Lectures on the

Curry-Howard Isomorphism

Morten Heine B. Sørensen
University of Copenhagen

Paweł Urzyczyn
University of Warsaw

Preface

The Curry-Howard isomorphism states an amazing correspondence between systems of formal logic as encountered in *proof theory* and computational calculi as found in *type theory.* For instance, minimal propositional logic corresponds to simply typed λ-calculus, first-order logic corresponds to dependent types, second-order logic corresponds to polymorphic types, etc.

The isomorphism has many aspects, even at the syntactic level: formulas correspond to types, proofs correspond to terms, provability corresponds to inhabitation, proof normalization corresponds to term reduction, etc.

But there is much more to the isomorphism than this. For instance, it is an old idea—due to Brouwer, Kolmogorov, and Heyting, and later formalized by Kleene's realizability interpretation—that a constructive proof of an implication is a procedure that transforms proofs of the antecedent into proofs of the succedent; the Curry-Howard isomorphism gives syntactic representations of such procedures.

These notes give an introduction to parts of proof theory and related aspects of type theory relevant for the Curry-Howard isomorphism.

Outline

Since most calculi found in type theory build on λ-calculus, the notes begin, in Chapter 1, with an introduction to *type-free λ-calculus.* The introduction derives the most rudimentary properties of β-reduction including the Church-Rosser theorem. It also presents Kleene's theorem stating that all recursive functions are λ-definable and Church's theorem stating that β-equality is undecidable.

As explained above, an important part of the Curry-Howard isomorphism is the idea that a constructive proof of an implication is a certain procedure. This calls for some elaboration of what is meant by constructive proofs, and Chapter 2 therefore presents *intuitionistic propositional logic.* The chapter presents a natural deduction formulation of minimal and intuitionistic propositional logic. The usual semantics in terms of Heyting algebras and in terms of Kripke models are introduced—the former explained on the basis of Boolean algebras—and the soundness and completeness results are then proved. An informal proof semantics, the so-called BHK-interpretation, is also presented.

Chapter 3 presents the *simply typed λ-calculus* and its most fundamental properties up to the subject reduction property and the Church-Rosser property. The distinction between simply typed λ-calculus à la Church and à la Curry is introduced, and the uniqueness of types property—which fails for the Curry system—is proved for the Church system. The equivalence between the two systems, in a certain sense, is also established. The chapter also proves the weak normalization property by the Turing-Prawitz method, and ends with Schwichtenberg's theorem stating that the numeric functions representable in simply typed λ-calculus are exactly the extended polynomials.

This provides enough background material for our first presentation of the *Curry-Howard isomorphism* in Chapter 4, as it appears in the context of natural deduction for minimal propositional logic and simpy typed λ-calculus. The chapter presents another formulation of natural deduction, which is often used in the proof theory literature, and which facilitates a finer distinction between similar proofs. The exact correspondence between natural deduction for minimal propositional logic and simply typed λ-calculus is then presented. The extension to product and sum types is also discussed. After a brief part on proof-theoretical applications of the weak normalization property, the chapter ends with a proof of strong normalization using the Tait-Girard method, here phrased in terms of saturated sets.

Chapter 5 presents the variation of the Curry-Howard isomorphism in which one replaces natural deduction by *Hilbert style proofs* and simply typed λ-calculus by simply typed *combinatory logic*. After type-free combinators and weak reduction—and the Church-Rosser property—the usual translations from λ-calculus to combinators, and vice versa, are introduced and shown to preserve some of the desired properties pertaining to weak reduction and β-reduction. Then combinators with types are introduced, and the translations studied in this setting. Finally Hilbert-style proofs are introduced, and the connection to combinators with types proved. The chapter ends with a part on subsystems of combinators in which relevance and linearity play a role.

Having seen two logics or, equivalently, two calculi with types, Chapter 6 then studies *decision problems* in these calculi, mainly the type checking, the type reconstruction, and the type inhabitation problem. The type reconstruction problem is shown to be P-complete by reduction to and from unification (only the reduction *to* unification is given in detail). The type inhabitation problem is shown to be PSPACE-complete by a reduction from the satisfiability problem for classical second-order propositional formulas. The chapter ends with Statman's theorem stating that equality on typed terms is non-elementary.

After introducing natural deduction systems and Hilbert-style systems, the notes introduce in Chapter 7 Gentzen's sequent calculus systems for propositional logic. Both classical and intuitionistic variants are introduced. In both cases a somewhat rare presentation—taken from Prawitz—with assumptions as sets, not sequences, is adopted. For the intuitionistic system the cut-elimination theorem is mentioned, and from this the subformula property and decidability of the logic are inferred. Two aproaches to term assignment for sequent calculus proofs are studied. In the first approach, the terms are those of the simply typed λ-calculus. For this approach, the connection between normal forms and cut-free proofs is studied in some detail. In the second approach, the terms are intended to mimic exactly the rules of the calculus, and this assignment is used to prove the cut-elimination theorem in a compact way.

The remaining chapters study variations of the Curry-Howard isomorphism for more expressive type systems and logics.

In Chapter 8 we consider the most elementary connections between natural deduction for *classical propositional logic* and simply typed λ-calculus with *control operators,* in particular, the correspondence between classical proof normalization and reduction of control operators. Kolmogorov's embedding of classical logic into intuitionistic logic is shown to induce a continuation passing style translation which eliminates control operators.

Chapter 9 is about *first-order logic.* After a presentation of the syntax for quantifiers, the proof systems and interpretations seen in earlier chapters are generalized to the first-order case.

Chapter 10 presents *dependent types,* as manifest in the calculus λP. The strong normalization property is proved by a translation to simply typed λ-calculus. A variant of λP à la Curry is introduced. By another translation it is shown that a term is typable in λP à la Curry iff it is typable in simply typed λ-calculus. While this shows that type reconstruction is no harder than in simply typed λ-calculus, the type checking problem in λP à la Curry turns out to be undecidable. The last result of the chapter shows that first-order logic can be encoded in λP.

In Chapter 11 we study *arithmetic.* The chapter introduces Peano Arithmetic (PA) and briefly recalls Gödel's theorems and the usual result stating that exactly the recursive functions can be represented in Peano Arithmetic. The notion of a provably total recursive function is also introduced. Heyting arithmetic (HA) is then introduced and Kreisel's theorem stating that provable totality in HA and PA coincide is presented. Then Kleene's realizability interpretation is introduced—as a way of formalizing the BHK-interpretation—and used to prove consistency of HA. Gödel's system **T** is then introduced and proved to be strongly normalizing. The failure of arithmetization of proofs of this property is mentioned. The result stating that the functions definable in **T** are the functions provably total in Peano Arithmetic is also presented. Finally, Gödel's *Dialectica* interpretation is presented and used to prove consistency of HA and to prove that all functions provably total in Peano Arithmetic are definable in **T**.

Chapter 12 is about *second-order logic* and *polymorphism.* For the sake of simplicity, only second-order propositional systems are considered. Natural deduction, Heyting algebras, and Kripke models are extended to the new setting. The polymorphic λ-calculus is then presented, and the correspondence with second-order logic developed. After a part about definability of data types, a Curry version of the polymorphic λ-calculus is introduced, and Wells' theorem stating that type reconstruction and type checking are undecidable is mentioned. The strong normalization property is also proved.

The last chapter, Chapter 13, presents the *λ-cube* and *pure type systems.* First Barendregt's cube is presented, and its systems shown equivalent to previous formulations by means of a classification result. Then the cube is geneneralized to pure type systems which are then developed in some detail.

About the notes

Each chapter is provided with a number of exercises. We recommend that the reader try as many of these as possible. At the end of the notes, answers and hints are provided to some of the exercises.[1]

The notes cover material from the following sources:

- Girard, Lafont, Taylor: *Proofs and Types,* Cambridge Tracts in Theoretical Computer Science 7, 1989.
- Troelstra, Schwichtenberg: *Basic Proof Theory,* Cambridge Tracts in Theoretical Computer Science 43, 1996.
- Hindley: *Basic Simple Type Theory,* Cambridge Tracts in Theoretical Computer Science 42, 1997.
- Barendregt: Lambda Calculi with Types, pages 117–309 of *Abramsky, S. and D.M. Gabbay and T.S.E. Maibaum*, editors, *Handbook of Logic in Computer Science*, Volume II, Oxford University Press, 1992.

Either of these sources make excellent supplementary reading.

The notes are largely self-contained, although a greater appreciation of some parts can probably be obtained by readers familiar with mathematical logic, recursion theory and complexity. We recommend the following textbooks as basic references for these areas:

- Mendelson: *Introduction to Mathematical Logic*, fourth edition, Chapman & Hall, London, 1997.

[1]This part is quite incomplete due to the "work-in-progress" character of the notes.

- Jones: *Computability and Complexity From a Programming Perspective,* MIT Press, 1997.

The notes have been used for a one-semester graduate/Ph.D. course at the Department of Computer Science at the University of Copenhagen (DIKU). Roughly one chapter was presented at each lecture, sometimes leaving material out.

The notes are still in progress and should not be conceived as having been proof read carefully to the last detail. Nevertheless, we are grateful to the students attending the course for pointing out numerous typos, for spotting actual mistakes, and for suggesting improvements to the exposition.

This joint work was made possible thanks to the visiting position funded by the University of Copenhagen, and held by the second author at DIKU in the winter and summer semesters of the academic year 1997-8.

M.H.B.S. & P.U., May 1998

Contents

CHAPTER 1

Type-free λ-calculus

The λ-*calculus* is a collection of formal theories of interest in, e.g., computer science and logic. The λ-calculus and the related systems of *combinatory logic* were originally proposed as a foundation of mathematics around 1930 by Church and Curry, but the proposed systems were subsequently shown to be inconsistent by Church's students Kleene and Rosser in 1935.

However, a certain subsystem consisting of the λ-*terms* equipped with so-called β-*reduction* turned out to be useful for formalizing the intuitive notion of effective computability and led to *Church's thesis* stating that λ-*definability* is an appropriate formalization of the intuitive notion of effective computability. The study of this subsystem—which was proved to be consistent by Church and Rosser in 1936—was a main inspiration for the development of *recursion theory.*

With the invention of physical computers came also programming languages, and λ-calculus has proved to be a useful tool in the design, implementation, and theory of programming languages. For instance, λ-calculus may be considered an idealized sublanguage of some programming languages like *LISP*. Also, λ-calculus is useful for expressing semantics of programming languages as done in *denotational semantics.* According to Hindley and Seldin [55, p.43], "λ-calculus and combinatory logic are regarded as 'test-beds' in the study of higher-order programming languages: techniques are tried out on these two simple languages, developed, and then applied to other more 'practical' languages."

The λ-calculus is sometimes called *type-free* or *untyped* to distinguish it from variants in which *types* play a role; these variants will be introduced in the next chapter.

1.1. λ-terms

The objects of study in λ-calculus are λ-*terms.* In order to introduce these, it is convenient to introduce the notion of a *pre-term.*

1.1.1. Definition. Let

$$V = \{v_0, v_1, \dots\}$$

denote an infinite alphabet. The set Λ^- of *pre-terms* is the set of strings defined by the grammar:

$$\Lambda^- ::= V \mid (\Lambda^- \; \Lambda^-) \mid (\lambda V \, \Lambda^-)$$

1.1.2. Example. The following are pre-terms.

(i) $((v_0 \; v_1) \; v_2) \in \Lambda^-$;

(ii) $(\lambda v_0 \, (v_0 \; v_1)) \in \Lambda^-$;

(iii) $((\lambda v_0 \, v_0) \; v_1) \in \Lambda^-$;

(iv) $((\lambda v_0 \, (v_0 \; v_0)) \; (\lambda v_1 \, (v_1 \; v_1))) \in \Lambda^-$.

1.1.3. Notation. We use uppercase letters, e.g., K, L, M, N, P, Q, R with or without subscripts to denote arbitrary elements of Λ^- and lowercase letters, e.g., x, y, z with or without subscripts to denote arbitrary elements of V.

1.1.4. Terminology.

(i) A pre-term of form x (i.e., an element of V) is called a *variable*;

(ii) A pre-term of form $(\lambda x \, M)$ is called an *abstraction* (over x);

(iii) A pre-term of form $(M \; N)$ is called an *application* (of M to N).

The heavy use of parentheses is rather cumbersome. We therefore introduce the following, standard conventions for omitting parentheses without introducing ambiguity. We shall make use of these conventions under a no-compulsion/no-prohibition agreement—see Remark 1.1.10.

1.1.5. Notation. We use the shorthands

(i) $(K \; L \; M)$ for $((K \; L) \; M)$;

(ii) $(\lambda x \, \lambda y \, M)$ for $(\lambda x \, (\lambda y \, M))$;

(iii) $(\lambda x \, M \; N)$ for $(\lambda x \, (M \; N))$;

(iv) $(M \; \lambda x \, N)$ for $(M \; (\lambda x \, N))$.

We also omit outermost parentheses.

1.1.6. Remark. The two first shorthands concern nested applications and abstractions, respectively. The two next ones concern applications nested inside abstractions and vice versa, respectively.

To remember the shorthands, think of application as associating to the left, and think of abstractions as extending as far to the right as possible.

When abstracting over a number of variables, each variable must be accompanied by an abstraction. It is therefore convenient to introduce the following shorthand.

1.1.7. NOTATION. We write $\lambda x_1 \dots x_n.M$ for $\lambda x_1 \, \dots \lambda x_n \, M$. As a special case, we write $\lambda x.M$ for $\lambda x \, M$.

1.1.8. REMARK. Whereas abstractions are written with a λ, there is no corresponding symbol for applications; these are written simply by juxtaposition. Hence, there is no corresponding shorthand for applications.

1.1.9. EXAMPLE. The pre-terms in Example 1.1.2 can be written as follows, respectively:

(i) $v_0 \; v_1 \; v_2$;

(ii) $\lambda v_0.v_0 \; v_1$;

(iii) $(\lambda v_0.v_0) \; v_1$;

(iv) $(\lambda v_0.v_0 \; v_0) \; \lambda v_1.v_1 \; v_1$.

1.1.10. REMARK. The conventions mentioned above are used in the remainder of these notes. However, we refrain from using them—wholly or partly—when we find this more convenient. For instance, we might prefer to write $(\lambda v_0.v_0 \; v_0) \; (\lambda v_1.v_1 \; v_1)$ for the last term in the above example.

1.1.11. DEFINITION. For $M \in \Lambda^-$ define the set $\mathrm{FV}(M) \subseteq V$ of *free variables* of M as follows.

$$\begin{array}{lcl} \mathrm{FV}(x) & = & \{x\}; \\ \mathrm{FV}(\lambda x.P) & = & \mathrm{FV}(P)\backslash\{x\}; \\ \mathrm{FV}(P \; Q) & = & \mathrm{FV}(P) \cup \mathrm{FV}(Q). \end{array}$$

If $\mathrm{FV}(M) = \{\}$ then M is called *closed.*

1.1.12. EXAMPLE. Let x, y, z denote distinct variables. Then

(i) $\mathrm{FV}(x \; y \; z) = \{x, y, z\}$;

(ii) $\mathrm{FV}(\lambda x.x \; y) = \{y\}$;

(iii) $\mathrm{FV}((\lambda x.x \; x) \; \lambda y.y \; y) = \{\}$.

1.1.13. DEFINITION. For $M, N \in \Lambda^-$ and $x \in V$, the *substitution of* N *for* x *in* M, written $M[x := N] \in \Lambda^-$, is defined as follows, where $x \neq y$:

$$\begin{array}{lll} x[x := N] & = N; & \\ y[x := N] & = y; & \\ (P \; Q)[x := N] & = P[x := N] \; Q[x := N]; & \\ (\lambda x.P)[x := N] & = \lambda x.P; & \\ (\lambda y.P)[x := N] & = \lambda y.P[x := N], & \text{if } y \notin \mathrm{FV}(N) \text{ or } x \notin \mathrm{FV}(P); \\ (\lambda y.P)[x := N] & = \lambda z.P[y := z][x := N], & \text{if } y \in \mathrm{FV}(N) \text{ and } x \in \mathrm{FV}(P). \end{array}$$

where z is chosen as the $v_i \in V$ with minimal i such that $v_i \notin \mathrm{FV}(P) \cup \mathrm{FV}(N)$ in the last clause.

1.1.14. Example. If x, y, z are distinct variables, then for a certain variable u:

$$((\lambda x.x\ yz)\ (\lambda y.x\ y\ z)\ (\lambda z.x\ y\ z))[x := y] = (\lambda x.x\ yz)\ (\lambda u.y\ u\ z)\ (\lambda z.y\ y\ z)$$

1.1.15. Definition. Let *α-equivalence,* written $=_\alpha$, be the smallest relation on Λ^-, such that

$$\begin{array}{ll} P =_\alpha P & \text{for all } P; \\ \lambda x.P =_\alpha \lambda y.P[x := y] & \text{if } y \notin \mathrm{FV}(P), \end{array}$$

and closed under the rules:

$$\begin{array}{llll} P =_\alpha P' & \Rightarrow & \forall x \in V: & \lambda x.P =_\alpha \lambda x.P'; \\ P =_\alpha P' & \Rightarrow & \forall Z \in \Lambda^-: & P\ Z =_\alpha P'\ Z; \\ P =_\alpha P' & \Rightarrow & \forall Z \in \Lambda^-: & Z\ P =_\alpha Z\ P'; \\ P =_\alpha P' & \Rightarrow & P' =_\alpha P; & \\ P =_\alpha P' \ \&\ P' =_\alpha P'' & \Rightarrow & P =_\alpha P''. & \end{array}$$

1.1.16. Example. Let x, y, z denote different variables. Then

(i) $\lambda x.x =_\alpha \lambda y.y$;

(ii) $\lambda x.x\ z =_\alpha \lambda y.y\ z$;

(iii) $\lambda x.\lambda y.x\ y =_\alpha \lambda y.\lambda x.y\ x$;

(iv) $\lambda x.x\ y \neq_\alpha \lambda x.x\ z$.

1.1.17. Definition. Define for any $M \in \Lambda^-$, the *equivalence class* $[M]_\alpha$ by:

$$[M]_\alpha = \{N \in \Lambda^- \mid M =_\alpha N\}$$

Then define the set Λ of *λ-terms* by:

$$\Lambda \quad = \quad \Lambda^- / =_\alpha \quad = \quad \{[M]_\alpha \mid M \in \Lambda^-\}$$

1.1.18. Warning. The notion of a pre-term and the associated explicit distinction between pre-terms and λ-terms introduced above are not standard in the literature. Rather, it is customary to call our pre-terms λ-terms, and then informally remark that α-equivalent λ-terms are "identified."

In the remainder of these notes we shall be almost exclusively concerned with λ-terms, not pre-terms. Therefore, it is convenient to introduce the following.

1.1.19. Notation. We write M instead of $[M]_\alpha$ in the remainder. This leads to ambiguity: is M a pre-term or a λ-term? In the remainder of these notes, M should always be construed as $[M]_\alpha \in \Lambda$, *except when explicitly stated otherwise.*

We end this section with two definitions introducing the notions of free variables and substitution on λ-terms (recall that, so far, these notions have been introduced only for pre-terms). These two definitions provide the first example of how to rigorously understand definitions involving λ-terms.

1.1.20. Definition. For $M \in \Lambda$ define the set $\mathrm{FV}(M) \subseteq V$ of *free variables* of M as follows.

$$\begin{array}{lcl} \mathrm{FV}(x) & = & \{x\}; \\ \mathrm{FV}(\lambda x.P) & = & \mathrm{FV}(P)\backslash\{x\}; \\ \mathrm{FV}(P\,Q) & = & \mathrm{FV}(P) \cup \mathrm{FV}(Q). \end{array}$$

If $\mathrm{FV}(M) = \{\}$ then M is called *closed.*

1.1.21. Remark. According to Notation 1.1.19, what we really mean by this is that we define FV as the map from Λ to subsets of V satisfying the rules:

$$\begin{array}{lcl} \mathrm{FV}([x]_\alpha) & = & \{x\}; \\ \mathrm{FV}([\lambda x.P]_\alpha) & = & \mathrm{FV}([P]_\alpha)\backslash\{x\}; \\ \mathrm{FV}([P\,Q]_\alpha) & = & \mathrm{FV}([P]_\alpha) \cup \mathrm{FV}([Q]_\alpha). \end{array}$$

Strictly speaking we then have to demonstrate there there is at most one such function (uniqueness) and that there is at least one such function (existence).

Uniqueness can be established by showing for any two functions FV_1 and FV_2 satisfying the above equations, and any λ-term, that the results of FV_1 and FV_2 on the λ-term are the same. The proof proceeds by induction on the number of symbols in any member of the equivalence class.

To demonstrate existence, consider the map that, given an equivalence class, picks a member, and takes the free variables of that. Since any choice of member yields the same set of variables, this latter map is well-defined, and can easily be seen to satisfy the above rules.

In the rest of these notes such considerations will be left implicit.

1.1.22. Definition. For $M, N \in \Lambda$ and $x \in V$, the *substitution of N for x in M,* written $M\{x := N\}$, is defined as follows:

$$\begin{array}{lcll} x[x := N] & = & N; & \\ y[x := N] & = & y, & \text{if } x \neq y; \\ (P\,Q)[x := N] & = & P[x := N]\;Q[x := N]; & \\ (\lambda y.P)[x := N] & = & \lambda y.P[x := N], & \text{if } x \neq y, \text{ where } y \notin \mathrm{FV}(N). \end{array}$$

1.1.23. Example.

(i) $(\lambda x.x\,y)[x := \lambda z.z] = \lambda x.x\,y$;

(ii) $(\lambda x.x\,y)[y := \lambda z.z] = \lambda x.x\,\lambda z.z$.

1.2. Reduction

Next we introduce reduction on λ-terms.

1.2.1. DEFINITION. Let $\to_\beta$ be the smallest relation on Λ such that

$$(\lambda x.P)\ Q \to_\beta P[x := Q],$$

and closed under the rules:

$$\begin{array}{lcll} P \to_\beta P' & \Rightarrow & \forall x \in V: & \lambda x.P \to_\beta \lambda x.P' \\ P \to_\beta P' & \Rightarrow & \forall Z \in \Lambda: & P\ Z \to_\beta P'\ Z \\ P \to_\beta P' & \Rightarrow & \forall Z \in \Lambda: & Z\ P \to_\beta Z\ P' \end{array}$$

A term of form $(\lambda x.P)\ Q$ is called a *β-redex,* and $P[x := Q]$ is called its *β-contractum.* A term M is a *β-normal form* if there is no term N with $M \to_\beta N$.

There are other notions of reduction than β-reduction, but these will not be considered in the present chapter. Therefore, we sometimes omit "β-" from the notions *β-redex, β-reduction,* etc.

