified by a computer. Automath became an inspiration for the whole area of logical frameworks and proof assistants. Many fundamental concepts were for the first time used in Automath. Dependent types, with implication defined as a special case of universal quantifier is an outstanding example. The idea of proofs as objects is exploited from the very beginning. Propositions are represented as sets ("categories") of their proofs, and the question of provability becomes the question of non-emptiness.

Scott appreciated the importance of de Bruijn's work very early. The influential paper [437], which appeared in the same volume as [58], applied the concepts and ideas from Automath (designed for classical proof encoding) in an intuitionistic "calculus of constructions".

In the early 1970's also Per Martin-Löf began his effort to develop a constructive type theory, aimed at providing a foundational basis for constructive mathematics. After overcoming some initial difficulties (the first version was inconsistent—this is now called *Girard's paradox*) Martin-Löf's type theory [325, 326, 327] turned out to be very successful. It is often said to be as important for constructivism as axiomatic set theory is for classical mathematics. In particular, Martin-Löf's work was essential for our present understanding of the Curry-Howard isomorphism as the *identification* between propositions and types. Not so unexpectedly, Martin-Löf's theory can be seen as a prototypic programming language [328, 365, 366] and a programming logic in one.

As mentioned above, Automath preceded many systems introduced later with the aim of mechanizing the process of proof design and proof verification. The design of such systems is often more or less related to the "proofs-as-objects" paradigm, see [33]. Among these, let us name a few.

The acronym LCF (for "Logic of Computable Functions") denotes several versions of a system originating in 1972 from a proof-checker designed by Milner. Currently, the most important descendants of LCF are system HOL (for "Higher-Order Logic") and Isabelle, see [197, 364].

Another line of research, inspired by Martin-Löf's type theory, was undertaken by Constable and his co-workers also in the early 1970's. The "Program Refinement Logic" (or "Proof Refinement Logic") PRL gave rise to the system Nuprl, currently one of the most well-known and actively evolving systems for proof development and software verification [81, 82].

The systems of the 1970's can all be recognized as inspiration for the Formel project initiated in France at the beginning of the 1980's. An important theoretical tool in this research was the Calculus of Constructions proposed by Coquand and Huet in mid 1980's and then developed and extended by Luo, Paulin-Mohring and others. The Calculus of Constructions, to which we return in Chapter 14, provided the formalism on which the proof-assistants Coq [44] and Lego are based.

PROOFS INTO PROGRAMS. *Witness extraction* is the problem of extracting from a proof of $\exists x P(x)$ a specific value a such that $P(a)$ holds. Similarly, one can consider the problem of extracting from a proof of $\forall x \exists y P(x, y)$ a function f such that $\forall x P(x, f(x))$; this is called *program extraction*. For instance, a constructive proof of a specification, say of the form

$$\forall x\, \exists y.\, \mathit{ordered}(y) \wedge \mathit{permutation}(x, y)$$

should implicitly contain a sorting algorithm. The idea that a proof of a specification might be turned into a program fulfilling this specification attracted researchers in theoretical computer science already in the 1960's, and various techniques were applied in this direction, see e.g. [321]. Witness extraction methods based on resolution proofs developed in the late 1960's contributed to the birth of what is now called logic programming. For instance, Constable [79] and then Marini and Miglioli [322] discussed the idea of extracting an algorithm from a proof with the help of Kleene's realizability approach. Constable's PRL and Nuprl were partly motivated by this direction of work. A few years later, Goto and Sato investigated a program extraction procedure based on the *Dialectica* interpretation (see Chapter 10). The Japanese school provided the background on which Hayashi invented the system PX [217].

The Curry-Howard isomorphism identifies a proof (of a specification) and the program (meeting that specification). Thus the program extraction process simply amounts to deleting all "computationally irrelevant" contents from the proof. In Chapter 12 we will see an example of a direct program extraction procedure in the proof of Girard's result: If $\forall x \exists y P(x, y)$ is provable in second-order arithmetic, then a function f satisfying $\forall x P(x, f(x))$ is definable in the polymorphic λ-calculus.

Issues related to Curry-Howard style program extraction occur in the literature since mid 1970's. See [25, 43, 80, 191, 280, 288, 382, 386] as a small sample of that category. Although the issue of practical software development based on these ideas is still a matter of the future, the research goes on (see e.g. the system MINLOG of [430]) and the last word is certainly not said yet in this area.

MISCELLANEA. To make a few more specific comments, the decidability of propositional intuitionistic logic was already known to Gentzen in 1935 and we will discuss his (syntactic) proof in Chapter 7. The algorithm we used in the proof of Lemma 4.2.3, implicit in in Wajsberg's [505], is often referred to as the *Ben-Yelles algorithm*, for Choukri-Bey Ben-Yelles, who defined a procedure to *count* the number of inhabitants of a type (see [237]). A polynomial space algorithm (based on a Kripke model construction) was also given by Ladner [292], and the PSPACE-completeness is from Statman [455]. There is also a semantical PSPACE-hardness proof by Švejdar [463].

Normalization for natural deduction proofs with all the standard propositional connectives was first shown by Prawitz [403], and variants of this proof occur in newer books like [113, 489]. Strong normalization proofs can be found e.g. in [398] and [160], and for especially short ones the reader is referred to [122, 255]. The latter paper only covers the case of implication and disjunction, but that is quite enough, see Exercise 4.16.

We are concerned in this book with the correspondence between logics and lambda-calculi. However, there is another correspondence: between logics and dialogue games. The "dialogical" approach to logic originates with Lorenzen [315], and works by Lorenz [314], Felscher [147] and others contributed to the subject (see notes to Chapters 6 and 7). However, the games considered in this chapter are somewhat different than those of Lorenzen. We return to the latter type in Chapter 7. There are various other game-related approaches to logic, of which we only mention Girard's new concept of *ludics* [186, 187].