1.2.2. DEFINITION.

(i) The relation $\twoheadrightarrow_\beta$ (*multi-step β-reduction*) is the transitive-reflexive closure of $\to_\beta$; that is, $\twoheadrightarrow_\beta$ is the smallest relation closed under the rules:

$$\begin{array}{lcl} P \to_\beta P' & \Rightarrow & P \twoheadrightarrow_\beta P'; \\ P \twoheadrightarrow_\beta P' \ \&\ P' \twoheadrightarrow_\beta P'' & \Rightarrow & P \twoheadrightarrow_\beta P''; \\ P \twoheadrightarrow_\beta P. & & \end{array}$$

(ii) The relation $=_\beta$ (*β-equality)* is the transitive-reflexive-symmetric closure of $\to_\beta$; that is, $=_\beta$ is the smallest relation closed under the rules:

$$\begin{array}{lcl} P \to_\beta P' & \Rightarrow & P =_\beta P'; \\ P =_\beta P' \ \&\ P' =_\beta P'' & \Rightarrow & P =_\beta P''; \\ P =_\beta P; & & \\ P =_\beta P' & \Rightarrow & P' =_\beta P. \end{array}$$

1.2.3. WARNING. In these notes, the symbol $=$ without any qualification is used to express the fact that two objects, e.g., pre-terms or λ-terms are identical. This symbol is very often used in the literature for β-equality.

1.2.4. EXAMPLE.

(i) $(\lambda x.x\ x)\ \lambda z.z \to_\beta (x\ x)[x := \lambda z.z] = (\lambda z.z)\ \lambda y.y$;

(ii) $(\lambda z.z)\ \lambda y.y \to_\beta z[z := \lambda y.y] = \lambda y.y$;

(iii) $(\lambda x.x\ x)\ \lambda z.z \twoheadrightarrow_\beta \lambda y.y$;

(iv) $(\lambda x.x)\ y\ z =_\beta y\ ((\lambda x.x)\ z)$.

1.3. Informal interpretation

Informally, λ-terms express functions and applications of functions in a pure form. For instance, the λ-term

$$\mathbf{I} = \lambda x.x$$

intuitively denotes the function that maps any argument to itself, i.e., the identity function. This is similar to the notation $n \mapsto n$ employed in mathematics. However, $\lambda x.x$ is a *string* over an alphabet with symbols λ, x, etc. (or rather an equivalence class of such objects), whereas $n \mapsto n$ is a *function,* i.e., a certain set of pairs. The difference is the same as that between a *program* written in some language and the mathematical function it computes, e.g., addition.

As in the notation $n \mapsto n$, the name of the abstracted variable x in $\lambda x.x$ is not significant, and this is why we identify $\lambda x.x$ with, e.g., $\lambda y.y$.

Another λ-term is

$$\mathbf{K}^* = \lambda y.\lambda x.x$$

which, intuitively, denotes the function that maps any argument to a *function,* namely the one that maps any argument to itself, i.e., the identity function. This is similar to programming languages where a function may return a function as a result. A related λ-term is

$$\mathbf{K} = \lambda y.\lambda x.y$$

which, intuitively, denotes the function that maps any argument to the function that, for any argument, returns the former argument.

Since λ-terms intuitively denote functions, there is a way to invoke one λ-term on another; this is expressed by application. Thus, the λ-term

$$\mathbf{I}\,\mathbf{K}$$

expresses application of $\mathbf{I}$ to $\mathbf{K}$. Since $\mathbf{K}$ intuitively denotes a function too, $\mathbf{I}$ denotes a function which may have another function as argument. This is similar to programming languages where a procedure may receive another procedure as argument.

In mathematics we usually write application of a function, say $f(n) = n^2$, to an argument, say 4, with the argument in parentheses: $f(4)$. In the λ-calculus we would rather write this as $(f\ 4)$, or just $f\,4$, keeping Notation 1.1.5 in mind. Not all parentheses can be omitted, though; for instance,

$$(\lambda x.x)\,\mathbf{I} \qquad \lambda x.x\,\mathbf{I}$$

are not the same λ-term; the first is $\mathbf{I}$ applied to $\mathbf{I}$, whereas the second expects an argument x which is applied to $\mathbf{I}$.

Intuitively, if $\lambda x.M$ denotes a function, and N denotes an argument, then the the value of the function on the argument is denoted by the λ-term that arises by substituting N for x in M. This latter λ-term is exactly the term

$$M[x := N]$$

This is similar to common practice in mathematics; if f is as above, then $f(4) = 4^2$, and we get from the application $f(4)$ to the value 4^2 by substituting 4 for n in the body of the definition of f.

The process of calculating values is formalized by β-reduction. Indeed, $M \to_\beta N$ if N arises from M by replacing a β-redex, i.e., a part of form

$$(\lambda x.P)\ Q$$

by its β-contractum.

$$P[x := Q]$$

For instance,

$$\mathbf{I}\ \mathbf{K} = (\lambda x.x)\ \mathbf{K} \to_\beta x[x := \mathbf{K}] = \mathbf{K}$$

Then the relation $\twoheadrightarrow_\beta$ formalizes the process of computing the overall result. Also, $=_\beta$ identifies λ-terms that, intuitively, denote the same function.

Note that λ-calculus is a *type-free* formalism. Unlike common mathematical practice, we do not insist that λ-terms denote functions from certain domains, e.g., the natural numbers, and that arguments be drawn from these domains. In particular, we may have self-application as in the λ-term

$$\omega = \lambda x.x\ x$$

and we may apply *this* λ-term to *itself* as in the λ-term

$$\Omega = \omega\ \omega$$

The type-free nature of λ-calculus leads to some interesting phenomena; for instance, a λ-term may reduce to itself as in

$$\Omega = (\lambda x.x\ x)\ \omega \to_\beta \omega\ \omega = \Omega$$

Therefore, there are also λ-terms with infinite reduction sequences, like

$$\Omega \to_\beta \Omega \to_\beta \dots$$

1.4. The Church-Rosser Theorem

Since a λ-term M may contain several β-redexes, i.e., several parts of form $(\lambda x.P)\ Q$, there may be several N such that $M \to_\beta N$. For instance,

$$\mathbf{K}\ (\mathbf{I}\ \mathbf{I}) \to_\beta \lambda x.(\mathbf{I}\ \mathbf{I})$$

and also

$$\mathbf{K}\,(\mathbf{I}\,\mathbf{I}) \to_\beta \mathbf{K}\,\mathbf{I}$$

However, the *Church-Rosser theorem,* proved below, states that if

$$M \twoheadrightarrow_\beta M_1$$

and

$$M \twoheadrightarrow_\beta M_2$$

then a single λ-term M_3 can be found with

$$M_1 \twoheadrightarrow_\beta M_3$$

and

$$M_2 \twoheadrightarrow_\beta M_3$$

In particular, if M_1 and M_2 are β-normal forms, i.e., λ-terms that admit no further β-reductions, then they must be the same λ-term, since the β-reductions from M_1 and M_2 to M_3 must be in zero steps. This is similar to the fact that when we calculate the value of an arithmetical expression, e.g.,

$$(4+2)\cdot(3+7)\cdot 11$$

the end result is independent of the order in which we do the calculations.

1.4.1. Definition. A relation $>$ on Λ satisfies the *diamond property* if, for all $M_1, M_2, M_3 \in \Lambda$, if $M_1 > M_2$ and $M_1 > M_3$, then there exists an $M_4 \in \Lambda$ such that $M_2 > M_4$ and $M_3 > M_4$.

1.4.2. Lemma. *Let $>$ be a relation on Λ and suppose that its transitive closure*[1] *is $\twoheadrightarrow_\beta$. If $>$ satisfies the diamond property, then so does $\twoheadrightarrow_\beta$.*

Proof. First show by induction on n that $M_1 > N_1$ and $M_1 > \ldots > M_n$ implies that there are $N_2, \ldots, N_n$ such that $N_1 > N_2 > \ldots > N_n$ and $M_n > N_n$.

Using this property, show by induction on m that if $N_1 > \ldots > N_m$ and $N_1 >^* M_1$ then there are $M_2, \ldots, M_m$ such that $M_1 > M_2 > \ldots > M_m$ and $N_m >^* M_m$.

[1]Let R be a relation on Λ. The *transitive closure* of R is the least relation R^* satisfying:

$$\begin{array}{lll} PRP' & \Rightarrow & PR^*P' \\ PR^*P' \ \&\ P'R^*P'' & \Rightarrow & PR^*P'' \end{array}$$

The *reflexive closure* of R is the least relation $R^=$ satisfying:

$$\begin{array}{lll} PRP' & \Rightarrow & PR^=P' \\ PR^=P & & \end{array}$$

Now assume $M_1 \twoheadrightarrow_\beta M_2$ and $M_1 \twoheadrightarrow_\beta M_3$. Since $\twoheadrightarrow_\beta$ is the transitive closure of $>$ we have $M_1 > \ldots > M_2$ and $M_1 > \ldots > M_3$. By what was shown above, we can find M_4 such that $M_2 > \ldots > M_4$ and $M_3 > \ldots > M_4$. Since $\twoheadrightarrow_\beta$ is the transitive closure of $>$, also $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$. □

1.4.3. Definition. Let $\twoheadrightarrow_l$ be the relation on Λ defined by:

$$\begin{array}{lcl} P \twoheadrightarrow_l P & & \\ P \twoheadrightarrow_l P' & \Rightarrow & \lambda x.P \twoheadrightarrow_l \lambda x.P' \\ P \twoheadrightarrow_l P' \ \&\ Q \twoheadrightarrow_l Q' & \Rightarrow & P\,Q \twoheadrightarrow_l P'\,Q' \\ P \twoheadrightarrow_l P' \ \&\ Q \twoheadrightarrow_l Q' & \Rightarrow & (\lambda x.P)\,Q \twoheadrightarrow_l P'[x := Q'] \end{array}$$

1.4.4. Lemma. $M \twoheadrightarrow_l M' \ \&\ N \twoheadrightarrow_l N' \Rightarrow M[x := N] \twoheadrightarrow_l M'[x := N']$.

Proof. By induction on the definition of $M \twoheadrightarrow_l M'$. In case M' is M, proceed by induction on M. □

1.4.5. Lemma. *$\twoheadrightarrow_l$ satisfies the diamond property, i.e., for all $M_1, M_2, M_3 \in \Lambda$, if $M_1 \twoheadrightarrow_l M_2$ and $M_1 \twoheadrightarrow_l M_3$, then there exists an $M_4 \in \Lambda$ such that $M_2 \twoheadrightarrow_l M_4$ and $M_3 \twoheadrightarrow_l M_4$.*

Proof. By induction on the definition of $M_1 \twoheadrightarrow_l M_2$, using the above lemma. □

1.4.6. Lemma. *$\twoheadrightarrow_\beta$ is the transitive closure of $\twoheadrightarrow_l$.*

Proof. Clearly[2]

$$(\rightarrow_\beta)^= \ \subseteq \twoheadrightarrow_l \subseteq \twoheadrightarrow_\beta$$

Then

$$\twoheadrightarrow_\beta = \ ((\rightarrow_\beta)^=)^* \ \subseteq \twoheadrightarrow_l^* \subseteq \ (\twoheadrightarrow_\beta)^* \ = \twoheadrightarrow_\beta$$

In particular, $\twoheadrightarrow_l^* = \twoheadrightarrow_\beta$. □

1.4.7. Theorem (Church and Rosser, 1936). *For every $M_1, M_2, M_3 \in \Lambda$, if $M_1 \twoheadrightarrow_\beta M_2$ and $M_1 \twoheadrightarrow_\beta M_3$, then there exists an $M_4 \in \Lambda$ such that $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$.*

Proof (Tait & Martin-Löf). By the above three lemmas. □

1.4.8. Corollary. *For all $M, N \in \Lambda$, if $M =_\beta N$, then there exists an $L \in \Lambda$ such that $M \twoheadrightarrow_\beta L$ and $N \twoheadrightarrow_\beta L$.*

1.4.9. Corollary. *For all $M, N_1, N_2 \in \Lambda$, if $M \twoheadrightarrow_\beta N_1$ and $M \twoheadrightarrow_\beta N_2$ and both N_1 and N_2 are in β-normal form, then $N_1 = N_2$.*

[2]Recall the relations R^* and $R^=$ defined earlier.

1.4.10. Corollary. *For all $M, N \in \Lambda$, if there are β-normal forms L_1 and L_2 such that $M \twoheadrightarrow_\beta L_1$, $N \twoheadrightarrow_\beta L_2$, and $L_1 \neq L_2$, then $M \neq_\beta N$.*

1.4.11. Example. $\lambda x.x \neq_\beta \lambda x.\lambda y.x$.

1.4.12. Remark. One can consider the lambda calculus as an equational theory, i.e., a formal theory with formulas of the form $M =_\beta N$. The preceding example establishes *consistency* of this theory, in the following sense: there exists a formula P which cannot be proved.

This may seem to be a very weak property, compared to "one cannot prove a contradiction" (where a suitable notion of "contradiction" in ordinary logic is e.g., $P \wedge \neg P$). But note that in most formal theories, where a notion of contradiction can be expressed, its provability implies provability of all formulas. Thus, consistency can be equally well defined as "one cannot prove everything".

1.5. Expressibility and undecidability

Although we have given an informal explanation of the meaning of λ-terms it remains to explain in what sense β-reduction more precisely can express computation. In this section we show that λ-calculus can be seen as an alternative formulation of recursion theory.

The following gives a way of representing numbers as λ-terms.

1.5.1. Definition.

(i) For any $n \in \mathbb{N}$ and $F, A \in \Lambda$ define $F^n(A)$ (*n-times iterated application of F to A*) by:

$$\begin{array}{lcl} F^0(A) & = & A \\ F^{n+1}(A) & = & F(F^n(A)) \end{array}$$

(ii) For any $n \in \mathbb{N}$, the *Church numeral* c_n is the λ-term

$$c_n = \lambda s.\lambda z.s^n(z)$$

1.5.2. Example.

(i) $c_0 = \lambda s.\lambda z.z$;

(ii) $c_1 = \lambda s.\lambda z.s\ z$;

(iii) $c_2 = \lambda s.\lambda z.s\ (s\ z)$;

(iv) $c_3 = \lambda s.\lambda z.s\ (s\ (s\ z))$.

1.5.3. Remark. c_n is the number n represented inside the λ-calculus.

The following shows how to do arithmetic on Church numerals.

1.5.4. Proposition (Rosser). *Let*

$$\begin{array}{lcl} A_+ & = & \lambda x.\lambda y.\lambda s.\lambda z.x\ s\ (y\ s\ z); \\ A_* & = & \lambda x.\lambda y.\lambda s.x\ (y\ s); \\ A_e & = & \lambda x.\lambda y.y\ x. \end{array}$$

Then

$$\begin{array}{lcl} A_+\ c_n\ c_m & = & c_{n+m}; \\ A_*\ c_n\ c_m & = & c_{n\cdot m}; \\ A_e\ c_n\ c_m & = & c_{n^m}\ \textit{if}\ m > 0. \end{array}$$

Proof. For any $n \in \mathbb{N}$,

$$\begin{array}{lcl} c_n\ s\ z & = & (\lambda f.\lambda x.f^n(x))\ s\ z \\ & =_\beta & (\lambda x.s^n(x))\ z \\ & =_\beta & s^n(z) \end{array}$$

Thus

$$\begin{array}{lcl} A_+\ c_n\ c_m & = & (\lambda x.\lambda y.\lambda s.\lambda z.x\ s\ (y\ s\ z))\ c_n\ c_m \\ & =_\beta & \lambda s.\lambda z.c_n\ s\ (c_m\ s\ z) \\ & =_\beta & \lambda s.\lambda z.c_n\ s\ (s^m(z)) \\ & =_\beta & \lambda s.\lambda z.s^n(s^m(z)) \\ & = & \lambda s.\lambda z.s^{n+m}(z) \\ & = & c_{n+m} \end{array}$$

The similar properties for multiplication and exponentiation are left as exercises. □

1.5.5. Remark. Recall that $M =_\beta N$ when, intuitively, M and N denote the same object. For instance $\mathbf{I}\,\mathbf{I} =_\beta \mathbf{I}$ since both terms, intuitively, denote the identity function.

Now consider the two terms

$$\begin{array}{lcl} A_s & = & \lambda x.\lambda s.\lambda z.s\ (x\ s\ z) \\ A'_s & = & \lambda x.\lambda s.\lambda z.x\ s\ (s\ z) \end{array}$$

It is easy to calculate that

$$\begin{array}{lcl} A_s\ c_n & =_\beta & c_{n+1} \\ A'_s\ c_n & =_\beta & c_{n+1} \end{array}$$

So both terms denote, informally, the successor function on Church numerals, but the two terms are not β-equal (why not?)

The following shows how to define booleans and conditionals inside λ-calculus.

1.5.6. Proposition. *Define*

$$\begin{aligned} \mathbf{true} &= \lambda x.\lambda y.x; \\ \mathbf{false} &= \lambda x.\lambda y.y; \\ \mathbf{if}\ B\ \mathbf{then}\ P\ \mathbf{else}\ Q &= B\,P\,Q. \end{aligned}$$

Then

$$\begin{aligned} \mathbf{if\ true\ then}\ P\ \mathbf{else}\ Q &=_\beta P; \\ \mathbf{if\ false\ then}\ P\ \mathbf{else}\ Q &=_\beta Q. \end{aligned}$$

Proof. We have:

$$\begin{aligned} \mathbf{if\ true\ then}\ P\ \mathbf{else}\ Q &= (\lambda x.\lambda y.x)\,P\,Q \\ &=_\beta (\lambda y.P)\,Q \\ &=_\beta P. \end{aligned}$$

The proof that **if false then** P **else** $Q =_\beta Q$ is similar. □

We can also define pairs in λ-calculus.

1.5.7. Proposition. *Define*

$$\begin{aligned} [P,Q] &= \lambda x.x\,P\,Q; \\ \pi_1 &= \lambda x.\lambda y.x; \\ \pi_2 &= \lambda x.\lambda y.y. \end{aligned}$$

Then

$$\begin{aligned} [P,Q]\,\pi_1 &=_\beta P; \\ [P,Q]\,\pi_2 &=_\beta Q. \end{aligned}$$

Proof. We have:

$$\begin{aligned} [P,Q]\,\pi_1 &= (\lambda x.x\,P\,Q)\,\lambda x.\lambda y.x \\ &=_\beta (\lambda x.\lambda y.x)\,P\,Q \\ &=_\beta (\lambda y.P)\,Q \\ &=_\beta P. \end{aligned}$$

The proof that $[P,Q]\,\pi_2 =_\beta Q$ is similar. □

1.5.8. Remark. Note that we do not have $[M\ \pi_1, M\ \pi_2] =_\beta M$ for all $M \in \Lambda$; that is, our pairing operator is not *surjective.*

1.5.9. Remark. The construction is easily generalized to tuples $[M_1, \ldots, M_n]$ with projections π_i where $i \in \{1, \ldots, n\}$.

The following gives one way of expressing recursion in λ-calculus.

1.5.10. THEOREM (Fixed point theorem). *For all F there is an X such that*

$$F\ X =_\beta X$$

In fact, there is a λ-term $\mathbf{Y}$ such that, for all F:

$$F\ (\mathbf{Y}\ F) =_\beta \mathbf{Y}\ F$$

PROOF. Put

$$\mathbf{Y} = \lambda f.(\lambda x.f\ (x\ x))\ \lambda x.f\ (x\ x)$$

Then

$$\begin{array}{rcl} \mathbf{Y}\ F & = & (\lambda f.(\lambda x.f\ (x\ x))\ \lambda x.f\ (x\ x))\ F \\ & =_\beta & (\lambda x.F\ (x\ x))\ \lambda x.F\ (x\ x) \\ & =_\beta & F\ ((\lambda x.F\ (x\ x))\ \lambda x.F\ (x\ x)) \\ & =_\beta & F\ ((\lambda f.(\lambda x.f\ (x\ x))\ \lambda x.f\ (x\ x))\ F) \\ & = & F\ (\mathbf{Y}\ F) \end{array}$$

as required. □

1.5.11. COROLLARY. *Given $M \in \Lambda$ there is $F \in \Lambda$ such that:*

$$F =_\beta M[f := F]$$

PROOF. Put

$$F = \mathbf{Y}\ \lambda f.M$$

Then

$$\begin{array}{rcl} F & = & \mathbf{Y}\ \lambda f.M \\ & =_\beta & (\lambda f.M)\ (\mathbf{Y}\ \lambda f.M) \\ & = & (\lambda f.M)\ F \\ & =_\beta & M[f := F] \end{array}$$

as required. □

Corollary 1.5.11 allows us to write *recursive definitions* of λ-terms; that is, we may define F as a λ-term satisfying a *fixed point equation* $F =_\beta \lambda x.M$ where the term F occurs somewhere inside M. However, there may be several terms F satisfying this equation (will these be β-equal?).

1.5.12. EXAMPLE. Let C be some λ-term which expresses a condition, i.e., let $C\ c_n =_\beta$ **true** or $C\ c_n =_\beta$ **false**, for all $n \in \mathbb{N}$. Let S define the successor function (see Remark 1.5.5). Suppose we want to compute in λ-calculus, for any number, the smallest number greater than the given one that satisfies the condition. This is expressed by the λ-term F:

$$\begin{array}{rcl} H & = & \lambda f.\lambda x.\mathbf{if}\ (C\ x)\ \mathbf{then}\ x\ \mathbf{else}\ f\ (S\ x) \\ F & = & \mathbf{Y}\ H \end{array}$$

Indeed, for example

$$\begin{array}{lll} F\ c_4 & = & (\mathbf{Y}\ H)\ c_4 \\ & =_\beta & H\ (\mathbf{Y}\ H)\ c_4 \\ & = & (\lambda f.\lambda x.\mathbf{if}\ (C\ x)\ \mathbf{then}\ x\ \mathbf{else}\ f\ (S\ x))\ (\mathbf{Y}\ H)\ c_4 \\ & =_\beta & \mathbf{if}\ (C\ c_4)\ \mathbf{then}\ c_4\ \mathbf{else}\ (\mathbf{Y}\ H)\ (S\ c_4) \\ & = & \mathbf{if}\ (C\ c_4)\ \mathbf{then}\ c_4\ \mathbf{else}\ F\ (S\ c_4) \end{array}$$

So far we have been informal as to how λ-terms "express" certain functions. This notion is made precise as follows.

1.5.13. DEFINITION.

(i) A *numeric function* is a map

$$f : \mathbb{N}^m \to \mathbb{N}.$$

(ii) A numeric function $f : \mathbb{N}^m \to \mathbb{N}$ is λ-*definable* if there is an $F \in \Lambda$ such that

$$F\ c_{n_1}\ \ldots\ c_{n_m} =_\beta c_{f(n_1,\ldots,n_m)}$$

for all $n_1, \ldots, n_m \in \mathbb{N}$.

1.5.14. REMARK. By the Church-Rosser property, (ii) implies that, in fact,

$$F\ c_{n_1}\ \ldots\ c_{n_m} \twoheadrightarrow_\beta c_{f(n_1,\ldots,n_m)}$$

There are similar notions for partial functions—see [7].

We shall show that all recursive functions are λ-definable.

1.5.15. DEFINITION. The class of *recursive functions* is the smallest class of numeric functions containing the *initial functions*

(i) *projections:* $U_i^m(n_1, \ldots, n_m) = n_i$ for all $1 \le i \le m$;

(ii) *successor:* $S^+(n) = n + 1$;

(iii) *zero:* $Z(n) = 0$.

and closed under *composition, primitive recursion,* and *minimization:*

(i) *composition:* if $g : \mathbb{N}^k \to \mathbb{N}$ and $h_1, \ldots, h_k : \mathbb{N}^m \to \mathbb{N}$ are recursive, then so is $f : \mathbb{N}^m \to \mathbb{N}$ defined by

$$f(n_1, \ldots, n_m) = g(h_1(n_1, \ldots, n_m), \ldots, h_k(n_1, \ldots, n_m)).$$

(ii) *primitive recursion:* if $g : \mathbb{N}^m \to \mathbb{N}$ and $h : \mathbb{N}^{m+2} \to \mathbb{N}$ are recursive, then so is $f : \mathbb{N}^{m+1} \to \mathbb{N}$ defined by

$$\begin{array}{lll} f(0, n_1, \ldots, n_m) & = & g(n_1, \ldots, n_m); \\ f(n+1, n_1, \ldots, n_m) & = & h(f(n, n_1, \ldots, n_m), n, n_1, \ldots, n_m). \end{array}$$

(iii) *minimization:* if $g : \mathbb{N}^{m+1} \to \mathbb{N}$ is recursive and for all $n_1, \dots, n_m$ there is an n such that $g(n, n_1, \dots, n_m) = 0$, then $f : \mathbb{N}^m \to \mathbb{N}$ defined as follows is also recursive[3]

$$f(n_1, \dots, n_m) = \mu n.g(n, n_1, \dots, n_m) = 0$$

1.5.16. Lemma. *The initial functions are λ-definable.*

Proof. With

$$\begin{array}{rcl} \mathbf{U}_i^m & = & \lambda x_1 \dots \lambda x_m.x_i \\ \mathbf{S}^+ & = & \lambda x.\lambda s.\lambda z.s\ (x\ s\ z) \\ \mathbf{Z} & = & \lambda x.c_0 \end{array}$$

the necessary properties hold. □

1.5.17. Lemma. *The λ-definable functions are closed under composition.*

Proof. If $g : \mathbb{N}^k \to \mathbb{N}$ is λ-definable by $G \in \Lambda$ and $h_1, \dots, h_k : \mathbb{N}^m \to \mathbb{N}$ are λ-definable by some $H_1, \dots, H_k \in \Lambda$, then $f : \mathbb{N}^m \to \mathbb{N}$ defined by

$$f(n_1, \dots, n_m) = g(h_1(n_1, \dots, n_m), \dots, h_k(n_1, \dots, n_m))$$

is λ-definable by

$$F = \lambda x_1 \dots \lambda x_m.G\ (H_1\ x_1 \dots x_m) \dots (H_k\ x_1 \dots x_m),$$

as is easy to verify. □

1.5.18. Lemma. *The λ-definable functions are closed under primitive recursion.*

Proof. If $g : \mathbb{N}^m \to \mathbb{N}$ is λ-definable by some $G \in \Lambda$ and $h : \mathbb{N}^{m+2} \to \mathbb{N}$ is λ-definable by some $H \in \Lambda$, then $f : \mathbb{N}^{m+1} \to \mathbb{N}$ defined by

$$\begin{array}{rcl} f(0, n_1, \dots, n_m) & = & g(n_1, \dots, n_m); \\ f(n+1, n_1, \dots, n_m) & = & h(f(n, n_1, \dots, n_m), n, n_1, \dots, n_m), \end{array}$$

is λ-definable by $F \in \Lambda$ where

$$\begin{array}{rcl} F & = & \lambda x.\lambda x_1 \dots \lambda x_m.x\ T\ [c_0, G\ x_1 \dots x_n]\ \pi_2; \\ T & = & \lambda p.[\mathbf{S}^+\ (p\ \pi_1), H\ (p\ \pi_2)\ (p\ \pi_1)\ x_1 \dots x_m]. \end{array}$$

Indeed, we have

$$\begin{array}{rcl} F\ c_n\ c_{n_1} \dots c_{n_m} & =_\beta & c_n\ T\ [c_0, G\ c_{n_1} \dots c_{n_m}]\ \pi_2 \\ & =_\beta & T^n([c_0, G\ c_{n_1} \dots c_{n_m}])\ \pi_2 \end{array}$$

[3] $\mu n.g(n, n_1, \dots, n_m) = 0$ denotes the smallest number n satisfying the equation $g(n, n_1, \dots, n_m) = 0$.

Also,

$$\begin{array}{rcl} T\,[c_n, c_{f(n,n_1,\dots,n_m)}] & =_\beta & [\mathbf{S}^+(c_n), H\,c_{f(n,n_1,\dots,n_m)}\,c_n\,c_{n_1}\dots c_{n_m}] \\ & =_\beta & [c_{n+1}, c_{h(f(n,n_1,\dots,n_m),n,n_1,\dots,n_m)}] \\ & =_\beta & [c_{n+1}, c_{f(n+1,n_1,\dots,n_m)}] \end{array}$$

So

$$T^n([c_0, G\,c_{n_1}\dots c_{n_m}]) \quad =_\beta \quad [c_n, c_{f(n,n_1,\dots,n_m)}]$$

From this the required property follows. □

1.5.19. Lemma. *The λ-definable functions are closed under minimization.*

Proof. If $g : \mathbb{N}^{m+1} \to \mathbb{N}$ is λ-definable by $G \in \Lambda$ and for all $n_1, \dots, n_m$ there is an n, such that $g(n, n_1, \dots, n_m) = 0$, then $f : \mathbb{N}^m \to \mathbb{N}$ defined by

$$f(n_1, \dots, n_m) = \mu m.g(n, n_1, \dots, n_m) = 0$$

is λ-definable by $F \in \Lambda$, where

$$F = \lambda x_1. \dots \lambda x_m.H\;c_0$$

and where $H \in \Lambda$ is such that

$$H =_\beta \lambda y.\textbf{if}\ (\textbf{zero?}\ (G\,x_1\dots x_m\,y))\ \textbf{then}\ y\ \textbf{else}\ H\ (\mathbf{S}^+\ y).$$

Here,

$$\textbf{zero?} = \lambda x.x\ (\lambda y.\textbf{false})\ \textbf{true}$$

We leave it as an exercise to verify that the required properties hold. □

The following can be seen as a form of *completeness* of the λ-calculus.

1.5.20. Theorem (Kleene). *All recursive functions are λ-definable.*

Proof. By the above lemmas. □

The converse also holds, as one can show by a routine argument. Similar results hold for partial functions as well—see [7].

1.5.21. Definition. Let $\langle \bullet, \bullet \rangle : \mathbb{N}^2 \to \mathbb{N}$ be a bijective, recursive function. The map $\# : \Lambda^- \to \mathbb{N}$ is defined by:

$$\begin{array}{rcl} \#(v_i) & = & \langle 0, i \rangle \\ \#(\lambda x.M) & = & \langle 2, \langle \#(x), \#(M) \rangle \rangle \\ \#(M\ N) & = & \langle 3, \langle \#(M), \#(N) \rangle \rangle \end{array}$$

For $M \in \Lambda$, we take $\#(M)$ to be the least possible number $\#(M')$ where M' is an alpha-representative of M. Also, for $M \in \Lambda$, we define $\ulcorner M \urcorner = c_{\#(M)}$.

1.5.22. Definition. Let $A \subseteq \Lambda$.

(i) A is *closed under* $=_\beta$ if

$$M \in A \ \& \ M =_\beta N \Rightarrow N \in A$$

(ii) A is *non-trival* if

$$A \neq \emptyset \ \& \ A \neq \Lambda$$

(iii) A is *recursive* if

$$\#A = \{\#(M) \mid M \in A\}$$

is recursive.

1.5.23. Theorem (Curry, Scott). *Let A be non-trivial and closed under $=_\beta$. Then A is not recursive.*

Proof (J. Terlouw). Suppose A is recursive. Define

$$B = \{M \mid M \ulcorner M \urcorner \in A\}$$

There exists an $F \in \Lambda$ with

$$\begin{array}{lcl} M \in B & \Leftrightarrow & F \ulcorner M \urcorner =_\beta c_0; \\ M \notin B & \Leftrightarrow & F \ulcorner M \urcorner =_\beta c_1. \end{array}$$

Let $M_0 \in A$, $M_1 \in \Lambda \backslash A$, and let

$$G = \lambda x.\mathbf{if}\ (\mathbf{zero?}\ (F\ x))\ \mathbf{then}\ M_1\ \mathbf{else}\ M_0$$

Then

$$\begin{array}{lcl} M \in B & \Leftrightarrow & G \ulcorner M \urcorner =_\beta M_1 \\ M \notin B & \Leftrightarrow & G \ulcorner M \urcorner =_\beta M_0 \end{array}$$

so

$$\begin{array}{lcccccc} G \in B & \Leftrightarrow & G \ulcorner G \urcorner =_\beta M_1 & \Rightarrow & G \ulcorner G \urcorner \notin A & \Rightarrow & G \notin B \\ G \notin B & \Leftrightarrow & G \ulcorner G \urcorner =_\beta M_0 & \Rightarrow & G \ulcorner G \urcorner \in A & \Rightarrow & G \in B \end{array}$$

a contradiction. □

1.5.24. Remark. The above theorem is analogous to *Rice's theorem* known in recursion theory.

The following is a variant of the halting problem. Informally it states that the formal theory of β-equality mentioned in Remark 1.4.12 is undecidable.

1.5.25. Corollary (Church). $\{M \in \Lambda \mid M =_\beta \mathbf{true}\}$ *is not recursive.*

1.5.26. Corollary. *The following set is not recursive:*

$$\{M \in \Lambda \mid \exists N \in \Lambda : M \twoheadrightarrow_\beta N \ \&\ N \textit{ is a } \beta\textit{-normal form }\}.$$

One can also infer from these results the well-known theorem due to Church stating that first-order predicate calculus is undecidable.

1.6. Historical remarks

For more on the history of λ-calculus, see e.g., [55] or [7]. First hand information may be obtained from Rosser and Kleene's eye witness statements [94, 62], and from Curry and Feys' book [24] which contains a wealth of historical information. Curry and Church's original aims have recently become the subject of renewed attention—see, e.g., [9, 10] and [50].

1.7. Exercises

1.7.1. Exercise. Show, step by step, how application of the conventions in Notation 1.1.5 allows us to express the pre-terms in Example 1.1.2 as done in Example 1.1.9.

1.7.2. Exercise. Which of the following abbreviations are correct?

1. $\lambda x.x\ y = (\lambda x.x)\ y$;
2. $\lambda x.x\ y = \lambda x.(x\ y)$;
3. $\lambda x.\lambda y.\lambda z.x\ y\ z = (\lambda x.\lambda y.\lambda z.x)\ (y\ z)$;
4. $\lambda x.\lambda y.\lambda z.x\ y\ z = ((\lambda x.\lambda y.\lambda z.x)\ y)\ z$;
5. $\lambda x.\lambda y.\lambda z.x\ y\ z = \lambda x.\lambda y.\lambda z.((x\ y)\ z)$.

1.7.3. Exercise. Which of the following identifications are correct?

1. $\lambda x.\lambda y.x = \lambda y.\lambda x.y$;
2. $(\lambda x.x)\ z = (\lambda z.z)\ x$.

1.7.4. Exercise. Do the following terms have normal forms?

1. $\mathbf{I}$, where $\lambda x.x$;
2. Ω, i.e., $\omega\ \omega$, where $\omega = \lambda x.x\ x$;
3. $\mathbf{K}\,\mathbf{I}\,\Omega$ where $\mathbf{K} = \lambda x.\lambda y.x$;
4. $(\lambda x.\mathbf{K}\,\mathbf{I}\,(x\ x))\ \lambda y.\mathbf{K}\,\mathbf{I}\,(y\ y)$;
5. $(\lambda x.z\ (x\ x))\ \lambda y.z\ (y\ y)$.

1.7.5. Exercise. A *reduction path* from a λ-term M is a finite or infinite sequence

$$M \to_\beta M_1 \to_\beta M_2 \to_\beta \dots$$

A term that has a normal form is also called *weakly normalizing* (or just *normalizing*), since at least one of its reduction paths terminate in a normal form. A term is *strongly normalizing* if *all* its reduction paths eventually terminate in normal forms, i.e., if the term has no infinite reduction paths. Which of the five terms in the preceding exercise are weakly/strongly normalizing? In which cases do different reduction paths lead to different normal forms?

1.7.6. Exercise. Which of the following are true?

1. $(\lambda x.\lambda y.\lambda z.(x\ z)\ (y\ z))\ \lambda u.u =_\beta (\lambda v.v\ \lambda y.\lambda z.\lambda u.u)\ \lambda x.x$;
2. $(\lambda x.\lambda y.x\ \lambda z.z)\ \lambda a.a =_\beta (\lambda y.y)\ \lambda b.\lambda z.z$;
3. $\lambda x.\Omega =_\beta \Omega$.

1.7.7. Exercise. Prove (without using the Church-Rosser Theorem) that for all $M_1, M_2, M_3 \in \Lambda$, if $M_1 \to_\beta M_2$ and $M_1 \to_\beta M_3$, then there exists an $M_4 \in \Lambda$ such that $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$.

Can you extend your proof technique to yield a proof of the Church-Rosser theorem?

1.7.8. Exercise. Fill in the details of the proof Lemma 1.4.4.

1.7.9. Exercise. Fill in the details of the proof Lemma 1.4.5.

1.7.10. Exercise. Which of the following are true?

1. $(\mathbf{I}\,\mathbf{I})\ (\mathbf{I}\,\mathbf{I}) \twoheadrightarrow_l \mathbf{I}\,\mathbf{I}$;
2. $(\mathbf{I}\,\mathbf{I})\ (\mathbf{I}\,\mathbf{I}) \twoheadrightarrow_l \mathbf{I}$;
3. $\mathbf{I}\,\mathbf{I}\,\mathbf{I}\,\mathbf{I} \twoheadrightarrow_l \mathbf{I}\,\mathbf{I}\,\mathbf{I}$;
4. $\mathbf{I}\,\mathbf{I}\,\mathbf{I}\,\mathbf{I} \twoheadrightarrow_l \mathbf{I}$;

1.7.11. Exercise. Show that the fourth clause in Definition 1.4.3 cannot be replaced by

$$(\lambda x.P)\ Q \twoheadrightarrow_l P[x := Q].$$

That is, show that, if this is done, then $\twoheadrightarrow_l$ does not satisfy the diamond property.

1.7.12. Exercise. Prove Corollary 1.4.8–1.4.10.

1.7.13. Exercise. Write λ-terms (without using the notation $s^n(z)$) whose β-normal forms are the Church numerals c_5 and c_{100}.

1.7.14. Exercise. Prove that A_* and A_e satisfy the equations stated in Proposition 1.5.4.

1.7.15. Exercise. For each $n \in \mathbb{N}$, write a λ-term B_n such that

$$B_n\, c_i\, Q_1 \ldots Q_n =_\beta Q_i,$$

for all $Q_1, \ldots Q_n \in \Lambda$.

1.7.16. Exercise. For each $n \in \mathbb{N}$, write λ-terms $P_n, \pi_1, \ldots, \pi_n$, such that for all $Q_1, \ldots, Q_n \in \Lambda$:

$$(P_n\, Q_1 \ldots Q_n)\; \pi_i =_\beta Q_i.$$

1.7.17. Exercise (Klop, taken from [7]). Let $\lambda x_1 x_2 \ldots x_n.M$ be an abbreviation for $\lambda x_1.\lambda x_2.\ldots\lambda x_n.M$. Let

$$\begin{array}{lcl} ? & = & \lambda abcdefghijklmnopqstuvwxyzr.r\ (thisisafixedpointcombinator); \\ \$ & = & ??????????????????????????. \end{array}$$

Show that \$ is a *fixed point combinator,* i.e., that $\$\ F =_\beta F\ (\$\ F)$, holds for all $F \in \Lambda$.

1.7.18. Exercise. Define a λ-term **neg** such that

$$\begin{array}{lcl} \textbf{neg true} & =_\beta & \textbf{false}; \\ \textbf{neg false} & =_\beta & \textbf{true}. \end{array}$$

1.7.19. Exercise. Define λ-terms O and E such that, for all $n \in \mathbb{N}$:

$$Oc_m =_\beta \begin{cases} \textbf{true} & \text{if } m \text{ is odd;} \\ \textbf{false} & \text{otherwise,} \end{cases}$$

and

$$Ec_m =_\beta \begin{cases} \textbf{true} & \text{if } m \text{ is even;} \\ \textbf{false} & \text{otherwise.} \end{cases}$$

1.7.20. Exercise. Define a λ-term P such that

$$P\ c_{n+1} =_\beta c_n$$

Hint: use the same trick as in the proof that the λ-definable functions are closed under primitive recursion. (Kleene got this idea during a visit at his dentist.)

1.7.21. Exercise. Define a λ-term **eq?** such that, for all $n, m \in \mathbb{N}$:

$$\textbf{eq?}\ c_n\ c_m =_\beta \begin{cases} \textbf{true} & \text{if } m = n; \\ \textbf{false} & \text{otherwise.} \end{cases}$$

Hint: use the fixed point theorem to construct a λ-term H such that

$$\begin{aligned} H\ c_n\ c_m \ \ =_\beta\ & \textbf{if } (\textbf{zero?}\ c_n) \\ & \textbf{then } (\textbf{if } (\textbf{zero?}\ c_m) \textbf{ then true else false}) \\ & \textbf{else } (\textbf{if } (\textbf{zero?}\ c_m) \textbf{ then false else } (H\ (P\ c_n)\ (P\ c_m))) \end{aligned}$$

where P is as in the preceding exercise.

Can you prove the result using instead the construction in Lemma 1.5.18?

1.7.22. Exercise. Define a λ-term H such that for all $n \in \mathbb{N}$:

$$H\ c_{2n} =_\beta c_n$$

1.7.23. Exercise. Define a λ-term F such that for all $n \in \mathbb{N}$:

$$H\ c_{n^2} =_\beta c_n$$

1.7.24. Exercise. Prove Corollary 1.5.25.

CHAPTER 2

Intuitionistic logic

The classical understanding of logic is based on the notion of *truth*. The truth of a statement is "absolute" and independent of any reasoning, understanding, or action. Statements are either true or false with no regard to any "observer". Here "false" means the same as "not true", and this is expressed by the *tertium non datur* principle that "$p \vee \neg p$" must hold no matter what the meaning of p is.

Needless to say, the information contained in the claim $p \vee \neg p$ is quite limited. Take the following sentence as an example:

> *There is seven 7's in a row somewhere in the decimal representation of the number π.*

Note that it may very well happen that nobody ever will be able to determine the truth of the above sentence. Yet we are forced to accept that one of the cases must necessarily hold. Another well-known example is as follows:

> *There are two irrational numbers x and y, such that x^y is rational.*

The proof of this fact is very simple: if $\sqrt{2}^{\sqrt{2}}$ is a rational number then we can take $x = y = \sqrt{2}$; otherwise take $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$.

The problem with this proof is that we do not know which of the two possibilities is the right one. Again, there is very little information in this proof, because it is not *constructive*.

These examples demonstrate some of the drawbacks of classical logic, and give hints on why intuitionistic (or constructive) logic is of interest. Although the roots of constructivism in mathematics reach deeply into the XIXth Century, the principles of intuitionistic logic are usually attributed to the works of the Dutch mathematician and philosopher Luitzen Egbertus Jan Brouwer from the beginning of XXth Century. Brouwer is also the inventor of the term "intuitionism", which was originally meant to denote a philosophical approach to the foundations of mathematics, being in opposition to Hilbert's "formalism".

Intuitionistic logic as a branch of formal logic was developed later around the year 1930. The names to be quoted here are Heyting, Glivenko, Kolmogorov and Gentzen. To learn more about the history and motivations see [26] and Chapter 1 of [107].

2.1. Intuitive semantics

In order to understand intuitionism, one should forget the classical, Platonic notion of "truth". Now our judgements about statements are no longer based on any predefined value of that statement, but on the existence of a proof or "construction" of that statement.

The following rules explain the informal constructive semantics of propositional connectives. These rules are sometimes called the *BHK-interpretation* for Brouwer, Heyting and Kolmogorov. The algorithmic flavor of this definition will later lead us to the Curry-Howard isomorphism.

- *A construction of $\varphi_1 \wedge \varphi_2$ consists of a construction of φ_1 and a construction of φ_2;*
- *A construction of $\varphi_1 \vee \varphi_2$ consists of a number $i \in \{1, 2\}$ and a construction of of φ_i;*
- *A construction of $\varphi_1 \to \varphi_2$ is a method (function) transforming every construction of φ_1 into a construction of φ_2;*
- *There is no possible construction of $\bot$ (where $\bot$ denotes falsity).*

Negation $\neg\varphi$ is best understood as an abbreviation of an implication $\varphi \to \bot$. That is, we assert $\neg\varphi$ when the assumption of φ leads to an absurd. It follows that

- *A construction of $\neg\varphi$ is a method that turns every construction of φ into a non-existent object.*

Note that the equivalence between $\neg\varphi$ and $\varphi \to \bot$ holds also in classical logic. But note also that the intuitionistic statement $\neg\varphi$ is much stronger than just "there is no construction for φ".

2.1.1. Example. Consider the following formulas:

1. $\bot \to p$;
2. $((p \to q) \to p) \to p$;
3. $p \to \neg\neg p$;

4. $\neg\neg p \to p$;
5. $\neg\neg\neg p \to \neg p$;
6. $(\neg q \to \neg p) \to (p \to q)$;
7. $(p \to q) \to (\neg q \to \neg p)$;
8. $\neg(p \wedge q) \to (\neg p \vee \neg q)$;
9. $(\neg p \vee \neg q) \to \neg(p \wedge q)$;
10. $((p \leftrightarrow q) \leftrightarrow r) \leftrightarrow (p \leftrightarrow (q \leftrightarrow r))$;
11. $((p \wedge q) \to r) \leftrightarrow (p \to (q \to r))$;
12. $(p \to q) \leftrightarrow (\neg p \vee q)$;
13. $\neg\neg(p \vee \neg p)$.

These formulas are all classical tautologies. Some of them can be easily given a BHK-interpretation, but some of them cannot. For instance, a construction for formula 3, which should be written as "$p \to ((p \to \bot) \to \bot)$", is as follows:

> Given a proof of p, here is a proof of $(p \to \bot) \to \bot$: Take a proof of $p \to \bot$. It is a method to translate proofs of p into proofs of $\bot$. Since we have a proof of p, we can use this method to obtain a proof of $\bot$.

On the other hand, formula 4 does not seem to have such a construction. (The classical symmetry between formula 3 and 4 disappears!)

2.2. Natural deduction

The language of intuitionistic propositional logic is the same as the language of classical propositional logic. We assume an infinite set PV of *propositional variables* and we define the set Φ of *formulas* by induction, represented by the following grammar:

$$\Phi ::= \bot \mid PV \mid (\Phi \to \Phi) \mid (\Phi \vee \Phi) \mid (\Phi \wedge \Phi).$$

That is, our basic connectives are: implication $\to$, disjunction $\vee$, conjunction $\wedge$, and the constant $\bot$ (false).

2.2.1. Convention. The connectives $\neg$ and $\leftrightarrow$ are abbreviations. That is,

- $\neg\varphi$ abbreviates $\varphi \to \bot$;

- $\varphi \leftrightarrow \psi$ abbreviates $(\varphi \to \psi) \wedge (\psi \to \varphi)$.

2.2.2. CONVENTION.

1. We sometimes use the convention that implication is right associative, i.e., we write e.g. $\varphi \to \psi \to \vartheta$ instead of $\varphi \to (\psi \to \vartheta)$.