If we look at the formulas-as-types paradigm from the point of view of categorical semantics, we discover yet another side of the same correspondence. Objects and morphisms correspond to types and terms, and also to formulas and proofs.

This analogy was first discovered by Lawvere [300] and Lambek. See [18, 294] for more on that subject.

Various other forms of the correspondence have been discovered in recent years, and many old ideas turned out to be relevant. For instance, Ajdukiewicz's and Lambek's grammars, used in the analysis of natural languages [40, 64, 293, 354], have a close relationship to type assignment, as first observed by Curry [107, p. 274]. Papers [16, 123, 127, 128, 287, 372, 373, 497] discuss various other faces of the Curry-Howard correspondence, related to modal logic, illative combinatory logic, intersection types, low-level programming, etc. etc. Among these, linear logic (see the notes to Chapter 5) takes a distinguished place.

The historical article [247], as well as the crucial section 9E of [107] are reprinted in the collection [206]. Other articles included in [206] cover various aspects of the Curry-Howard correspondence. Hindley's book [237] contains a detailed study of the paradigm on the simply typed level. In particular, the exact correspondence between closed terms and assumption-free proofs (cf. Exercise 4.8) is explicitly formulated there as Theorem 6B7(iii). The reader will enjoy the explanation in [22] on how the isomorphism extends to various data types and how the "inversion principle" works on these. For a further and deeper study of the Curry-Howard interpretation of inductive types and families, see [139, 140]. A textbook presentation of the Curry-Howard isomorphism can be found in [189], and a recent monograph [444] takes a serious look at the subject. Among various survey articles of the Curry-Howard isomorphism we recommend [85, 161, 238, 487, 504].

4.9. Exercises

4.1. (From [510].) Are the following formulas intuitionistically valid?

(i) $((p \to q) \to r) \to (p \to r) \to r$?

(ii) $((((p \to q) \to r) \to (p \to r) \to r) \to q) \to q$?

Hint: Are these types inhabited?

4.2. Show that **K** and **S** are the only closed normal forms of types $p \to q \to p$ and $(p \to q \to r) \to (p \to q) \to p \to r$, respectively.

4.3. For $n \in \mathbb{N} \cup \{\aleph_0\}$, show examples of types with exactly n normal inhabitants.

4.4. Run the Ben-Yelles algorithm on the inputs:

(i) $(p \to p) \to p$;

(ii) $((p \to q) \to q) \to (p \to q)$.

4.5. (Ben-Yelles) Show that it is decidable whether a type has a finite or an infinite number of different normal inhabitants. *Hint:* Modify the proof of Lemma 4.2.3.

4.6. Observe that the proof of Lemma 4.2.3 makes it possible to actually construct an inhabitant of a given type if it exists. Show that the size of this inhabitant is at most exponential in the size of the given type. Then show that the exponential bound is best possible. That is, for every n define a type τ_n of length $\mathcal{O}(n)$ such that every normal inhabitant of τ_n is of length at least of order 2^n. (If you find that too easy, delete the word "normal".)

4.7. Recall that the implication introduction rule in our natural deduction system

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\mathrm{I}\to)$$

is understood so that Γ, φ abbreviates $\Gamma \cup \{\varphi\}$. It may be the case that $\varphi \in \Gamma$. In the "traditional" proof notation this is expressed by the possibility of not discharging the assumption φ. What will change if we instead choose the *complete discharge convention:* When introducing an implication $\varphi \to \psi$, discharge *all* occurrences of the assumption φ?

4.8. Define precisely a natural deduction calculus with labeled discharges (in the style of Section 4.3). Establish a bijective correspondence between closed λ-terms and proofs in your system without free assumptions.

4.9. Let $\varphi = \tau_1 \to \cdots \to \tau_n \to p$ be an implicational formula and let p be the only type variable occurring in φ. Prove that φ is an inhabited type if and only if at least one of the τ_i's is *not* inhabited.

4.10. (Statman) Let p be the only propositional variable, and let $\to$ be the only connective occurring in a classical propositional tautology φ. Show that φ is intuitionistically valid.

4.11. (Prucnal, Dekkers [126]) A proof rule of the form

$$\frac{\tau_1, \ldots, \tau_n}{\tau}$$

is *sound* for IPC($\to$) iff for every substitution S with $S(\tau_1), \ldots S(\tau_n)$ all valid, also $S(\tau)$ must be valid. Prove that if such a rule is sound then the implication $\tau_1 \to \cdots \to \tau_n \to \tau$ is valid.

4.12. Write an inhabitant of type $(p \to \neg p) \to (\neg p \to p) \to \bot$.

4.13. Find inhabitants for the formulas in Example 2.1.3 in the extended calculus of Section 4.5.

4.14. Consider again Exercises 2.21–2.24. Give inhabitants for all the tautologies.

4.15. Prove that the simply typed lambda-calculus extended with product types, pairs and projections (no disjunction types or $\bot$) is strongly normalizing.

4.16. Assume that the simply typed lambda-calculus extended with disjunction types, injections and **case**-statements has the strong normalization property. Prove the SN property for the whole system of Section 4.5 (Theorem 4.5.4).

4.17. How should the notion of η-reduction be generalized for sums and products?

4.18. Design a Church-style calculus with $\wedge$ and $\vee$ and show that subject reduction property holds for that calculus for both beta- and eta-reductions

4.19. Define a Curry-style variant of lambda-calculus with $\wedge$ and show that the subject reduction property does not hold for η-reductions.

4.20. Explain the difference between the above two results.