2. We assume that negation has the highest, and implication the lowest priority, with no preference between $\vee$ and $\wedge$. That is, $\neg p \wedge q \to r$ means $((\neg p) \wedge q) \to r$.

3. And of course we forget about outermost parentheses.

In order to formalize the intuitionistic propositional calculus, we define a proof system, called *natural deduction*, which is motivated by the informal semantics of 2.1.

2.2.3. WARNING. What follows is a quite simplified presentation of natural deduction, which is often convenient for technical reasons, but which is not always adequate. To describe the relationship between various proofs in finer detail, we shall consider a variant of the system in Chapter 4.

2.2.4. DEFINITION.

(i) A *context* is a finite subset of Φ. We use Γ, Δ, etc. to range over contexts.

(ii) The relation $\Gamma \vdash \varphi$ is defined by the rules in Figure 2.1. We also write $\vdash_N$ for $\vdash$.

(iii) We write Γ, Δ instead of $\Gamma \cup \Delta$, and Γ, φ instead of $\Gamma, \{\varphi\}$. We also write $\vdash \varphi$ instead of $\{\} \vdash \varphi$.

(iv) A formal *proof* of $\Gamma \vdash \varphi$ is a finite tree, whose nodes are labelled by pairs of form (Γ', φ'), which will also be written $\Gamma' \vdash \varphi'$, satisfying the following conditions:

   - The root label is $\Gamma \vdash \varphi$;
   - All the leaves are labelled by axioms;
   - The label of each father node is obtained from the labels of the sons using one of the rules.

(v) For infinite Γ we define $\Gamma \vdash \varphi$ to mean that $\Gamma_0 \vdash \varphi$, for some finite subset Γ_0 of Γ.

(vi) If $\vdash \varphi$ then we say that φ is a *theorem* of the intuitionistic propositional calculus.

$$\Gamma, \varphi \vdash \varphi \ (\mathrm{Ax})$$

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ (\wedge\mathrm{I}) \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ (\wedge\mathrm{E})\ \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\ (\vee\mathrm{I})\ \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi} \qquad \frac{\Gamma, \varphi \vdash \rho \quad \Gamma, \psi \vdash \rho \quad \Gamma \vdash \varphi \vee \psi}{\Gamma \vdash \rho}\ (\vee\mathrm{E})$$

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

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\ (\bot\mathrm{E})$$

Figure 2.1: Intuitionistic propositional calculus

The proof system consists of an axiom scheme, and rules. For each logical connective (except $\bot$) we have one or two *introduction* rules and one or two *elimination* rules. An introduction rule for a connective $\bullet$ tells us how a conclusion of the form $\varphi \bullet \psi$ can be derived. An elimination rule describes the way in which $\varphi \bullet \psi$ can be used to derive other formulas. The intuitive meaning of $\Gamma \vdash \varphi$ is that φ is a consequence of the assumptions in Γ.

We give example proofs of our three favourite formulas:

2.2.5. Example. Let Γ abbreviate $\{\varphi \to (\psi \to \vartheta), \varphi \to \psi, \varphi\}$.

(i)

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

(ii)

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

(iii)

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

2.2.6. Remark. Note the distinction between the *relation* $\vdash$, and a formal *proof* of $\Gamma \vdash \varphi$.

The following properties will be useful.

2.2.7. Lemma. *Intuitionistic propositional logic is closed under weakening and substitution, that is,* $\Gamma \vdash \varphi$ *implies* $\Gamma, \psi \vdash \varphi$ *and* $\Gamma[p := \psi] \vdash \varphi[p := \psi]$, *where* $[p := \psi]$ *denotes a substitution of* ψ *for all occurrences of a propositional variable* p.

Proof. Easy induction with respect to the size of proofs. $\square$

2.3. Algebraic semantics of classical logic

To understand better the algebraic semantics for intuitionistic logic let us begin with classical logic. Usually, semantics of classical propositional formulas is defined in terms of the two truth values, 0 and 1 as follows.

2.3.1. Definition. Let $\mathbb{B} = \{0, 1\}$.

(i) A *valuation in* $\mathbb{B}$ is a map $v : PV \to \mathbb{B}$; such a map will also be called a 0-1 *valuation.*

(ii) Given a 0-1 valuation v, define the map $[\![\bullet]\!]_v : \Phi \to \mathbb{B}$ by:

$$\begin{array}{lcll} [\![p]\!]_v & = & v(p), & \text{for } p \in PV; \\ [\![\bot]\!]_v & = & 0; & \\ [\![\varphi \vee \psi]\!]_v & = & \max\{[\![\varphi]\!]_v, [\![\psi]\!]_v\}; & \\ [\![\varphi \wedge \psi]\!]_v & = & \min\{[\![\varphi]\!]_v, [\![\psi]\!]_v\}; & \\ [\![\varphi \to \psi]\!]_v & = & \max\{1 - [\![\varphi]\!]_v, [\![\psi]\!]_v\}. & \end{array}$$

We also write $v(\varphi)$ for $[\![\varphi]\!]_v$.

(iii) A formula $\varphi \in \Phi$ is a *tautology* if $v(\varphi) = 1$ for all valuations in $\mathbb{B}$.

Let us consider an alternative semantics, based on the analogy between classical connectives and set-theoretic operations.

2.3.2. Definition. A *field of sets (over X)* is a nonempty family $\mathcal{R}$ of subsets of X, closed under unions, intersections and complement (to X).

It follows immediately that $\{\}, X \in \mathcal{R}$, for each field of sets $\mathcal{R}$ over X. Examples of fields of sets are:

(i) $P(X)$;

(ii) $\{\{\}, X\}$;

(iii) $\{A \subseteq X : A \text{ finite or } -A \text{ finite}\}$ ($-A$ is the complement of A).

2.3.3. Definition. Let $\mathcal{R}$ be a field of sets over X.

(i) A *valuation in* $\mathcal{R}$ is a map $v : PV \to \mathcal{R}$.

(ii) Given a valuation v in $\mathcal{R}$, define the map $[\![\bullet]\!]_v : \Phi \to X$ by:

$$\begin{array}{lcll}
[\![p]\!]_v & = & v(p) & \text{for } p \in PV \\
[\![\bot]\!]_v & = & \{\} & \\
[\![\varphi \vee \psi]\!]_v & = & [\![\varphi]\!]_v \cup [\![\psi]\!]_v & \\
[\![\varphi \wedge \psi]\!]_v & = & [\![\varphi]\!]_v \cap [\![\psi]\!]_v & \\
[\![\varphi \to \psi]\!]_v & = & (X - [\![\varphi]\!]_v) \cup [\![\psi]\!]_v &
\end{array}$$

We also write $v(\varphi)$ for $[\![\varphi]\!]_v$.

2.3.4. Proposition. *The above two approaches to semantics are equivalent, i.e., the following conditions are equivalent for each field of subsets $\mathcal{R}$ over a nonempty set X:*

1. *φ is a tautology;*
2. *$v(\varphi) = X$, for all valuations v in $\mathcal{R}$.*

Proof. (1) $\Rightarrow$ (2): Suppose that $v(\varphi) \neq X$. There is an element $a \in X$ such that $a \notin v(\varphi)$. Define a 0-1 valuation w so that $w(p) = 1$ iff $a \in v(p)$. Prove by induction that for all formulas ψ

$$w(\psi) = 1 \quad \text{iff} \quad a \in v(\psi).$$

Then $w(\varphi) \neq 1$.

(2) $\Rightarrow$ (1): A 0-1 valuation can be seen as a valuation in $\mathcal{R}$ that assigns only X and $\{\}$ to propositional variables. □

2.3.5. Definition. A *Boolean algebra* is an algebraic system of the form $\mathcal{B} = \langle B, \cup, \cap, -, 0, 1\rangle$, where:

- $\cup, \cap$ are associative and commutative;
- $(a \cup b) \cap c = (a \cap c) \cup (b \cap c)$ and $(a \cap b) \cup c = (a \cup c) \cap (b \cup c)$;
- $a \cup 0 = a$ and $a \cap 1 = a$;
- $-a \cup a = 1$ and $-a \cap a = 0$.

The relation $\leq$ defined by $a \leq b$ iff $a \cup b = b$ is a partial order[1] in every Boolean algebra, and the operations $\cap, \cup$ are the *glb* and *lub* operations w.r.t. this order.

[1] A transitive, reflexive and anti-symmetric relation.

The notion of a Boolean algebra is a straightforward generalization of the notion of a field of sets. Another example of a Boolean algebra is the algebra of truth values $\langle \mathbb{B}, \max, \min, -, 0, 1\rangle$, where $-x$ is $1-x$.

We can generalize the above set semantics to arbitrary Boolean algebras by replacing valuations in a field of sets by valuations in a Boolean algebra in the obvious way. But in fact, every Boolean algebra is isomorphic to a field of sets, so this generalization does not change our semantics.

2.4. Heyting algebras

We will now develop a semantics for intuitionistic propositional logic.

Let Φ be the set of all propositional formulas, let $\Gamma \subseteq \Phi$ (in particular Γ may be empty) and let $\sim$ be the following equivalence relation:

$$\varphi \sim \psi \quad \text{iff} \quad \Gamma \vdash \varphi \to \psi \text{ and } \Gamma \vdash \psi \to \varphi.$$

Let $\mathcal{L}_\Gamma = \Phi/_\sim = \{[\varphi]_\sim : \varphi \in \Phi\}$, and define a partial order $\leq$ over $\mathcal{L}_\Gamma$ by:

$$[\varphi]_\sim \leq [\psi]_\sim \quad \text{iff} \quad \Gamma \vdash \varphi \to \psi.$$

That $\sim$ is an equivalence relation and that $\leq$ is a well-defined partial order is a consequence of the following formulas being provable:

- $\varphi \to \varphi$;
- $(\varphi \to \psi) \to ((\psi \to \vartheta) \to (\varphi \to \vartheta))$;

In addition, we can define the following operations over $\mathcal{L}_\Gamma$:

$$\begin{aligned} [\alpha]_\sim \cup [\beta]_\sim &= [\alpha \vee \beta]_\sim; \\ [\alpha]_\sim \cap [\beta]_\sim &= [\alpha \wedge \beta]_\sim; \\ -[\alpha]_\sim &= [\neg\alpha]_\sim. \end{aligned}$$

These operations are well-defined, because the following formulas are provable:

- $(\varphi \to \varphi') \to (\neg\varphi' \to \neg\varphi)$;
- $(\varphi \to \varphi') \to ((\psi \to \psi') \to ((\varphi \vee \psi) \to (\varphi' \vee \psi')))$;
- $(\varphi \to \varphi') \to ((\psi \to \psi') \to ((\varphi \wedge \psi) \to (\varphi' \wedge \psi')))$.

We can go on and show that operations $\cap$ and $\cup$ are the *glb* and *lub* operations w.r.t. the relation $\leq$, and that the distributivity laws

$$(a \cup b) \cap c = (a \cap c) \cup (b \cap c) \quad \text{and} \quad (a \cap b) \cup c = (a \cup c) \cap (b \cup c)$$

are satisfied.[2] The class $[\bot]_\sim$ is the least element 0 of $\mathcal{L}_\Gamma$, because $\bot \to \varphi$ is provable, and $[\top]_\sim$, where $\top = \bot \to \bot$, is the top element 1. We have $[\top]_\sim = \{\varphi : \Gamma \vdash \varphi\}$. However, there are (not unexpected) difficulties with the complement operation: We have $-a \cap a = [\bot]_\sim$ but not necessarily $-a \cup a = [\top]_\sim$.

The best we can assert about $-a$ is that it is *the greatest element such that* $-a \cap a = 0$, and we can call it a *pseudo-complement*. Since negation is a special kind of implication, the above calls for a generalization. An element c is called a *relative pseudo-complement* of a with respect to b, iff c is the greatest element such that $a \cap c \leq b$. The relative pseudo-complement, if it exists, is denoted $a \Rightarrow b$.

It is not difficult to find out that in our algebra $\mathcal{L}_\Gamma$, often called a *Lindenbaum algebra*, we have $[\varphi]_\sim \Rightarrow [\psi]_\sim = [\varphi \to \psi]_\sim$.

We have just discovered a new type of algebra, called *Heyting algebra* or *pseudo-Boolean algebra*.

2.4.1. Definition. A *Heyting algebra* is an algebraic system of the form $\mathcal{H} = \langle H, \cup, \cap, \Rightarrow, -, 0, 1\rangle$, that satisfies the following conditions:

- $\cup, \cap$ are associative and commutative;
- $(a \cup b) \cap c = (a \cap c) \cup (b \cap c)$ and $(a \cap b) \cup c = (a \cup c) \cap (b \cup c)$;
- $a \cup 0 = a$ and $a \cap 1 = a$;
- $a \cup a = a$;
- $a \cap c \leq b$ is equivalent to $c \leq a \Rightarrow b$ (where $a \leq b$ stands for $a \cup b = b$);
- $-a = a \Rightarrow 0$.

The above conditions amount to as much as saying that $\mathcal{H}$ is a distributive lattice with zero and relative pseudo-complement defined for each pair of elements. In particular, each Boolean algebra is a Heyting algebra with $a \Rightarrow b$ defined as $-a \cup b$. The most prominent example of a Heyting algebra which is not a Boolean algebra is the algebra of open sets of a topological space, for instance the algebra of open subsets of the Euclidean plane $\mathbb{R}^2$.

2.4.2. Definition.

- The symbol $\varrho(a, b)$ denotes the distance between points $a, b \in \mathbb{R}^2$;
- A subset A of $\mathbb{R}^2$ is *open* iff for every $a \in A$ there is an $r > 0$ with $\{b \in \mathbb{R}^2 : \varrho(a, b) < r\} \subseteq A$;
- If A is a subset of $\mathbb{R}^2$ then $\mathrm{Int}(A)$ denotes the *interior* of A, i.e., the union of all open subsets of A.

[2] That is, $\mathcal{L}_\Gamma$ is a *distributive lattice*.

2.4.3. Proposition. *Let* $\mathcal{H} = \langle \mathcal{O}(\mathbb{R}^2), \cup, \cap, \Rightarrow, \sim, 0, 1\rangle$, *where*

- $\mathcal{O}(\mathbb{R}^2)$ *is the family of all open subsets of* $\mathbb{R}^2$*;*
- *the operations* $\cap$, $\cup$ *are set-theoretic;*
- $A \Rightarrow B := \mathrm{Int}(-A \cup B)$, *for arbitrary open sets* A *and* B*;*
- $0 = \{\}$ *and* $1 = \mathbb{R}^2$.
- $\sim A = \mathrm{Int}(-A)$, *where* $-$ *is the set-theoretic complement.*

Then $\mathcal{H}$ *is a Heyting algebra.*

Proof. Exercise 2.7.6. □

In fact, every Heyting algebra is isomorphic to a subalgebra of the algebra of open sets of a topological space. A comprehensive study of the algebraic semantics for intuitionistic (and classical) logic is the book of Rasiowa and Sikorski [88]. See also Chapter 13 of [108].

The semantics of intuitionistic propositional formulas is now defined as follows.

2.4.4. Definition. Let $\mathcal{H} = \langle H, \cup, \cap, \Rightarrow, -, 0, 1\rangle$ be a Heyting algebra.

(i) A *valuation* v in a $\mathcal{H}$ is a map $v : PV \to H$.

(ii) Given a valuation v in $\mathcal{H}$, define the map $[\![\bullet]\!]_v : \Phi \to H$ by:

$$\begin{array}{lcll}
[\![p]\!]_v & = & v(p) & \text{for } p \in PV \\
[\![\bot]\!]_v & = & 0 & \\
[\![\varphi \vee \psi]\!]_v & = & [\![\varphi]\!]_v \cup [\![\psi]\!]_v & \\
[\![\varphi \wedge \psi]\!]_v & = & [\![\varphi]\!]_v \cap [\![\psi]\!]_v & \\
[\![\varphi \to \psi]\!]_v & = & [\![\varphi]\!]_v \Rightarrow [\![\psi]\!]_v &
\end{array}$$

As usual, we write $v(\varphi)$ for $[\![\varphi]\!]_v$.

2.4.5. Notation. Let $\mathcal{H} = \langle H, \cup, \cap, \Rightarrow, -, 0, 1\rangle$ be a Heyting algebra. We write:

- $\mathcal{H}, v \models \varphi$, whenever $v(\varphi) = 1$;
- $\mathcal{H} \models \varphi$, whenever $\mathcal{H}, v \models \varphi$, for all v;
- $\mathcal{H}, v \models \Gamma$, whenever $\mathcal{H}, v \models \varphi$, for all $\varphi \in \Gamma$;
- $\mathcal{H} \models \Gamma$, whenever $\mathcal{H}, v \models \Gamma$, for all v;
- $\models \varphi$, whenever $\mathcal{H}, v \models \varphi$, for all H, v;
- $\Gamma \models \varphi$, whenever $\mathcal{H}, v \models \Gamma$ implies $\mathcal{H}, v \models \varphi$, for all H and v.

We say that a formula φ such that $\models \varphi$ is *intuitionistically valid* or is an *intuitionistic tautology*. It follows from the following completeness theorem that the notions of a theorem and a tautology coincide for intuitionistic propositional calculus.

2.4.6. THEOREM (Soundness and Completeness). *The following conditions are equivalent*

1. $\Gamma \vdash \varphi$;

2. $\Gamma \models \varphi$.

PROOF. (1) $\Rightarrow$ (2): Verify that all provable formulas are valid in all Heyting algebras (induction w.r.t. proofs).

(2) $\Rightarrow$ (1): This follows from our construction of the Lindenbaum algebra. Indeed, suppose that $\Gamma \models \varphi$, but $\Gamma \not\vdash \varphi$. Then $\varphi \not\sim \top$, i.e., $[\varphi]_\sim \neq 1$ in $\mathcal{L}_\Gamma$. Define a valuation v by $v(p) = [p]_\sim$ in $\mathcal{L}_\Gamma$ and prove by induction that $v(\psi) = [\psi]_\sim$, for all formulas ψ. It follows that $v(\varphi) \neq 1$, a contradiction.

□

2.4.7. EXAMPLE. To see that Peirce's law $((p \to q) \to p) \to p$ is not intuitionistically valid, consider the algebra of open subsets of $\mathbb{R}^2$. Take $v(p)$ to be the whole space without one point, and $v(q) = \{\}$. (Note that $a \Rightarrow b = 1$ in a Heyting algebra iff $a \leq b$.)

Intuitionistic logic is not finite-valued: There is no single finite Heyting algebra $\mathcal{H}$ such that $\vdash \varphi$ is equivalent to $\mathcal{H} \models \varphi$. Indeed, consider the formula $\bigvee\{p_i \leftrightarrow p_j : i, j = 0, \dots, n \text{ and } i \neq j\}$. (Here the symbol $\bigvee$ abbreviates the disjunction of all members of the set.) This formula is not valid in general (Exercise 2.7.10), although it is valid in all Heyting algebras of cardinality at most n.

But finite Heyting algebras are sufficient for the semantics, as well as one sufficiently "rich" infinite algebra.

2.4.8. THEOREM.

1. *A formula φ of length n is valid iff it is valid in all Heyting algebras of cardinality at most 2^{2^n};*

2. *Let $\mathcal{H}$ be the algebra of all open subsets of a dense-in-itself*[3] *metric space V (for instance the algebra of all open subsets of $\mathbb{R}^2$). Then $\mathcal{H} \models \varphi$ iff φ is valid.*

[3]Every point x is a limit of a sequence $\{x_n\}_n$, where $x_n \neq x$, for all n.

We give only a sketch of the main ideas of the proof, for the most curious reader. See [88] for details.

For (1), suppose $\mathcal{H}, v \not\models \varphi$, and let $\varphi_1, \dots \varphi_m$ be all subformulas of φ. We construct a small model $\mathcal{H}'$ as a distributive sublattice of $\mathcal{H}$, with 0 and 1, generated by the elements $v(\varphi_1), \dots, v(\varphi_m)$. This lattice is a Heyting algebra (warning: this is not a Heyting subalgebra of $\mathcal{H}$), and $\mathcal{H}', v' \not\models \varphi$, for a suitable v'.

As for (2), every finite algebra can be embedded into the algebra of open subsets of some open subset of V, and this algebra is a homomorphic image of $\mathcal{H}$. Thus, every valuation in a finite algebra can be translated into a valuation in $\mathcal{H}$.

From part (1) of the above theorem, it follows that intuitionistic propositional logic is decidable. But the upper bound obtained this way (double exponential space) can be improved down to polynomial space, with help of other methods, see [103].

2.5. Kripke semantics

We now introduce another semantics of intuitionistic propositional logic.

2.5.1. DEFINITION. A *Kripke model* is defined as a tuple of the form $\mathcal{C} = \langle C, \leq, \Vdash \rangle$, where C is a non-empty set, $\leq$ is a partial order in C and $\Vdash$ is a binary relation between elements of C (called *states* or *possible worlds*) and propositional variables, that satisfies the following monotonicity condition:

$$\text{If } c \leq c' \text{ and } c \Vdash p \text{ then } c' \Vdash p.$$

The intuition is that elements of the model represent states of knowledge. The relation $\leq$ represents extending states by gaining more knowledge, and the relation $\Vdash$ tells which atomic formulas are known to be true in a given state. We extend this relation to provide meaning for propositional formulas as follows.

2.5.2. DEFINITION. If $\mathcal{C} = \langle C, \leq, \Vdash \rangle$ is a Kripke model, then

- $c \Vdash \varphi \vee \psi$ iff $c \Vdash \varphi$ or $c \Vdash \psi$;
- $c \Vdash \varphi \wedge \psi$ iff $c \Vdash \varphi$ and $c \Vdash \psi$;
- $c \Vdash \varphi \to \psi$ iff $c' \Vdash \psi$, for all c' such that $c \leq c'$ and $c' \Vdash \varphi$;
- $c \Vdash \bot$ never happens.

We use $\mathcal{C} \Vdash \varphi$ to mean that $c \Vdash \varphi$, for all $c \in \mathcal{C}$.

Note that the above definition implies the following rule for negation:

- $c \Vdash \neg\varphi$ iff $c' \nVdash \varphi$, for all $c' \geq c$.

and the following generalized monotonicity (proof by easy induction):

$$\text{If } c \leq c' \text{ and } c \Vdash \varphi \text{ then } c' \Vdash \varphi.$$

We now want to show completeness of Kripke semantics. For this, we transform every Heyting algebra into a Kripke model.

2.5.3. Definition. A *filter* in a Heyting algebra $\mathcal{H} = \langle H, \cup, \cap, \Rightarrow, -, 0, 1\rangle$ is a nonempty subset F of H, such that

- $a, b \in F$ implies $a \cap b \in F$;
- $a \in F$ and $a \leq b$ implies $b \in F$.

A filter F is *proper* iff $F \neq H$. A proper filter F is *prime* iff $a \cup b \in F$ implies that either a or b belongs to F.

2.5.4. Lemma. *Let F be a proper filter in $\mathcal{H}$ and let $a \notin F$. There exists a prime filter G such that $F \subseteq G$ and $a \notin G$.*

We only give a hint for the proof that can be found e.g., in [88]. Consider the family of all filters G containing F and such that $a \notin G$, ordered by inclusion. Apply Kuratowski-Zorn Lemma to show that this family has a maximal element. This is a prime filter (Exercise 2.7.12) although it is not necessarily a maximal proper filter.

2.5.5. Lemma. *Let v be a valuation in a Heyting algebra $\mathcal{H}$. There is a Kripke model $\mathcal{C} = \langle C, \leq, \Vdash \rangle$, such that $\mathcal{H}, v \models \varphi$ iff $\mathcal{C} \Vdash \varphi$, for all formulas φ.*

Proof. We take C to be the set of all prime filters in $\mathcal{H}$. The relation $\leq$ is inclusion, and we define $F \Vdash p$ iff $v(p) \in F$. By induction, we prove that, for all formulas ψ,

$$F \Vdash \psi \quad \text{iff} \quad v(\psi) \in F. \tag{2.1}$$

The only nontrivial case of this induction is when $\psi = \psi' \to \psi''$. Assume $F \Vdash \psi' \to \psi''$, and suppose that $v(\psi' \to \psi'') = v(\psi') \Rightarrow v(\psi'') \notin F$. Take the least filter G' containing $F \cup \{v(\psi')\}$. Then

$$G' = \{b : b \geq f \cap v(\psi') \text{ for some } f \in F\},$$

and we have $v(\psi'') \notin G'$, in particular G' is proper. Indeed, otherwise $v(\psi'') \geq f \cap v(\psi')$, for some $f \in F$, and thus $f \leq v(\psi') \Rightarrow v(\psi'') \in F$ — a contradiction.

We extend G' to a prime filter G, not containing $v(\psi'')$. By the induction hypothesis, $G \Vdash \psi'$. Since $F \Vdash \psi' \to \psi''$, it follows that $G \Vdash \psi''$. That is, $v(\psi'') \in G$ — a contradiction.

For the converse, assume that $v(\psi' \to \psi'') \in F \subseteq G \Vdash \psi'$. From the induction hypothesis we have $v(\psi') \in G$ and since $F \subseteq G$ we obtain $v(\psi') \Rightarrow v(\psi'') \in G$. Thus $v(\psi'') \geq v(\psi') \cap (v(\psi') \Rightarrow v(\psi'')) \in G$, and from the induction hypothesis we conclude $G \Vdash \psi''$ as desired.

The other cases are easy. Note that primality is essential for disjunction. Having shown (2.1), assume that $\mathcal{C} \Vdash \varphi$ and $\mathcal{H}, v \not\models \varphi$. Then $v(\varphi) \neq 1$ and there exist a proper filter not containing $v(\varphi)$. This filter extends to a prime filter G such that $v(\varphi) \notin G$ and thus $G \nVdash \varphi$. On the other hand, if $\mathcal{H}, v \models \varphi$, then $v(\varphi) = 1$ and 1 belongs to all filters in $\mathcal{H}$. □

2.5.6. Theorem. *The sequent $\Gamma \vdash \varphi$ is provable iff for all Kripke models $\mathcal{C}$, the condition $\mathcal{C} \Vdash \Gamma$ implies $\mathcal{C} \Vdash \varphi$.*

Proof. The left-to-right part is shown by induction (Exercise 2.7.13). For the other direction assume $\Gamma \nvdash \varphi$. Then $\mathcal{H}, v \models \Gamma$ but $\mathcal{H}, v \not\models \varphi$, for some $\mathcal{H}, v$. From the previous lemma we have a Kripke model $\mathcal{C}$ with $\mathcal{C} \Vdash \Gamma$ and $\mathcal{C} \nVdash \varphi$. □

Here is a nice application of Kripke semantics.

2.5.7. Proposition. *If $\vdash \varphi \vee \psi$ then either $\vdash \varphi$ or $\vdash \psi$.*

Proof. Assume $\nvdash \varphi$ and $\nvdash \psi$. There are Kripke models $\mathcal{C}_1 = \langle C_1, \leq_1, \Vdash_1 \rangle$ and $\mathcal{C}_2 = \langle C_2, \leq_2, \Vdash_2 \rangle$ and states $c_1 \in C_1$ and $c_2 \in C_2$, such that $c_1 \nVdash \varphi$ and $c_2 \nVdash \psi$. Without loss of generality we can assume that c_1 and c_2 are least elements of $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, and that $C_1 \cap C_2 = \{\}$. Let $\mathcal{C} = \langle C_1 \cup C_2 \cup \{c_0\}, \leq, \Vdash \rangle$, where $c_0 \notin C_1 \cup C_2$, the order is the union of $\leq_1$ and $\leq_2$ extended by c_0 taken as the least element, and $\Vdash$ is the union of $\Vdash_1$ and $\Vdash_2$. That is,

$$c_0 \nVdash p,$$

for all variables p. It is easy to see that this is a Kripke model. In addition we have $\mathcal{C}, c_1 \Vdash \vartheta$ iff $\mathcal{C}_1, c_1 \Vdash \vartheta$, for all formulas ϑ, and a similar property holds for c_2.

Now suppose that $\vdash \varphi \vee \psi$. By soundness, we have $c_0 \Vdash \varphi \vee \psi$, and thus either $c_0 \Vdash \varphi$ or $c_0 \Vdash \psi$, by definition of $\Vdash$. Then either $c_1 \Vdash \varphi$ or $c_2 \Vdash \psi$, because of monotonicity. □

2.6. The implicational fragment

The most important logical conjective is the implication. Thus, it is meaningful to study the fragment of propositional calculus with only one connective: the implication. This is the true *minimal logic*. The natural deduction system for the implicational fragment consists of the rules ($\to$ E), ($\to$ I) and the axiom scheme.

2.6.1. THEOREM. *The implicational fragment of intuitionistic propositional calculus is complete with respect to Kripke models, i.e.,* $\Gamma \vdash \varphi$ *is provable iff for all Kripke models* $\mathcal{C}$*, the condition* $\mathcal{C} \Vdash \Gamma$ *implies* $\mathcal{C} \Vdash \varphi$.

PROOF. The implication from left to right follows from soundness of the full natural deduction system, of which our minimal logic is a fragment. For the proof in the other direction, let us assume that $\Gamma \nvdash \varphi$. We define a Kripke model $\mathcal{C} = \langle C, \leq, \Vdash \rangle$, where

$$C = \{\Delta : \Gamma \subseteq \Delta, \text{ and } \Delta \text{ is closed under } \vdash\}.$$

That is, $\Delta \in C$ means that $\Delta \vdash \psi$ implies $\psi \in \Delta$.

The relation $\leq$ is inclusion and $\Vdash$ is $\in$, that is, $\Delta \Vdash p$ holds iff $p \in \Delta$, for all propositional variables p. By induction we show the following claim:

$$\Delta \Vdash \psi \quad \text{iff} \quad \psi \in \Delta,$$

for all implicational formulas ψ and all states Δ. The case of a variable is immediate from the definition. Let ψ be $\psi_1 \to \psi_2$ and let $\Delta \Vdash \psi$. Take $\Delta' = \{\vartheta : \Delta, \psi_1 \vdash \vartheta\}$. Then $\psi_1 \in \Delta'$ and, by the induction hypothesis, $\Delta' \Vdash \psi_1$. Thus $\Delta' \Vdash \psi_2$ and we get $\psi_2 \in \Delta'$, again by the induction hypothesis. Thus, $\Delta, \psi_1 \vdash \psi_2$, and by ($\to$ I) we get what we want.

Now assume $\psi \in \Delta$ (that is $\Delta \vdash \psi$) and take $\Delta' \geq \Delta$ with $\Delta' \Vdash \psi_1$. Then $\psi_1 \in \Delta'$, i.e., $\Delta' \vdash \psi_1$. But also $\Delta' \vdash \psi_1 \to \psi_2$, because $\Delta \subseteq \Delta'$. By ($\to$ E) we can derive $\Delta' \vdash \psi_2$, which means, by the induction hypothesis, that $\Delta' \Vdash \psi_2$. □

The completeness theorem has a very important consequence: the conservativity of the full intuitionistic propositional calculus over its implicational fragment.

2.6.2. THEOREM. *Let* φ *be an implicational formula, and let* Γ *be a set of implicational formulas. If* $\Gamma \vdash \varphi$ *can be derived in the intuitionistic propositional calculus then it can be derived in the implicational fragment.*

PROOF. Easy. But note that we use only one half from the two completeness theorems 2.5.6 and 2.6.1: we need only soundness of the full logic and only the other direction for the fragment. □

2.7. Exercises

2.7.1. EXERCISE. Find constructions for formulas (1), (3), (5), (7), (9), (11) and (13) of Example 2.1.1, and do not find constructions for the other formulas.

2.7.2. Exercise. Prove Lemma 2.2.7.

2.7.3. Exercise. Give natural deduction proofs for the formulas of Exercise 2.7.1.

2.7.4. Exercise. Show that the relation $\leq$ defined in a Boolean algebra by the condition $a \leq b$ iff $a \cup b = b$ is a partial order and that

- $a \cap b \leq a$;
- the condition $a \leq b$ is equivalent to $a \cap b = a$;
- the operations $\cup$ and $\cap$ are respectively the upper and lower bound wrt. $\leq$;
- the constants 0 and 1 are respectively the bottom and top element.

2.7.5. Exercise. Show that the relation $\leq$ defined in a Heyting algebra by the condition $a \leq b$ iff $a \cup b = b$ is a partial order and that

- $-a \cap a = 0$;
- $(a \cup b) \cap a = a$ and $a \cap b \leq a$;
- the condition $a \leq b$ is equivalent to $a \cap b = a$, and to $a \Rightarrow b = 1$;
- the operations $\cup$ and $\cap$ are respectively the upper and lower bound wrt. $\leq$;
- the constants 0 and 1 are respectively the bottom and top element.

2.7.6. Exercise. Prove Proposition 2.4.3.

2.7.7. Exercise. Fill in the details of the proof that the Lindenbaum algebra $\mathcal{L}_\Gamma$ of 2.4 is indeed a Heyting algebra.

2.7.8. Exercise. Complete the proof of the completeness theorem 2.4.6.

2.7.9. Exercise. Show that the formulas (2), (4), (6), (8) and (10) are not intuitionistically valid. (Use open subsets of $\mathbb{R}^2$ or construct Kripke models.)

2.7.10. Exercise. Show that the formula $\bigvee\{p_i \leftrightarrow p_j : i, j = 0, \ldots, n$ and $i \neq j\}$ is not intuitionistically valid.

2.7.11. Exercise. A filter is *maximal* iff it is a maximal proper filter. Show that each maximal filter is prime. Show also that in a Boolean algebra every prime filter is maximal.

2.7.12. Exercise. Complete the proof of Lemma 2.5.4.

2.7.13. EXERCISE. Complete the proof of Theorem 2.5.6, part $\Rightarrow$. (*Hint:* choose a proper induction hypothesis.)

2.7.14. EXERCISE. Can the proof of Theorem 2.6.1 be generalized to the full propositional calculus?

2.7.15. EXERCISE. A state c in a Kripke model $\mathcal{C}$ *determines* p iff either $c \Vdash p$ or $c \Vdash \neg p$. Define a 0-1 valuation v_c by $v_c(p) = 1$ iff $c \Vdash p$. Show that if c determines all propositional variables in φ then $v_c(\varphi) = 1$ implies $c \Vdash \varphi$.

2.7.16. EXERCISE. Let φ be a classical tautology such that all propositional variables in φ are among $p_1, \ldots, p_n$. Show that the formula $(p_1 \vee \neg p_1) \to \cdots \to (p_n \vee \neg p_n) \to \varphi$ is intuitionistically valid.

2.7.17. EXERCISE. Prove the Glivenko theorem: A formula φ is a classical tautology iff $\neg\neg\varphi$ is an intuitionistic tautology.

2.7.18. WARNING. The Glivenko theorem does not extend to first-order logic.

CHAPTER 3

Simply typed λ-calculus

Recall from the first chapter that a λ-term, unlike the functions usually considered in mathematics, does not have a fixed *domain* and *range.* Thus, whereas we would consider the function $n \mapsto n^2$ as a function from natural numbers to natural numbers (or from integers to natural numbers, etc.) there is no corresponding requirement in λ-calculus. Or, to be more precise, there is no corresponding requirement in *type-free* λ-calculus.

Curry [23] and Church [18] also introduced versions of their systems with *types.* These systems form the topic of the present chapter.

3.1. Simply typed λ-calculus à la Curry

We begin with the simply typed λ-calculus à la Curry.

3.1.1. Definition.

(i) Let U denote a denumerably infinite alphabet whose members will be called *type variables.* The set Π of *simple types* is the set of strings defined by the grammar:

$$\Pi ::= U \mid (\Pi \to \Pi)$$

We use $\alpha, \beta, \dots$ to denote arbitrary type variables, and $\sigma, \tau, \dots$ to denote arbitrary types. We omit outermost parentheses, and omit other parentheses with the convention that $\to$ associates to the right.

(ii) The set C of *contexts* is the set of all sets of pairs of the form

$$\{x_1 : \tau_1, \dots , x_n : \tau_n\}$$

with $\tau_1, \dots , \tau_n \in \Pi$, $x_1, \dots , x_n \in V$ (variables of Λ) and $x_i \neq x_j$ for $i \neq j$.

(iii) The *domain* of a context $\Gamma = \{x_1 : \tau_1, \dots, x_n : \tau_n\}$ is defined by:

$$\mathrm{dom}(\Gamma) = \{x_1, \dots, x_n\}$$

We write $x : \tau$ for $\{x : \tau\}$ and Γ, Γ' for $\Gamma \cup \Gamma'$ if $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \{\}$.

(iv) The *range* of a context $\Gamma = \{x_1 : \tau_1, \dots, x_n : \tau_n\}$ is defined by:

$$|\Gamma| = \{\tau \in \Pi \mid (x : \tau) \in \Gamma, \text{ for some } x\}.$$

(v) The *typability* relation $\vdash$ on $C \times \Lambda \times \Pi$ is defined by:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\, N : \tau}$$

where we require that $x \notin \mathrm{dom}(\Gamma)$ in the first and second rule.

(vi) The simply typed λ-calculus $\lambda\!\to$ is the triple $(\Lambda, \Pi, \vdash)$. To distinguish between this system and variants, the present one will also be called *simply typed λ-calculus à la Curry* or just $\lambda\!\to$ à la Curry.

3.1.2. EXAMPLE. Let σ, τ, ρ be arbitrary types. Then:

(i) $\vdash \lambda x.x : \sigma \to \sigma$;

(ii) $\vdash \lambda x.\lambda y.x : \sigma \to \tau \to \sigma$;

(iii) $\vdash \lambda x.\lambda y.\lambda z.x\ z\ (y\ z) : (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho$.

3.1.3. DEFINITION. If $\Gamma \vdash M : \sigma$ then we say that M *has type* σ *in* Γ. We say that $M \in \Lambda$ is *typable* if there are Γ and σ such that $\Gamma \vdash M : \sigma$.

The set of typable terms is a subset—in fact, a proper subset—of the set of all λ-terms. In this subset, restrictions are made regarding which λ-terms may be applied to other λ-terms.

Very informally, the type variables denote some unspecified sets, and $\sigma \to \tau$ denotes the set of functions from σ to τ. Saying that M has type $\sigma \to \tau$ in Γ then intuitively means that this set of functions contains the particular function that M informally denotes. For instance, $\vdash \lambda x.x : \sigma \to \sigma$ informally states that the identity function is a function from a certain set to itself.

A context is an assumption that some elements $x_1, \dots, x_n$ have certain types $\sigma_1, \dots, \sigma_n$, respectively. Moreover, if x has type σ and M has type τ then $\lambda x.M$ has type $\sigma \to \tau$. This reflects the intuition that if M denotes an element of τ for each x in σ, then $\lambda x.M$ denotes a function from σ to τ. In a similar vein, if M has type $\sigma \to \tau$ and N has type σ, then $M\ N$ has type τ.

3.1.4. WARNING. The idea that types denote sets should not be taken too literally. For instance, if A and B are sets then the set of functions from A to A is disjoint from the set of functions from B to B. In contrast, if σ and τ are different simple types, then a single term may have both types, at least in the version of simply typed λ-calculus presented above.

Just like λ-calculus provides a foundation of higher-order functional programming languages like LISP and Scheme, various typed λ-calculi provide a foundation of programming languages with types like Pascal, ML, and Haskell. Typed λ-calculi are also of independent interest in proof theory as we shall have frequent occasion to see in these notes.

We conclude this section with a brief review of some of the most fundamental properties of $\lambda{\to}$. The survey follows [8].

The following shows that only types of free variables of a term matter in the choice contexts.

3.1.5. LEMMA (Free variables lemma). *Assume that* $\Gamma \vdash M : \sigma$. *Then:*

(i) $\Gamma \subseteq \Gamma'$ *implies* $\Gamma' \vdash M : \sigma$;

(ii) $\mathrm{FV}(M) \subseteq \mathrm{dom}(\Gamma)$;

(iii) $\Gamma' \vdash M : \sigma$ *where* $\mathrm{dom}(\Gamma') = \mathrm{FV}(M)$ *and* $\Gamma' \subseteq \Gamma$.

PROOF. (i) by induction on the derivation of $\Gamma \vdash M : \sigma$. As in [8] we present the proof in some detail and omit such details in the remainder.

1. The derivation is
$$\frac{}{\Delta, x : \sigma \vdash x : \sigma}$$
where $\Gamma = \Delta, x : \sigma$, $x \notin \mathrm{dom}(\Delta)$ and $M = x$. Since $\Gamma \subseteq \Gamma'$ and Γ' is a context, $\Gamma' = \Delta', x : \sigma$ for some Δ' with $x \notin \mathrm{dom}(\Delta')$. Hence $\Delta', x : \sigma \vdash x : \sigma$, as required.

2. The derivation ends in
$$\frac{\Gamma, x : \tau_1 \vdash P : \tau_2}{\Gamma \vdash \lambda x.P : \tau_1 \to \tau_2}$$
where $x \notin \mathrm{dom}(\Gamma)$, $\sigma = \tau_1 \to \tau_2$, and $M = \lambda x.P$. Without loss of generality we can assume that $x \notin \mathrm{dom}(\Gamma')$. Then $\Gamma, x : \tau_1 \subseteq \Gamma', x : \tau_1$ so by the induction hypothesis $\Gamma', x : \tau_1 \vdash P : \tau_2$. Then we also have $\Gamma \vdash \lambda x.P : \tau_1 \to \tau_2$, as required.

3. The derivation ends in
$$\frac{\Gamma \vdash P : \tau \to \sigma \quad \Gamma \vdash Q : \tau}{\Gamma \vdash P\,Q : \sigma}$$
where $M = P\,Q$. By the induction hypothesis (twice) $\Gamma' \vdash P : \tau \to \sigma$ and $\Gamma' \vdash Q : \tau$, and then $\Gamma' \vdash P\,Q : \sigma$, as required.

(ii)-(iii) by induction on the derivation of $\Gamma \vdash M : \sigma$. □

The following shows how the type of some term must have been obtained depending on the form of the term.

3.1.6. Lemma (Generation lemma).

(i) *$\Gamma \vdash x : \sigma$ implies $x : \sigma \in \Gamma$;*

(ii) *$\Gamma \vdash M\ N : \sigma$ implies that there is a τ such that $\Gamma \vdash M : \tau \to \sigma$ and $\Gamma \vdash N : \tau$.*

(iii) *$\Gamma \vdash \lambda x.M : \sigma$ implies that there are τ and ρ such that $\Gamma, x : \tau \vdash M : \rho$ and $\sigma = \tau \to \rho$.*

Proof. By induction on the length of the derivation. □

3.1.7. Definition. The *substitution of type τ for type variable α in type σ,* written $\sigma[\alpha := \tau]$, is defined by:

$$\begin{array}{lcll} \alpha[\alpha := \tau] & = & \tau & \\ \beta[\alpha := \tau] & = & \beta & \text{if } \alpha \neq \beta \\ (\sigma_1 \to \sigma_2)[\alpha := \tau] & = & \sigma_1[\alpha := \tau] \to \sigma_2[\alpha := \tau] & \end{array}$$

The notation $\Gamma[\alpha := \tau]$ stands for the context $\{(x : \sigma[\alpha := \tau]) \mid (x : \sigma) \in \Gamma\}$.

The following shows that the type variables range over all types; this is a limited form of *polymorphism* [90]; we will hear much more about polymorphism later. The proposition also shows, similarly, that free term variables range over arbitrary terms.

3.1.8. Proposition (Substitution lemma).

(i) *If $\Gamma \vdash M : \sigma$, then $\Gamma[\alpha := \tau] \vdash M : \sigma[\alpha := \tau]$.*

(ii) *If $\Gamma, x : \tau \vdash M : \sigma$ and $\Gamma \vdash N : \tau$ then $\Gamma \vdash M[x := N] : \sigma$.*

Proof. By induction on the derivation of $\Gamma \vdash M : \sigma$ and generation of $\Gamma, x : \tau \vdash M : \sigma$, respectively. □

The following shows that reduction preserves typing.

3.1.9. Proposition (Subject reduction). *If $\Gamma \vdash M : \sigma$ and $M \to_\beta N$, then $\Gamma \vdash N : \sigma$.*

Proof. By induction on the derivation of $M \to_\beta N$ using the substitution lemma and the generation lemma. □

3.1.10. Remark. The similar property

$$\Gamma \vdash N : \sigma \;\&\; M \twoheadrightarrow_\beta N \;\Rightarrow\; \Gamma \vdash M : \sigma$$

is called *subject expansion* and does *not* hold in $\lambda\!\to$, see Exercise 3.6.2.

3.1.11. Corollary. *If $\Gamma \vdash M : \sigma$ and $M \twoheadrightarrow_\beta N$, then $\Gamma \vdash N : \sigma$.*

3.1.12. Theorem (Church-Rosser property for typable terms). *Suppose that $\Gamma \vdash M : \sigma$. If $M \twoheadrightarrow_\beta N$ and $M \twoheadrightarrow_\beta N'$, then there exists an L such that $N \twoheadrightarrow_\beta L$ and $N' \twoheadrightarrow_\beta L$ and $\Gamma \vdash L : \sigma$.*

Proof. By the Church-Rosser property for λ-terms and the subject reduction property. □

3.2. Simply typed λ-calculus à la Church

As mentioned earlier, simply typed λ-calculus was introduced by Curry [23] and Church [18]. More precisely, Curry considered types for *combinatory logic,* but his formulation was later adapted to λ-calculus [24].

There were several other important differences between the systems introduced by Church and Curry.

In Curry's system the terms are those of type-free λ-calculus and the typing relation selects among these the typable terms. For instance, $\lambda x.x$ is typable, whereas $\lambda x.x\,x$ is not.

In Church's original system, the typing rules were built into the term formation rules, as follows. Let V_σ denote a denumerable set of variables for each $\sigma \in \Pi$. Then define the set Λ_σ of simply typed terms of type σ by the clauses:

$$\begin{array}{lcl} x \in V_\sigma & \Rightarrow & x \in \Lambda_\sigma \\ M \in \Lambda_{\sigma\to\tau} \;\&\; N \in \Lambda_\sigma & \Rightarrow & M\,N \in \Lambda_\tau \\ M \in \Lambda_\tau \;\&\; x \in \Lambda_\sigma & \Rightarrow & \lambda x^\sigma.M \in \Lambda_{\sigma\to\tau} \end{array}$$

The set of all simply typed terms is then taken as the union over all simple types σ of the simply typed terms of type σ.

Instead of assuming that the set of variables is partitioned into disjoint sets indexed by the set of simple types, we can use contexts to decide the types of variables as in the system à la Curry. Also, as in the system à la Curry, we can select the typable terms among a larger set. This yields the following, more common, formulation of simply typed λ-calculus à la Church.

3.2.1. Definition.

(i) The set Λ_Π of pseudo-terms is defined by the following grammar:

$$\Lambda_\Pi ::= V \mid (\lambda x{:}\, \Pi\; \Lambda_\Pi) \mid (\Lambda_\Pi\; \Lambda_\Pi)$$

where V is the set of (λ-term) variables and Π is the set of simple types.[1] We adopt the same terminology, notation, and conventions for pseudo-terms as for λ-terms, see 1.3–1.10, mutatis mutandis.

(ii) The *typability* relation $\Vdash$ on $C \times \Lambda_\Pi \times \Pi$ is defined by:[2]

$$\frac{}{\Gamma, x : \tau \vdash^* x : \tau} \qquad \frac{\Gamma, x : \sigma \vdash^* M : \tau}{\Gamma \vdash^* \lambda x{:}\sigma.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash^* M : \sigma \to \tau \quad \Gamma \vdash^* N : \sigma}{\Gamma \vdash^* M\, N : \tau}$$

where we require that $x \notin \mathrm{dom}(\Gamma)$ in the first and second rule.

(iii) The simply typed λ-calculus à la Church ($\lambda\!\to$ *à la Church,* for short) is the triple $(\Lambda_\Pi, \Pi, \Vdash)$.

(iv) If $\Gamma \vdash^* M : \sigma$ then we say that M *has type* σ *in* Γ. We say that $M \in \Lambda_\Pi$ is *typable* if there are Γ and σ such that $\Gamma \vdash^* M : \sigma$.

3.2.2. Example. Let σ, τ, ρ be arbitrary simple types. Then:

(i) $\vdash^* \lambda x{:}\sigma.x : \sigma \to \sigma$;

(ii) $\vdash^* \lambda x{:}\sigma.\lambda y{:}\tau.x : \sigma \to \tau \to \sigma$;

(iii) $\vdash^* \lambda x{:}\sigma{\to}\tau{\to}\rho.\lambda y{:}\sigma{\to}\tau.\lambda z{:}\sigma.(x\ z)\ y\ z : (\sigma{\to}\tau{\to}\rho){\to}(\sigma{\to}\tau){\to}\sigma{\to}\rho$.

Even with the formulation of $\lambda\!\to$ à la Church in Definition 3.2.1 an important difference with $\lambda\!\to$ à la Curry remains: in Church's system abstractions have *domains*, i.e. are of the form $\lambda x{:}\,\sigma.M$, whereas in Curry's system abstractions have no domain, i.e. are of the form $\lambda x.M$. Thus, in Church's system one writes

$$\lambda x{:}\sigma.x : \sigma \to \sigma,$$

whereas in Curry's system one writes

$$\lambda x.x : \sigma \to \sigma.$$

[1]Strictly speaking, we should proceed as in the case of λ-terms and define a notion of pre-pseudo-terms, then define substitution and α-equivalence on these, and finally adopt the convention that by $M \in \Lambda_\Pi$ we always mean the α-equivalence class, see 1.13–1.19. We omit the details.

[2]In this chapter it is useful to distinguish syntactically between typing in the system à la Church and the system à la Curry, and therefore we use $\vdash^*$ here. In later chapters we shall also use $\vdash$ for $\vdash^*$.

The two different systems—Curry's and Church's—represent two different paradigms in programming languages. In Church's system the programmer has to explicitly write the types for all variables used in the program as in, e.g., Pascal, whereas in Curry's approach the programmer merely writes functions, and it is then the job of the compiler or the programming environment to infer the types of variables, as e.g., in ML and Haskell.

Having introduced a new set of terms (pseudo-terms instead of λ-terms) we are obliged to introduce the notions of substitution, reduction, etc., for the new notion. This is carried out briefly below. We reuse much notation and terminology.

3.2.3. Definition. For $M \in \Lambda_\Pi$ define the set $\mathrm{FV}(M)$ of *free variables* of M as follows.

$$\begin{array}{lcl} \mathrm{FV}(x) & = & \{x\} \\ \mathrm{FV}(\lambda x{:}\sigma.P) & = & \mathrm{FV}(P)\backslash\{x\} \\ \mathrm{FV}(P\,Q) & = & \mathrm{FV}(P) \cup \mathrm{FV}(Q) \end{array}$$

If $\mathrm{FV}(M) = \{\}$ then M is called *closed.*

3.2.4. Definition. For $M, N \in \Lambda_\Pi$ and $x \in V$, the *substitution of N for x in M,* written $M[x := N]$, is defined as follows:

$$\begin{array}{lcll} x[x := N] & = & N & \\ y[x := N] & = & y & \text{if } x \neq y \\ (P\,Q)[x := N] & = & P[x := N]\,Q[x := N] & \\ (\lambda y{:}\sigma.P)[x := N] & = & \lambda y{:}\sigma.P[x := N] & \text{where } x \neq y \text{ and } y \notin \mathrm{FV}(N) \end{array}$$

3.2.5. Definition. Let $\to_\beta$ be the smallest relation on Λ_Π closed under the rules:

$$\begin{array}{lcl} (\lambda x:\sigma\ .\ P)\ Q \to_\beta P[x := Q] & & \\ P \to_\beta P' & \Rightarrow & \forall x \in V, \sigma \in \Pi :\ \lambda x{:}\sigma.P \to_\beta \lambda x{:}\sigma.P' \\ P \to_\beta P' & \Rightarrow & \forall Z \in \Lambda :\ \ P\ Z \to_\beta P'\ Z \\ P \to_\beta P' & \Rightarrow & \forall Z \in \Lambda :\ \ Z\ P \to_\beta Z\ P' \end{array}$$

A term of form $(\lambda x : \sigma\ .\ P)\ Q$ is called a *β-redex,* and $P[x := Q]$ is called its *β-contractum.* A term M is a *β-normal form* if there is no term N with $M \to_\beta N$.

3.2.6. Definition.

(i) The relation $\twoheadrightarrow_\beta$ (*multi-step β-reduction*) is the transitive-reflexive closure of $\to_\beta$;

(ii) The relation $=_\beta$ (*β-equality)* is the transitive-reflexive-symmetric closure of $\to_\beta$.

We end the section by briefly repeating the development in the preceding subsection for simply typed λ-calculus à la Church.

3.2.7. Lemma (Free variables lemma). *Let* $\Gamma \vdash^* M : \sigma$. *Then:*

(i) $\Gamma \subseteq \Gamma'$ *implies* $\Gamma' \vdash^* M : \sigma$;

(ii) $\mathrm{FV}(M) \subseteq \mathrm{dom}(\Gamma)$;

(iii) $\Gamma' \vdash^* M : \sigma$ *where* $\mathrm{dom}(\Gamma') = \mathrm{FV}(M)$ *and* $\Gamma' \subseteq \Gamma$.

Proof. See the Exercises. □

3.2.8. Lemma (Generation lemma).

(i) $\Gamma \vdash^* x : \sigma$ *implies* $x : \sigma \in \Gamma$;

(ii) $\Gamma \vdash^* M\, N : \sigma$ *implies that there is a* τ *such that* $\Gamma \vdash^* M : \tau \to \sigma$ *and* $\Gamma \vdash^* N : \tau$.

(iii) $\Gamma \vdash^* \lambda x{:}\tau.M : \sigma$ *implies that there is a* ρ *such that* $\Gamma, x : \tau \vdash^* M : \rho$ *and* $\sigma = \tau \to \rho$.

Proof. See the Exercises. □

3.2.9. Proposition (Substitution lemma).

(i) *If* $\Gamma \vdash^* M : \sigma$, *then* $\Gamma[\alpha := \tau] \vdash^* M : \sigma[\alpha := \tau]$.

(ii) *If* $\Gamma, x : \tau \vdash^* M : \sigma$ *and* $\Gamma \vdash^* N : \tau$ *then* $\Gamma \vdash^* M[x := N] : \sigma$.

Proof. See the Exercises. □

3.2.10. Proposition (Subject reduction). *If* $\Gamma \vdash^* M : \sigma$ *and* $M \to_\beta N$, *then* $\Gamma \vdash^* N : \sigma$.

Proof. See the Exercises. □

3.2.11. Theorem (Church-Rosser property). *Suppose that* $\Gamma \vdash^* M : \sigma$. *If* $M \twoheadrightarrow_\beta N$ *and* $M \twoheadrightarrow_\beta N'$, *then there exists an* L *such that* $N \twoheadrightarrow_\beta L$ *and* $N' \twoheadrightarrow_\beta L$ *and* $\Gamma \vdash^* L : \sigma$.

Proof. One way to obtain this result is to repeat for Λ_Π an argument similar to that we used for untyped terms, and then use the subject reduction property. Another method, based on so called *logical relations* can be found in [74]. □

The following two properties of simply typed λ-calculus à la Church do not hold for the Curry system. Note that (ii) implies the subject expansion property—see Remark 3.1.10.

3.2.12. PROPOSITION (Uniqueness of types).

(i) *If* $\Gamma \vdash^* M : \sigma$ *and* $\Gamma \vdash^* M : \tau$ *then* $\sigma = \tau$.

(ii) *If* $\Gamma \vdash^* M : \sigma$ *and* $\Gamma \vdash^* N : \tau$ *and* $M =_\beta N$, *then* $\sigma = \tau$.

PROOF.

(i) By induction on M.

(ii) If $M =_\beta N$ then by the Church-Rosser property, $M \twoheadrightarrow_\beta L$ and $N \twoheadrightarrow_\beta L$, for some L. By subject reduction, $\Gamma \vdash^* L : \sigma$ and $\Gamma \vdash^* L : \tau$. Now use (i). □

It is easy to see that these properties fail in $\lambda\!\to$ à la Curry. For instance, $\vdash \lambda x.x : \alpha \to \alpha$ and $\vdash \lambda x.x : (\alpha \to \alpha) \to (\alpha \to \alpha)$ by the derivations:

$$\frac{x : \alpha \vdash x : \alpha}{\vdash \lambda x.x : \alpha \to \alpha}$$

and

$$\frac{x : \alpha\to\alpha \vdash x : \alpha\to\alpha}{\vdash \lambda x.x : (\alpha\to\alpha) \to (\alpha\to\alpha)}$$

Although these two derivations have the same structure, their conclusions are different due to different type assumptions for x. In contrast, if the Church term M has type σ in Γ, then there is exactly one derivation of this fact, which is uniquely encoded by M.

This difference leads to some interesting problems for the Curry system. Given a term M which types can be assigned to M, if any? Is there a single best type in some sense? Such problems are studied in *type inference,* which we return to later.

Because of the above difference, $\lambda\!\to$ à la Curry and other similar systems are often called *type assignment* systems, in contrast to $\lambda\!\to$ à la Church and similar systems which are called, e.g., *typed* systems.

3.3. Church versus Curry typing

Although the simply typed λ-calculus à la Curry and Church are different, one has the feeling that essentially the same thing is going on. To some extent this intuition is correct, as we now show.

Every pseudo-term induces a type-free λ-term by erasing the domains of abstractions.

3.3.1. DEFINITION. The *erasure* map $|\bullet| : \Lambda_\Pi \to \Lambda$ is defined as follows:

$$\begin{array}{rcl} |x| & = & x; \\ |M\,N| & = & |M|\,|N|; \\ |\lambda x{:}\sigma.M| & = & \lambda x.|M|. \end{array}$$

Erasure preserves reduction and typing:

3.3.2. Proposition (Erasing). *Let* $M, N \in \Lambda_\Pi$.

(i) *If* $M \to_\beta N$ *then* $|M| \to_\beta |N|$*;*

(ii) *If* $\Gamma \vdash^* M : \sigma$ *then* $\Gamma \vdash |M| : \sigma$.

Proof. (i) prove by induction on M that

$$|M[x := N]| = |M|[x := |N|] \qquad (*)$$

Then proceed by induction on the derivation of $M \to_\beta N$ using $(*)$.

(ii) by induction on the derivation of $\Gamma \vdash^* M : \sigma$. □

Conversely, one can "lift" every Curry derivation to a Church one.

3.3.3. Proposition (Lifting). *For all* $M, N \in \Lambda$*:*

(i) *If* $M \to_\beta N$ *then for each* $M' \in \Lambda_\Pi$ *with* $|M'| = M$ *there is* $N' \in \Lambda_\Pi$ *such that* $|N'| = N$*, and* $M' \to_\beta N'$*;*

(ii) *If* $\Gamma \vdash M : \sigma$ *then there is an* $M' \in \Lambda_\Pi$ *with* $|M'| = M$ *and* $\Gamma \vdash^* M' : \sigma$.

Proof. By induction on the derivation of $M \to_\beta N$ and $\Gamma \vdash M : \sigma$, respectively. □

3.3.4. Warning. The above two propositions allow one to derive certain properties of Curry-style typable lambda-terms from analogous properties of Church-style typed lambda-terms, or conversely. For instance, strong normalization for one variant of $(\lambda \to)$ easily implies strong normalization for the other (Exercise 3.6.4).

However, one has to be very cautious with such proof methods, sometimes they do not work. A common mistake (cf. Exercise 3.6.5) is the following attempt to derive the Church-Rosser property for Church-style $(\lambda \to)$ from the Church-Rosser property for untyped lambda-terms:

> Assume that $M_0 \twoheadrightarrow_\beta M_1$ and $M_0 \twoheadrightarrow_\beta M_2$. Then, by Proposition 3.3.2, we have $|M_0| \twoheadrightarrow_\beta |M_1|$ and $|M_0| \twoheadrightarrow_\beta |M_2|$. By Church-Rosser property for untyped lambda-terms, we have a term P with $|M_1| \twoheadrightarrow_\beta P$ and $|M_2| \twoheadrightarrow_\beta P$. In addition, by the subject reduction property, P is typable into the desired type. It remains to apply Proposition 3.3.3, to obtain a Church-style term M_3 with $|M_3| = P$, and such that $M_2 \twoheadrightarrow_\beta M_3$ and $M_1 \twoheadrightarrow_\beta M_3$.

For an explanation why the gap in this argument cannot be easily fixed, and how it *can* be fixed, see [74, pp. 269, 559].

In the remainder, when stating properties of simply typed λ-calculus it must always be understood that the result applies to both $\lambda\!\to$ à la Curry and à la Church, except when explicitly stated otherwise.

3.4. Normalization

In this section we are concerned with $\lambda\!\!\to$ à la Church.

A simple type can be regarded as a finite binary tree—this is where the alternative name "finite type" comes from—where all internal nodes are labeled by arrows and all leaves are labeled by type variables. We shall often refer to properties of types expressing them as properties of this tree representation. For instance, the function $h(\tau)$ defined below is just the height of the corresponding tree.

3.4.1. Definition. Define the function $h : \Pi \to \mathbb{N}$ by:

$$\begin{array}{lcl} h(\alpha) & = & 0 \\ h(\tau \to \sigma) & = & 1 + \max(h(\tau), h(\sigma)) \end{array}$$

It is often convenient to write Church style terms (typable pseudo-terms) in such a way that types of some or all subterms are displayed by superscripts, as in e.g., $(\lambda x{:}\tau.P^{\rho})^{\tau\to\rho}R^{\tau}$. Recall that a Church style term can be typed in only one way, provided the context of free variables is known. Thus our labelling is always determined by the term and the context. But the labelling itself is not a part of syntax, just a meta-notation.

The following property is the first non-trivial property of $\lambda\!\!\to$.

3.4.2. Theorem (Weak normalization). *Suppose* $\Gamma \vdash^* M : \sigma$. *Then there is a finite reduction* $M_1 \to_\beta M_2 \to_\beta \ldots \to_\beta M_n \in \mathrm{NF}_\beta$.

Proof. We use a proof idea due independently to Turing and Prawitz.

Define the *height* of a redex $(\lambda x{:}\tau.P^{\rho})R$ to be $h(\tau \to \rho)$. For $M \in \Lambda_\Pi$ with $M \notin \mathrm{NF}_\beta$ define

$$m(M) = (h(M), n)$$

where

$$h(M) = \max\{h(\Delta) \mid \Delta \text{ is a redex in } M\}$$

and n is the number of redex occurrences in M of height $h(M)$. If $M \in \mathrm{NF}_\beta$ we define $h(M) = (0, 0)$.

We show by induction on lexicographically ordered pairs $m(M)$ that if M is typable in $\lambda\!\!\to$ à la Church, then M has a reduction to normal-form.

Let $\Gamma \vdash M : \sigma$. If $M \in \mathrm{NF}_\beta$ the assertion is trivially true. If $M \notin \mathrm{NF}_\beta$, let Δ be the rightmost redex in M of maximal height h (we determine the position of a subterm by the position of its leftmost symbol, i.e., the rightmost redex means the redex which *begins* as much to the right as possible).

Let M' be obtained from M by reducing the redex Δ. The term M' may in general have more redexes than M. But we claim that the number of redexes of height h in M' is smaller than in M. Indeed, the redex Δ has disappeared, and the reduction of Δ may only create new redexes of height less than h. To see this, note that the number of redexes can increase by either copying existing redexes or by creating new ones. Now observe that if a new redex is created then one of the following cases must hold:

1. The redex Δ is of the form $(\lambda x{:}\tau.\ \dots xP^\rho \dots)(\lambda y^\rho.Q^\mu)^\tau$, where $\tau = \rho \to \mu$, and reduces to $\dots (\lambda y^\rho.Q^\mu)P^\rho \dots$. There is a new redex $(\lambda y^\rho.Q^\mu)P^\rho$ of height $h(\tau) < h$.

2. We have $\Delta = (\lambda x{:}\tau.\lambda y{:}\rho.R^\mu)P^\tau$, occurring in the context $\Delta^{\rho\to\mu}Q^\rho$. The reduction of Δ to $\lambda y{:}\rho.R_1^\mu$, for some R_1, creates a new redex $(\lambda y{:}\rho.R_1^\mu)Q^\rho$ of height $h(\rho \to \mu) < h(\tau \to \rho \to \mu) = h$.

3. The last case is when $\Delta = (\lambda x{:}\tau.x)(\lambda y^\rho.P^\mu)$, with $\tau = \rho \to \mu$, and it occurs in the context $\Delta^\tau Q^\rho$. The reduction creates the new redex $(\lambda y^\rho.P^\mu)Q^\rho$ of height $h(\tau) < h$.

The other possibility of adding redexes is by copying. If we have $\Delta = (\lambda x{:}\tau.P^\rho)Q^\tau$, and P contains more than one free occurrence of x, then all redexes in Q are multiplied by the reduction. But we have chosen Δ to be the rightmost redex of height h, and thus all redexes in Q must be of smaller heights, because they are to the right of Δ.

Thus, in all cases $m(M) > m(M')$, so by the induction hypothesis M' has a normal-form, and then M also has a normal-form. □

In fact, an even stronger property than weak normalization holds: if $\vdash^* M : \sigma$, then no infinite reduction $M_1 \to_\beta M_2 \to_\beta \dots$ exists. This property is called *strong normalization* and will be proved later.

The subject reduction property together with the Church-Rosser property and strong normalization imply that reduction of any typable λ-term terminates in a normal form of the same type, where the normal form is independent of the particular order of reduction chosen.

3.5. Expressibility

As we saw in the preceding section, every simply typable λ-term has a normal-form. In fact, one can effectively find this normal-form by repeated reduction of the *leftmost* redex. (These results hold for both the à la Curry and à la Church system.) Therefore one can easily figure out whether two simply typable terms are β-equal: just reduce the terms to their respective normal-forms and compare *them.*

These results should suggest that there will be difficulties in representing all the partial recursive functions and possibly also the total recursive functions by simply typable λ-terms, as we shall now see. In the rest of this section we are concerned with simply typed λ-calculus à la Curry.

3.5.1. Definition. Let

$$\mathbf{int} = (\alpha \to \alpha) \to (\alpha \to \alpha)$$

where α is an arbitrary type variable. A numeric function $f : \mathbb{N}^n \to \mathbb{N}$ is $\lambda\!\to$*-definable* if there is an $F \in \Lambda$ with $\vdash\ F : \mathbf{int} \to \cdots \to \mathbf{int} \to \mathbf{int}$ ($n+1$ occurrences of **int**) such that

$$F\ c_{n_1}\ \dots\ c_{n_m} =_\beta c_{f(n_1,\dots,n_m)}$$

for all $n_1, \dots, n_m \in \mathbb{N}$.

It is natural to investigate which of the constructions from Chapter 1 carry over to the typed setting. This is carried out below.

3.5.2. Lemma. *The constant and projection functions are* $\lambda\!\to$*-definable.*

Proof. See the Exercises. □

3.5.3. Lemma. *The function* $sg : \mathbb{N} \to \mathbb{N}$ *defined by* $sg(0) = 0, sg(m+1) = 1$ *is* $\lambda\!\to$*-definable.*

Proof. See the Exercises. □

3.5.4. Lemma. *Addition and multiplication are* $\lambda\!\to$*-definable.*

Proof. See the Exercises. □

3.5.5. Definition. The class of *extended polynomials* is the smallest class of numeric functions containing the

(i) *projections:* $U_i^m(n_1, \dots, n_m) = n_i$ for all $1 \le i \le m$;

(ii) *constant functions:* $k(n) = k$;

(iii) *signum function:* $sg(0) = 0$ and $sg(m+1) = 1$.

and closed under *addition* and *multiplication*:

(i) *addition:* if $f : \mathbb{N}^k \to \mathbb{N}$ and $g : \mathbb{N}^l \to \mathbb{N}$ are extended polynomials, then so is $(f+g) : \mathbb{N}^{k+l} \to \mathbb{N}$

$$(f+g)(n_1, \dots, n_k, m_1, \dots, m_l) = f(n_1, \dots, n_k) + g(m_1, \dots, m_l)$$

(ii) *multiplication:* if $f : \mathbb{N}^k \to \mathbb{N}$ and $g : \mathbb{N}^l \to \mathbb{N}$ are extended polynomials, then so is $(f \cdot g) : \mathbb{N}^{k+l} \to \mathbb{N}$

$$(f \cdot g)(n_1, \dots, n_k, m_1, \dots, m_l) = f(n_1, \dots, n_k) \cdot g(m_1, \dots, m_l)$$

3.5.6. Theorem (Schwichtenberg). *The* $\lambda\!\to$*-definable functions are exactly the extended polynomials.*

The proof is omitted. One direction follows easily from what has been said already; the other direction is proved in [97].

If one does not insist that numbers be uniformly represented as terms of type **int**, more functions become $\lambda\!\to$-definable—see [35].

3.6. Exercises

3.6.1. EXERCISE. Show that the following λ-terms have no type in $\lambda\!\rightarrow$ à la Curry.

1. $\lambda x.x\,x$;
2. Ω
3. $\mathbf{K}\,\mathbf{I}\,\Omega$;
4. $\mathbf{Y}$;
5. $c_2\,\mathbf{K}$.

3.6.2. EXERCISE. Find terms M and M' and types σ, σ' such that $\vdash\, M : \sigma$, $\vdash\, M' : \sigma'$, $M \twoheadrightarrow_\beta M'$, and not $\vdash\, M : \sigma'$.

3.6.3. EXERCISE. Is the following true? If $M \rightarrow_\beta N$ (where $M, N \in \Lambda$) and $M', N' \in \Lambda_\Pi$ are such that $|M'| = M$, $|N'| = N$ then $M' \rightarrow_\beta N'$.

3.6.4. EXERCISE. Show that strong normalization for $(\lambda \rightarrow)$ à la Curry implies strong normalization for $(\lambda \rightarrow)$ à la Church, and conversely.

3.6.5. EXERCISE. Find the bug in the example argument in Warning 3.3.4.

3.6.6. EXERCISE. Consider the proof of weak normalization. Assume that a given term M is of length n including type annotations. Give a (rough) upper bound (in terms of a function in n) for the length of the normalizing sequence of reductions for M, obtained under the strategy defined in that proof. Can your function be bounded by $\exp_k(n)$, for some k? Can this be done under the assumption that the height of redexes in M is bounded by a given constant h? (Here, $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{exp_k(n)}$.)

3.6.7. EXERCISE. This exercise, and the next one, are based on [35]. Define the *rank* of a type τ, denoted $rk(\tau)$, as follows:

$$\begin{array}{lcl} rk(\alpha) & = & 0 \\ rk(\tau \rightarrow \sigma) & = & \max(h(\tau)+1, h(\sigma)) \end{array}$$

Alternatively, we have

$$rk(\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \alpha) = 1 + \max(h(\tau_1), \ldots, h(\tau_n)).$$

The rank of a redex $(\lambda x{:}\tau.P^\rho)R$ is $rk(\tau \rightarrow \rho)$. Then define the *depth* of a term M, denoted $d(M)$, by the conditions

$$\begin{array}{lcl} d(x) & = & 0; \\ d(MN) & = & 1 + \max(d(M), d(N)); \\ d(\lambda x{:}\sigma.M) & = & 1 + d(M). \end{array}$$

Let r be the maximum rank of a redex occurring in M, and let $d(M) = d$. Show (by induction w.r.t M) that M can be reduced in at most $2^d - 1$ steps to a term M_1 such that the maximum rank of a redex occurring in M_1 is at most $r - 1$, and $d(M_1) \leq 2^d$.

3.6.8. Exercise. Let r be the maximum rank of a redex occurring in M, and let $d(M) = d$. Use the previous exercise to prove that the normal form of M is of depth at most $\exp_r(d)$, and can be obtained in at most $\exp_r(d)$ reduction steps.

3.6.9. Exercise. Show that the constant functions and projection functions are $\lambda{\rightarrow}$-definable.

3.6.10. Exercise. Show that sg is $\lambda{\rightarrow}$-definable.

3.6.11. Exercise. Show that addition and multiplication are $\lambda{\rightarrow}$-definable.

CHAPTER 4

The Curry-Howard isomorphism

Having met one formalism for expressing effective functions—λ-calculus—and another formalism for expressing proofs—natural deduction for intuitionistic logic—we shall now demonstrate an amazing analogy between the two formalisms, known as the *Curry-Howard isomorphism.*

We have already seen several hints that effective functions and proofs should be intimately related. For instance, as mentioned in Chapter 2, the BHK-interpretation [17, 53, 63] states that a proof of an implication $\varphi_1 \to \varphi_2$ is a "construction" which transforms any proof of φ_1 into a proof of φ_2. What is a construction? A possible answer is that it is some kind of effective function. There are several ways to make this answer precise. In this chapter we present one such way; another one is given by Kleene's realizability interpretation, which we present later.

4.1. Natural deduction without contexts

Recall that Chapter 2 presented a so-called *natural deduction* formulation of intuitionistic propositional logic. Such systems were originally introduced by Gentzen [39]. More precisely, Gentzen introduced two kinds of systems, nowadays called *natural deduction* systems and *sequent calculus* systems, respectively. In this chapter we are concerned with the former kind; sequent calculus systems will be introduced in the next chapter.

One of the most significant studies of natural deduction systems after Gentzen's work in the 1930s appears in Prawitz' classical book [85], which is still very readable.

There is an informal way of writing natural deduction proofs. Instead of maintaining explicitly in each node of a derivation the set of assumptions on which the conclusion depends (the *context*), one writes all the assumptions at the top of the derivation with a marker on those assumptions that have been discharged by the implication introduction rule.

Since this style is quite common in the proof theory literature—at least until the Curry-Howard isomorphism became widely appreciated—we also briefly review that notation informally here. Another reason for doing so, is that the notation displays certain interesting problems concerning assumptions that are hidden in our formulation of Chapter 2.

Consider the proof tree:

$$\frac{\dfrac{\varphi \qquad \dfrac{(\varphi \to \psi) \wedge (\varphi \to \rho)}{\varphi \to \psi}}{\psi} \qquad \dfrac{\varphi \qquad \dfrac{(\varphi \to \psi) \wedge (\varphi \to \rho)}{\varphi \to \rho}}{\rho}}{\psi \wedge \rho}$$

First note that, as always, the proof tree is written upside-down. The leaves are the assumptions, and the root is the conclusion, so the proof tree demonstrates that one can infer $\psi \wedge \rho$ from φ and $(\varphi \to \psi) \wedge (\varphi \to \rho)$.

As usual there is an $\to$-introduction rule which discharges assumptions. Thus we are able to infer $\varphi \to \psi \wedge \rho$ from $(\varphi \to \psi) \wedge (\varphi \to \rho)$. Notationally this is done by putting brackets around the assumption in question which is then called *closed,* as opposed to the other assumptions which are called *open:*

$$\frac{\dfrac{\dfrac{[\varphi] \qquad \dfrac{(\varphi \to \psi) \wedge (\varphi \to \rho)}{\varphi \to \psi}}{\psi} \qquad \dfrac{[\varphi] \qquad \dfrac{(\varphi \to \psi) \wedge (\varphi \to \rho)}{\varphi \to \rho}}{\rho}}{\psi \wedge \rho}}{\varphi \to \psi \wedge \rho}$$

Note that the above step discharges *both* occurrences of φ. In general, in an $\to$-introduction step, we may discharge zero, one, or more occurrences of an assumption.

Taking this one step further we get

$$\frac{\dfrac{\dfrac{\dfrac{[\varphi] \qquad \dfrac{[(\varphi \to \psi) \wedge (\varphi \to \rho)]}{\varphi \to \psi}}{\psi} \qquad \dfrac{[\varphi] \qquad \dfrac{[(\varphi \to \psi) \wedge (\varphi \to \rho)]}{\varphi \to \rho}}{\rho}}{\psi \wedge \rho}}{\varphi \to \psi \wedge \rho}}{(\varphi \to \psi) \wedge (\varphi \to \rho) \to \varphi \to \psi \wedge \rho}$$

Since we may decide to discharge only *some* of the occurrences of an open assumption in an $\to$-introduction step, one sometimes adopts for readability the convention of assigning numbers to assumptions, and one then indicates in an $\to$-introduction step which of the occurrences where discharged. In the above example we thus might have the following sequence of proof trees.

First:

$$\dfrac{\dfrac{\varphi^{(1)} \quad \dfrac{(\varphi \to \psi) \wedge (\varphi \to \rho)^{(2)}}{\varphi \to \psi}}{\psi} \qquad \dfrac{\varphi^{(1)} \quad \dfrac{(\varphi \to \psi) \wedge (\varphi \to \rho)^{(2)}}{\varphi \to \rho}}{\rho}}{\psi \wedge \rho}$$

Then by closing both occurrences of φ:

$$\dfrac{\dfrac{\dfrac{[\varphi]^{(1)} \quad \dfrac{(\varphi \to \psi) \wedge (\varphi \to \rho)^{(2)}}{\varphi \to \psi}}{\psi} \qquad \dfrac{[\varphi]^{(1)} \quad \dfrac{(\varphi \to \psi) \wedge (\varphi \to \rho)^{(2)}}{\varphi \to \rho}}{\rho}}{\psi \wedge \rho}}{\varphi \to \psi \wedge \rho^{(1)}}$$

And by closing both occurrences of $(\varphi \to \psi) \wedge (\varphi \to \rho)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{[\varphi]^{(1)} \quad \dfrac{[(\varphi \to \psi) \wedge (\varphi \to \rho)]^{(2)}}{\varphi \to \psi}}{\psi} \qquad \dfrac{[\varphi]^{(1)} \quad \dfrac{[(\varphi \to \psi) \wedge (\varphi \to \rho)]^{(2)}}{\varphi \to \rho}}{\rho}}{\psi \wedge \rho}}{\varphi \to \psi \wedge \rho^{(1)}}}{(\varphi \to \psi) \wedge (\varphi \to \rho) \to \varphi \to \psi \wedge \rho^{(2)}}$$

It is interesting to note that the notation where we indicate which assumption is discharged allows us to distinguish between certain very similar proofs. For instance, in

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\varphi]^{(1)} \quad \dfrac{[(\varphi \to \psi) \wedge (\varphi \to \rho)]^{(2)}}{\varphi \to \psi}}{\psi} \qquad \dfrac{[\varphi]^{(1)} \quad \dfrac{[(\varphi \to \psi) \wedge (\varphi \to \rho)]^{(3)}}{\varphi \to \rho}}{\rho}}{\psi \wedge \rho}}{\varphi \to \psi \wedge \rho^{(1)}}}{(\varphi \to \psi) \wedge (\varphi \to \rho) \to \varphi \to \psi \wedge \rho^{(2)}}}{(\varphi \to \psi) \wedge (\varphi \to \rho) \to (\varphi \to \psi) \wedge (\varphi \to \rho) \to \varphi \to \psi \wedge \rho^{(3)}}$$

and

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\varphi]^{(1)} \quad \dfrac{[(\varphi \to \psi) \wedge (\varphi \to \rho)]^{(2)}}{\varphi \to \psi}}{\psi} \qquad \dfrac{[\varphi]^{(1)} \quad \dfrac{[(\varphi \to \psi) \wedge (\varphi \to \rho)]^{(3)}}{\varphi \to \rho}}{\rho}}{\psi \wedge \rho}}{\varphi \to \psi \wedge \rho^{(1)}}}{(\varphi \to \psi) \wedge (\varphi \to \rho) \to \varphi \to \psi \wedge \rho^{(3)}}}{(\varphi \to \psi) \wedge (\varphi \to \rho) \to (\varphi \to \psi) \wedge (\varphi \to \rho) \to \varphi \to \psi \wedge \rho^{(2)}}$$

we discharge the two occurrences of $(\varphi \to \psi) \wedge (\varphi \to \rho)$ separately, but in different orders.

Similarly,

$$\dfrac{\dfrac{[\varphi]^{(1)}}{\varphi \to \varphi^{(1)}}}{\varphi \to \varphi \to \varphi^{(2)}}$$

and

$$\dfrac{\dfrac{[\varphi]^{(1)}}{\varphi \to \varphi^{(2)}}}{\varphi \to \varphi \to \varphi^{(1)}}$$

are two different proofs of $\varphi \to \varphi \to \varphi$. In the first proof there is first a discharge step in which the single occurrence of φ is discharged, and then a discharge step in which zero occurrences of φ are discharged. In the second proof the order is reversed.

In order to avoid confusion with assumption numbers, we require that if two assumptions φ and ψ have the same number, then φ and ψ are the same formula. Also, when we discharge the assumptions with a given number (i), we require that every assumption with this number actually occur on a branch from the node where the discharging occurs.

In general, the rules for constructing the above proof trees look as follows.

$$\dfrac{\varphi \quad \psi}{\varphi \wedge \psi} \qquad\qquad \dfrac{\varphi \wedge \psi}{\varphi} \qquad \dfrac{\varphi \wedge \psi}{\psi}$$

$$\dfrac{\varphi}{\varphi \vee \psi} \qquad \dfrac{\psi}{\varphi \vee \psi} \qquad \dfrac{\varphi \vee \psi \quad \begin{array}{c}[\varphi]^{(i)}\\ \vdots\\ \rho\end{array} \quad \begin{array}{c}[\psi]^{(j)}\\ \vdots\\ \rho\end{array}}{\rho^{(i,j)}}$$

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

$$\dfrac{\bot}{\varphi}$$

For instance, the upper left rule ($\wedge$-introduction) states that two proof trees ending in roots φ and ψ, respectively, may be joined into a single proof tree by addition of a new root $\varphi \wedge \psi$ with the former roots as children. The $\to$-introduction rule states that one may infer an implication by discharging the assumptions with the label indicated by the step.

As we shall see later in this chapter, there is an interest in proofs of a certain simple form. One arrives at such proofs from arbitrary proofs by means of *proof normalization* rules that eliminate detours in a proof. More concretely, consider the proof tree:

$$\dfrac{\dfrac{\dfrac{[\varphi]^{(1)}}{\varphi \to \varphi^{(1)}} \qquad \dfrac{[\psi]^{(2)}}{\psi \to \psi^{(2)}}}{(\varphi \to \varphi) \wedge (\psi \to \psi)}}{\varphi \to \varphi}$$

The proof tree demonstrates that $\varphi \to \varphi$ is derivable. However, it does so by first showing $\varphi \to \varphi$, then inferring $(\varphi \to \varphi) \wedge (\psi \to \psi)$, and then, finally, concluding $\varphi \to \varphi$. A more direct proof tree, which does not make an excursion via $(\varphi \to \varphi) \wedge (\psi \to \psi)$ is:

$$\dfrac{[\varphi]^{(1)}}{\varphi \to \varphi^{(1)}}$$

Note that the detour in the former proof tree is signified by an introduction rule immediately followed by the corresponding elimination rule, for example $\wedge$-introduction and $\wedge$-elimination. In fact, the above style of detour-elimination is possible whenever an introduction rule is immediately followed by the corresponding elimination rule.

As another example, consider the proof tree:

$$\dfrac{\dfrac{[\varphi]^{(3)}}{\varphi \to \varphi^{(3)}} \qquad \dfrac{\dfrac{[\varphi \to \varphi]^{(1)}}{\psi \to \varphi \to \varphi^{(2)}}}{(\varphi \to \varphi) \to \psi \to \varphi \to \varphi^{(1)}}}{\psi \to \varphi \to \varphi}$$

Here we infer $\psi \to \varphi \to \varphi$ from $\varphi \to \varphi$ and $(\varphi \to \varphi) \to \psi \to \varphi \to \varphi$. The proof of the latter formula proceeds by inferring $\psi \to \varphi \to \varphi$ from the assumption $\varphi \to \varphi$. Since we can *prove* this assumption, we could simply take this proof and replace the assumption $\varphi \to \varphi$ with the proof of this formula:

$$\dfrac{\dfrac{[\varphi]^{(3)}}{\varphi \to \varphi^{(3)}}}{\psi \to \varphi \to \varphi^{(2)}}$$

In general one considers the following proof normalization rules (symmetric cases omitted):

$$\dfrac{\dfrac{\dfrac{\Sigma}{\varphi} \qquad \dfrac{\Pi}{\psi}}{\varphi\wedge\psi}}{\varphi} \quad\to\quad \dfrac{\Sigma}{\varphi}$$

$$\dfrac{\dfrac{\Sigma}{\psi} \qquad \dfrac{\dfrac{\dfrac{[\psi]^{(i)}}{\Pi}}{\varphi}}{\psi\to\varphi^{(i)}}}{\varphi} \quad\to\quad \dfrac{\dfrac{\dfrac{\Sigma}{\psi}}{\Pi}}{\varphi}$$

$$\dfrac{\dfrac{\dfrac{\Theta}{\varphi}}{\varphi\vee\psi} \qquad \dfrac{\dfrac{[\varphi]^{(i)}}{\Sigma}}{\rho} \qquad \dfrac{\dfrac{[\psi]^{(j)}}{\Pi}}{\rho}}{\rho^{(i,j)}} \quad\to\quad \dfrac{\dfrac{\dfrac{\Theta}{\varphi}}{\Sigma}}{\rho}$$

The first rule states that if we, somewhere in a proof, infer φ and ψ and then use $\wedge$-introduction to infer $\varphi \wedge \psi$ followed by $\wedge$-elimination to infer φ, we might as well avoid the detour and replace this proof simply by the subproof of φ.

The second rule states that if we have a proof of φ from assumption ψ and we use this and $\to$-introduction to get a proof of $\psi \to \varphi$, and we have a proof of ψ then, instead of inferring φ by $\to$-elimination, we might as well replace this proof by the original proof of φ where we plug in the proof of ψ in all the places where the assumption ψ occurs.

The reading of the third rule is similar.

The process of eliminating proof detours of the above kind, is called *proof normalization,* and a proof tree with no detours is said to be in *normal form.* Another similar process, called *cut elimination,* eliminates detours in *sequent calculus proofs* whereas proof normalization eliminates detours in natural deduction proofs. Sequent calculus systems are introduced in the next chapter.

Proof normalization and cut elimination were studied in the 1930s by Gentzen, and his studies were continued by several researchers, perhaps most importantly by Prawitz in [85]. Nowadays, proof theory is an independent discipline of logic.

In these notes we shall not consider natural deduction proofs in the above style any further.

4.2. The Curry-Howard isomorphism

We could introduce reductions à la those of the preceding section for the natural deduction formulation of Chapter 2, but we shall not do so. The rules for that formulation are rather tedious (try the rule for $\rightarrow$!). It would be more convenient to have each proof tree denoted by some 1-dimensional expression and then state transformations on such expressions rather than on proof trees. It happens that the terms of the simply typed λ-calculus are ideal for this purpose, as we shall see in this section.

We show that any derivation in intuitionistic propositional logic corresponds to a typable λ-term à la Church, and vice versa. More precisely we show this for the *implicational fragment* of intuitionistic propositional logic.

Recall from Section 2.6 that the implicational fragment is the subsystem in which the only connective is $\rightarrow$ and in which the only rules are $(\rightarrow E)$ and $(\rightarrow I)$. This fragment is denoted IPC($\rightarrow$). The whole system is denoted IPC($\rightarrow, \wedge, \vee, \bot$) or plainly IPC.

If we take PV (the set of propositional variables) equal to U (the set of type variables), then Φ (the set of propositional formulas in the implicational fragment of intuitionistic propositional logic) and Π (the set of simply types) are identical. This will be used implicitly below.

4.2.1. Proposition (Curry-Howard isomorphism).

(i) *If* $\Gamma \vdash M : \varphi$ *then* $|\Gamma| \vdash \varphi$.[1]

(ii) *If* $\Gamma \vdash \varphi$ *then there exists* $M \in \Lambda_\Pi$ *such that* $\Delta \vdash M : \varphi$, *where* $\Delta = \{(x_\varphi : \varphi) \mid \varphi \in \Gamma\}$.

Proof. (i): by induction on the derivation of $\Gamma \vdash M : \varphi$.

(ii): by induction on the derivation of $\Gamma \vdash \varphi$. Let $\Delta = \{x_\varphi : \varphi \mid \varphi \in \Gamma\}$.

1. The derivation is
$$\Gamma, \varphi \vdash \varphi$$
We consider two subcases:
   (a) $\varphi \in \Gamma$. Then $\Delta \vdash x_\varphi : \varphi$.
   (b) $\varphi \notin \Gamma$. Then $\Delta, x_\varphi : \varphi \vdash x_\varphi : \varphi$.

2. The derivation ends in
$$\frac{\Gamma \vdash \varphi \rightarrow \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$$
By the induction hypothesis $\Delta \vdash M : \varphi \rightarrow \psi$ and $\Delta \vdash N : \varphi$, and then also $\Delta \vdash M\,N : \psi$.

[1]Recall that $|\Gamma|$ denotes the range of Γ.

3. The derivation ends in
$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}$$
We consider two subcases:

   (a) $\varphi \in \Gamma$. Then by the induction hypothesis $\Delta \vdash M : \psi$. By Weakening (Lemma 3.19(i)) $\Delta, x : \varphi \vdash M : \psi$, where $x \notin \mathrm{dom}(\Delta)$. Then also $\Delta \vdash \lambda x{:}\varphi \,.\, M : \varphi \to \psi$.

   (b) $\varphi \notin \Gamma$. Then by the induction hypothesis $\Delta, x_\varphi : \varphi \vdash M : \psi$ and then also $\Delta \vdash \lambda x_\varphi{:}\varphi \,.\, M : \varphi \to \psi$. □

4.2.2. Remark. The correspondence displays certain interesting problems with the natural deduction formulation of Chapter 2. For instance

$$\cfrac{\cfrac{x : \varphi, y : \varphi \vdash x : \varphi}{x : \varphi \vdash \lambda y{:}\varphi \,.\, x : \varphi \to \varphi}}{\vdash \lambda x{:}\varphi \,.\, \lambda y{:}\varphi \,.\, x : \varphi \to \varphi \to \varphi}$$

and

$$\cfrac{\cfrac{x : \varphi, y : \varphi \vdash y : \varphi}{x : \varphi \vdash \lambda y{:}\varphi \,.\, y : \varphi \to \varphi}}{\vdash \lambda x{:}\varphi \,.\, \lambda y{:}\varphi \,.\, y : \varphi \to \varphi \to \varphi}$$

are two different derivations in $\lambda{\to}$ showing that both $\lambda x{:}\varphi \,.\, \lambda y{:}\varphi \,.\, x$ and $\lambda x{:}\varphi \,.\, \lambda y{:}\varphi \,.\, y$ have type $\varphi \to \varphi \to \varphi$.

Both of these derivations are projected to

$$\cfrac{\cfrac{\varphi \vdash \varphi}{\varphi \vdash \varphi \to \varphi}}{\vdash \varphi \to \varphi \to \varphi}$$

This reflects the fact that, in the natural deduction system of Chapter 2, one cannot distinguish proofs in which assumptions are discharged in different orders. Indeed, $\lambda{\to}$ can be viewed as an extension of IPC($\to$) in which certain aspects such as this distinction are elaborated.

The correspondence between derivations in IPC($\to$) and $\lambda{\to}$ can be extended to the whole system IPC by extending the simply typed λ-calculus with pairs and disjoint sums. One extends the language Λ_Π with clauses:

$$\begin{array}{lll} \Lambda_\Pi & ::= \ \ldots & \mid \; <\Lambda_\Pi, \Lambda_\Pi> \; \mid \pi_1(\Lambda_\Pi) \mid \pi_2(\Lambda_\Pi) \\ & & \mid \mathrm{in}_1^{\psi\vee\varphi}(\Lambda_\Pi) \mid \mathrm{in}_2^{\psi\vee\varphi}(\Lambda_\Pi) \mid \mathrm{case}(\Lambda_\Pi; V.\Lambda_\Pi; V.\Lambda_\Pi) \end{array}$$

and adds typing rules:

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

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

$$\frac{\Gamma \vdash L : \psi \vee \varphi \quad \Gamma, x : \psi \vdash M : \rho \quad \Gamma, y : \varphi \vdash N : \rho}{\Gamma \vdash \mathrm{case}(L; x.M; y.N) : \rho}$$

and reduction rules:

$$\begin{array}{lcl} \pi_1(< M_1, M_2 >) & \to & M_1 \\ \pi_2(< M_1, M_2 >) & \to & M_2 \\ & & \\ \mathrm{case}(\mathrm{in}_1^{\varphi}(N); x.K; y.L) & \to & K\{x := N\} \\ \mathrm{case}(\mathrm{in}_2^{\varphi}(N); x.K; y.L) & \to & L\{y := N\} \end{array}$$

Intuitively, $\phi \wedge \psi$ is a product type, so $< M_1, M_2 >$ is a pair, and $\pi_1(M)$ is the first projection. In type-free λ-calculus these could be defined in terms of pure λ-terms (see Proposition 1.46), but this is not possible in $\lambda\!\to$. This is related to the fact that one cannot define conjunction in IPC in terms of implication (contrary to the situation in classical logic, as we shall see later).

In the same spirit, $\phi \vee \psi$ is a sum (or "variant") type. A sum type is a data type with two unary constructors. Compare this to the data type "integer list", which is usually defined as a data type with two constructors: the 0-ary constructor *Nil* and the 2-ary constructor $Cons(\bullet, \bullet)$ which takes a number and a list of numbers. In a sum we have the two unary constructors left and right injection.

Thus $\mathrm{case}(M; x.K; y.L)$ is a case-expression which tests whether M has form $\mathrm{in}_1^{\varphi}(N)$ (and then returns K with N for x) or $\mathrm{in}_2(N)$ (and then returns L with N for y), just like in a functional programming language we could have a case-expression testing whether an expression is *Nil* or $Cons(n, ns)$.

Thus, uses of the axiom of intuitionistic propositional logic are reflected by variables in the term, uses of the $\to$-elimination rule correspond to applications, and uses of the $\to$-introduction rule correspond to abstractions.

In fact, we can view $\lambda\!\to$ as a more elaborate formulation of IPC($\to$) in which the terms "record" the rules it was necessary to apply to prove the type of the term, when we view that type as a proposition. For instance, $\lambda x : \varphi \,.\, x$ has type $\varphi \to \varphi$, signifying the fact that we can prove $\varphi \to \varphi$ by first using the axiom recorded by the variable x and then using $\to$-introduction, recorded by $\lambda x : \varphi$. In short, in $\lambda\!\to$ viewed as a logic, the terms serve as a linear representation of proof trees, and are usually called *constructions* [58]. These are also constructions in the sense of the BHK-interpretation: a construction of $\varphi \to \psi$ is a λ-term $\lambda x : \varphi \, . \, M$ of type $\varphi \to \psi$.

Two different propositions cannot have the same construction, since we work with Church terms. In contrast, several constructions may correspond to the same proposition. This is because the same proposition may be proven in different ways.

Thus $\lambda{\to}$ and IPC($\to$) may be viewed as different names for essentially the same thing. This means that each of the concepts and properties considered in $\lambda{\to}$ makes sense in IPC($\to$) and vice versa.

As mentioned, terms in $\lambda \to$ correspond to constructions in IPC($\to$). Types correspond to formulas, type constructors (sum and pair) to connectives. Asking whether there exists a term of a given type (*inhabitation*), corresponds to asking whether there exist a construction for a given proposition (*provability.*) Asking whether there exists a type for a given term (*typability*), corresponds to asking whether the construction is a construction of some formula.

What is a redex in a construction? Well, each introduction rule introduces a *constructor* (a lambda, a pair, or an injection) in the construction, and each elimination rule introduces a *destructor* (an application, a projection, or a case-expression). Now, a redex consists of a constructor immediately surrounded by the corresponding destructor. Therefore a redex in the construction represents a proof tree containing an application of an introduction rule immediately followed by an application of the corresponding elimination rule; this was what we called a detour in a proof tree. Therefore, reduction on terms corresponds to normalization of constructions. A term in normal form corresponds to a construction representing a proof tree in normal form. The subject reduction proposition states that reducing a construction of a formula yields a construction for the same formula. The Church-Rosser Theorem states that the order of normalization is immaterial. Also, it states that we managed to identify essentially identical proofs without identifying all proofs.

In summary:

$\lambda{\to}$	IPC($\to$)
term variable	assumption
term	construction (proof)
type variable	propositional variable
type	formula
type constructor	connective
inhabitation	provability
typable term	construction for a proposition
redex	construction representing proof tree with redundancy
reduction	normalization
value	normal construction

4.2.3. Example. Consider the following example deduction containing redundancy. The original derivation with constructions is:

$$\frac{x : \varphi \vdash x : \varphi}{\vdash \lambda x{:}\varphi \,.\, x : \varphi \to \varphi}$$

The complicated proof with constructions is:

$$\cfrac{\cfrac{\cfrac{y : \psi \vdash y : \psi}{\vdash \lambda y{:}\psi \,.\, y : \psi \to \psi} \qquad \cfrac{x : \varphi \vdash x : \varphi}{\vdash \lambda x{:}\varphi \,.\, x : \varphi \to \varphi}}{\vdash < \lambda x{:}\varphi \,.\, x, \lambda y{:}\psi \,.\, y >: (\varphi \to \varphi) \wedge (\psi \to \psi)}}{\vdash \pi_1(< \lambda x{:}\varphi \,.\, x, \lambda y{:}\psi \,.\, y >) : \varphi \to \varphi}$$

The construction of the latter proof tree in fact contains a redex which upon reduction yields the construction of the former proof tree.

The perfect correspondence between reduction and normalization and the related concepts, justifies the name "isomorphism" rather than simply "bijection." In fact, reduction has been studied extensively in the λ-calculus literature, while normalization has been studied independently in proof theory.

Since the "discovery" of the isomorphism, the two worlds have merged, and some authors feel that it is exactly in the correspondence between reduction and normalization that the isomorphism is deepest and most fruitful. This point of view is supported by the fact that some typed λ-calculi have been introduced as means of studying normalization for logics, most notably Girard's System **F** introduced in his work [44] from 1971. System **F** corresponds to second order minimal propositional logic and will be discussed in Chapter 12.

As an appealing illustration of the isomorphism and an appropriate conclusion of this section, this system was independently invented at roughly the same time in computer science by Reynolds [90] in his study of polymorphism in typed functional programming languages.

In the remainder of these notes the concepts corresponding to one another under the isomorphism are used interchangeably. In particular, any system as that of the preceding subsection will be called both a logic and a λ-calculus depending on the aspects being emphasized.

4.3. Consistency from normalization

A number of properties regarding unprovability can be difficult to establish directly, but more easy to establish by semantical methods as we saw in Chapter 2. Often these semantical methods can be replaced by methods involving the weak normalization property.

The following shows that IPC($\rightarrow$) is consistent.

4.3.1. Proposition. $\nvdash \bot$.

Proof. Assume that $\vdash \bot$. Then $\vdash M : \bot$ for some $M \in \Lambda_\Pi$. By the weak normalization theorem and the subject reduction theorem there is then an $N \in \mathrm{NF}_\beta$ such that $\vdash N : \bot$.

Now, λ-terms in normal form have form $x\ N_1 \dots N_m$ (where $N_1, \dots, N_n$ are normal-forms) and $\lambda x{:}\sigma\ .\ N'$ (where N' is in normal form). We cannot have N of the first form (then $x \in \mathrm{FV}(N)$, but since $\vdash N : \bot$, $\mathrm{FV}(N) = \{\}$). We also cannot have N of the second form (then $\bot = \sigma \rightarrow \tau$ for some σ, τ which is patently false). □

4.4. Strong normalization

As suggested by the application in the preceding section, the weak normalization property of $\lambda\!\rightarrow$ is a very useful tool in proof theory. In this section we prove the *strong normalization* property of $\lambda\!\rightarrow$ which is sometimes even more useful.

The standard method of proving strong normalization of typed λ-calculi was invented by Tait [104] for simply typed λ-calculus, generalized to second-order typed λ-calculus by Girard [44], and subsequently simplified by Tait [105].

Our presentation follows [8]; we consider in this section terms à la Curry.

4.4.1. Definition.

(i) $\mathrm{SN}_\beta = \{M \in \Lambda \mid M \text{ is strongly normalizing }\}$.

(ii) For $A, B \subseteq \Lambda$, define $A \rightarrow B = \{F \in \Lambda \mid \forall a \in A : F\ a \in B\}$.

(iii) For every simple type σ, define $[\![\sigma]\!] \subseteq \Lambda$ by:

$$\begin{array}{lcl} [\![\alpha]\!] & = & \mathrm{SN}_\beta \\ [\![\sigma \to \tau]\!] & = & [\![\sigma]\!] \to [\![\tau]\!] \end{array}$$

4.4.2. Definition.

(i) A set $X \subseteq \mathrm{SN}_\beta$ is *saturated* if

1. For all $n \geq 0$ and $M_1, \dots M_n \in \mathrm{SN}_\beta$:

$$x\, M_1 \dots M_n \in X$$

2. For all $n \geq 1$ and $M_1, \dots, M_n \in \mathrm{SN}_\beta$:

$$M_0\{x := M_1\}\, M_2 \dots M_n \in X \Rightarrow (\lambda x.M_0)\, M_1\, M_2 \dots M_n \in X$$

(ii) $\mathbb{S} = \{X \subseteq \Lambda \mid X \text{ is saturated }\}$.

4.4.3. Lemma.

(i) $\mathrm{SN}_\beta \in \mathbb{S}$;

(ii) $A, B \in \mathbb{S} \Rightarrow A \to B \in \mathbb{S}$;

(iii) $\sigma \in \Pi \Rightarrow [\![\sigma]\!] \in \mathbb{S}$.

Proof. Exercise 4.6.3. □

4.4.4. Definition.

(i) a *valuation* is a map $\rho : V \to \Lambda$, where V is the set of term variables. The valuation $\rho(x := N)$ is defined by

$$\rho(x := N)(y) = \begin{cases} N & \text{if } x \equiv y \\ \rho(y) & \text{otherwise} \end{cases}$$

(ii) Let ρ be a valuation. Then $[\![M]\!]_\rho = M\{x_1 := \rho(x_1), \dots, x_n := \rho(x_n)\}$, where $\mathrm{FV}(M) = \{x_1, \dots, x_n\}$.

(iii) Let ρ be a valuation. Then $\rho \models M : \sigma$ iff $[\![M]\!]_\rho \in [\![\sigma]\!]$. Also, $\rho \models \Gamma$ iff $\rho(x) \in [\![\sigma]\!]$ for all $x : \sigma \in \Gamma$.

(iv) $\Gamma \models M : \sigma$ iff $\forall \rho : \rho \models \Gamma \Rightarrow \rho \models M : \sigma$.

4.4.5. Proposition (Soundness). $\Gamma \vdash M : \sigma \Rightarrow \Gamma \models M : \sigma$.

Proof. By induction on the derivation of $\Gamma \vdash M : \sigma$.

1. The derivation is

$$\Gamma \vdash x : \sigma \qquad x : \sigma \in \Gamma$$

If $\rho \models \Gamma$, then $[\![x]\!]_\rho = \rho(x) \in [\![\sigma]\!]$.

2. The derivation ends in

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}$$

Suppose $\rho \models \Gamma$. By the induction hypothesis $\Gamma \models M : \sigma \to \tau$ and $\Gamma \models N : \sigma$, so $\rho \models M : \sigma \to \tau$ and $\rho \models N : \sigma$, i.e., $[\![M]\!]_\rho \in [\![\sigma]\!] \to [\![\tau]\!]$ and $[\![N]\!]_\rho \in [\![\sigma]\!]$. Then $[\![M\,N]\!]_\rho = [\![M]\!]_\rho\,[\![N]\!]_\rho \in [\![\tau]\!]$, as required.

3. The derivation ends in

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}$$

Suppose $\rho \models \Gamma$. Also, suppose $N \in [\![\sigma]\!]$. Then $\rho(x := N) \models \Gamma, x : \sigma$. By the induction hypothesis $\Gamma, x : \sigma \models M : \tau$, so $\rho(x := N) \models M : \tau$, i.e., $[\![M]\!]_{\rho(x:=N)} \in [\![\tau]\!]$. Now,

$$\begin{array}{rcl} [\![\lambda x.M]\!]_\rho\, N & \equiv & (\lambda x.M)\, \{y_1 := \rho(y_1), \ldots, y_n := \rho(y_n)\}N \\ & \to_\beta & M\{y_1 := \rho(y_1), \ldots, y_n := \rho(y_n), x := N\} \\ & \equiv & [\![M]\!]_{\rho(x:=N)} \end{array}$$

Since $N \in [\![\sigma]\!] \subseteq \mathrm{SN}_\beta$ and $[\![M]\!]_{\rho(x:=N)} \in [\![\tau]\!] \in \mathbb{S}$, it follows that $[\![\lambda x.M]\!]_\rho\, N \in [\![\tau]\!]$. Hence $[\![\lambda x.M]\!]_\rho \in [\![\sigma \to \tau]\!]$. □

4.4.6. Theorem. $\Gamma \vdash M : \sigma \Rightarrow M \in \mathrm{SN}_\beta$.

Proof. If $\Gamma \vdash M : \sigma$, then $\Gamma \models M : \sigma$. For each $x : \tau \in \Gamma$, let $\rho(x) = x$. Then $x \in [\![\tau]\!]$ holds since $[\![\tau]\!] \in \mathbb{S}$. Then $\rho \models \Gamma$, and we have $M = [\![M]\!]_\rho \in [\![\sigma]\!] \subseteq \mathrm{SN}_\beta$. □

The reader may think that the above proof is more complicated than the weak normalization proof of the preceding chapter; in fact, this feeling can be made into a technical property by noting that the latter proof involves quantifying over sets, whereas the former does not.

The fact that the strong normalization property seems more difficult to prove has led to some techniques that aim at inferring strong normalization from weak normalization—see [102].

There are many applications of strong normalization, but many of these applications can be obtained already by using the weak normalization theorem. The following is a true application of strong normalization where weak normalization does not suffice.

4.4.7. Definition. Let $\to$ be a binary relation on some set L, and write $M \twoheadrightarrow M'$ if $M = M_1 \to \ldots \to M_n = M'$, where $n \geq 1$. Then

1. $\to$ satisfies CR iff for all $M_1, M_2, M_3 \in L$, $M_1 \twoheadrightarrow M_2$ and $M_1 \twoheadrightarrow M_3$ implies that there is an $M_4 \in L$ such that $M_2 \twoheadrightarrow M_4$ and $M_3 \twoheadrightarrow M_4$.

2. $\to$ satisfies WCR iff for all $M_1, M_2, M_3 \in L$, $M_1 \to M_2$ and $M_1 \to M_3$ implies that there is an $M_4 \in L$ such that $M_2 \twoheadrightarrow M_4$ and $M_3 \twoheadrightarrow M_4$.

3. $\to$ satisfies SN iff for all $M \in L$, there is no infinite reduction sequence $M \to M' \to \dots$.

4. $\to$ satisfies WN iff for all $M \in L$, there is a finite reduction sequence $M \to M' \to \dots \to M''$ such that M'' is a normal form (i.e., for all $N \in L$: $M'' \not\to N$).

4.4.8. Proposition (Newman's lemma). *Let $\to$ be a binary relation satisfying* SN. *If $\to$ satisfies* WCR, *then $\to$ satisfies* CR.

Proof. See the exercises. □

The following shows that the assumption about strong normalization cannot be replaced by weak normalization.

4.4.9. Proposition. *There is a binary relation $\to$ satisfying* WN *and* WCR, *but not* CR.

Proof. See the exercises. □

4.4.10. Corollary. *Let $M_1 \in \Lambda$ be typable in $\lambda\!\to$ à la Church and assume that $M_1 \twoheadrightarrow_\beta M_2$ and $M_1 \twoheadrightarrow_\beta M_3$. Then there is an M_4 such that $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$.*

Proof. See the exercises. □

4.5. Historical remarks

The informal notion of a "construction" mentioned in the BHK-interpretation was first formalized in Kleene's *recursive realizability* interpretation [60, 61] in which proofs in *intuitionistic number theory* are interpreted as numbers, as we will see later in the notes. A proof of $\varphi_1 \to \varphi_2$ is interpreted as the *Gödel number* of a partial recursive function mapping the interpretation of any proof of φ_1 to the interpretation of a proof of φ_2.

One can see the Curry-Howard isomorphism—the correspondence between systems of formal logic and functional calculi with types, mentioned above—as a syntactic reflection of this interpretation. It shows that a certain notation system for denoting certain recursive functions coincides with a system for expressing proofs.

Curry [24] discovered that the provable formulas in a so-called *Hilbert formulation* of IPC($\to$) coincide with the inhabited types of *combinatory logic,* when one identifies function type with implication. Moreover, every proof in the logic corresponds to a term in the functional calculus, and vice versa. Curry also noted a similar correspondence between a natural deduction formulation of IPC($\rightarrow$) and simply typed λ-calculus, and between a *sequent calculus* formulation of IPC($\rightarrow$) and a sequent calculus version of simply typed λ-calculus.

Gentzen's *Hauptsatz* [39] shows how one can transform a sequent calculus proof into another proof with no applications of the *cut rule.* Curry now proved a corresponding result for the sequent calculus version of simply typed λ-calculus. He then formulated correspondences between sequent calculus systems, natural deduction systems, and Hilbert systems (in terms of the corresponding functional calculi) and used these to infer weak normalization for β-reduction in simply typed λ-calculus and for so-called *strong reduction* in combinatory logic.

A more direct relation between reduction on terms and normalization of proofs was given by Howard in a paper from 1968, published as [58]. Prawitz had studied reduction of natural deduction proofs extensively [85]—seven years after Curry's book—and had proved weak normalization of this notion of reduction. Howard now showed that reduction of a proof in the natural deduction system for minimal implicational logic corresponds to β-reduction on the corresponding term in the simply typed λ-calculus. He also extended this correspondence to first order intuitionistic arithmetic and a related typed λ-calculus.

Howard's correspondence and the weak normalization theorem give a syntactic version of Kleene's interpretation, where one replaces recursive functions by λ-terms in normal form. For instance, any proof of $\varphi_1 \rightarrow \varphi_2$ reduces to a λ-abstraction which, when applied to a proof of φ_1, yields a proof of φ_2.

Constable [19, 20] suggested that a type or proposition φ be viewed as a specification, and any proof M of φ as a program satisfying the specification. For instance, sorting can be specified by the formula

$$\forall x\, \exists y : \mathsf{ordered}(y) \wedge \mathsf{permutation}(x, y)$$

in *predicate logic,* and a proof of the formula will be a sorting algorithm. There is a literature devoted to methods for finding efficient programs in this way.

The Curry-Howard isomorphism has evolved with the invention of numerous typed λ-calculi and corresponding natural deduction logics, see [87, 55, 8, 41]. Other names for the isomorphism include *propositions-as-types, formula-as-types,* and *proofs-as-programs.*

4.6. Exercises

4.6.1. Exercise. Give derivations of the formulas (1),(3),(5),(7),(9),(11) from Section 2.2 using the natural deduction style of Section 4.1.

4.6.2. Exercise. Give λ-terms corresponding to the derivations from Exercise 4.6.1. Use the following rule for λ-terms corresponding to the ex-falso rule:

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

4.6.3. Exercise. Prove Lemma 4.4.3.

4.6.4. Exercise.

1. Prove Newman's Lemma.

   Hint: Prove by induction on the length of the longest reduction sequence from M that $M \twoheadrightarrow M_1$ and $M \twoheadrightarrow M_2$ implies that there is an M_3 such that $M_1 \twoheadrightarrow M_3$ and $M_2 \twoheadrightarrow M_3$.

2. Prove Proposition 4.4.9.

3. Infer from Newman's Lemma Corollary 4.4.10.

4.6.5. Exercise. Prove Proposition 4.2.1(i) in detail.

4.6.6. Exercise. A *β-reduction strategy* is a map $F : \Lambda \to \Lambda$ such that $M \to_\beta F(M)$ if $M \notin \mathrm{NF}_\beta$, and $F(M) = M$ otherwise. Informally, a reduction strategy selects from any term not in normal form a redex and reduces that. For example, F_l is the reduction strategy that always reduces the left-most redex.

A reduction strategy F is *normalizing* if, for any weakly normalizing term M, there is an i such that[2]

$$M \to_\beta F(M) \to_\beta \ldots \to_\beta F^i(M) \in \mathrm{NF}_\beta$$

That is, if the term has a normal form, then repeated application of F eventually ends in the normal form. A classical result due to Curry and Feys states that F_l is normalizing.

A reduction strategy F is *perpetual* if, for any term M which is not strongly normalizing, there is no i such that

$$M \to_\beta F(M) \to_\beta \ldots \to_\beta F^i(M) \in \mathrm{NF}_\beta$$

That is, if the term has an infinite reduction, then repeated application of F yields an infinite reduction sequence.

Define $F_\infty : \Lambda \to \Lambda$ as follows. If $M \in \mathrm{NF}_\beta$ then $F_\infty(M) = M$; otherwise[3]

$$\begin{array}{lcll}
F_\infty(x\, \vec{P}\, Q\, \vec{R}) & = & x\, \vec{P}\, F_\infty(Q)\, \vec{R} & \text{If } \vec{P} \in \mathrm{NF}_\beta, Q \notin \mathrm{NF}_\beta \\
F_\infty(\lambda x.P) & = & \lambda x.F_\infty(P) & \\
F_\infty((\lambda x.P)\ Q\ \vec{R}) & = & P\{x := Q\}\ \vec{R} & \text{If } x \in \mathrm{FV}(P) \text{ or } Q \in \mathrm{NF}_\beta \\
F_\infty((\lambda x.P)\ Q\ \vec{R}) & = & (\lambda x.P)\ F_\infty(Q)\ \vec{R} & \text{If } x \notin \mathrm{FV}(P) \text{ and } Q \notin \mathrm{NF}_\beta
\end{array}$$

[2]As usual, $F^0(M) = M$ and $F^{i+1}(M) = F(F^i(M))$.

[3]By $\vec{P}$ we denote a finite, possibly empty, sequence of terms.

Show that F_∞ is perpetual.

Let Λ_I be the set of all λ-terms M such that any part $\lambda x.P$ of M satisfies $x \in \mathrm{FV}(P)$. For instance, $\mathbf{I} \in \Lambda_I$ and $\Omega \in \Lambda_I$ but $\mathbf{K} = \lambda x.\lambda y.x \notin \Lambda_I$. Show that for any $M \in \Lambda_I$: $M \in \mathrm{WN}_\beta$ iff $M \in \mathrm{SN}_\beta$.

Hint: Compare being weakly normalizing with F_l leading to a normal form, and compare begin strongly normalizing with F_∞ leading to a normal form. What is the relation between F_l and F_∞ on $M \in \Lambda_I$?

Since Λ_I is a subset of Λ, the elements of Λ_I that have a type in $\lambda\!\rightarrow$ (à la Curry) must correspond to a subset of all proofs in IPC($\rightarrow$). Which proofs are these?

CHAPTER 5

Proofs as combinators

In the preceding chapters we have considered various systems of λ-calculi. One rather disturbing aspect of these systems is the role played by bound variables, especially in connection with substitution. In this chapter we consider a system, *combinatory logic,* which is equivalent to λ-calculus in a certain sense, and in which there are no bound variables.

The first section introduces a version of combinatory logic analogous to type-free λ-calculus. The second section presents *simply typed combinatory logic,* analogous to simply typed λ-calculus.

Since simply typed λ-calculus corresponds to the natural deduction formulation of intuitionistic propositional logic via the Curry-Howard isomorphism, and combinatory logic is a variant of λ-calculus, it is natural to expect that simply typed combinatory logic also corresponds to some variant of intuitionistic propositional logic. This variant, traditionally called *Hilbert-style* as opposed to natural deduction style, is introduced in the third section, and the fourth section presents the Curry-Howard isomorphism between the Hilbert-style intuitionistic propositional logic and combinatory logic.

The fifth section studies special cases of the Curry-Howard isomorphism by investigating how certain restrictions in the logic are reflected by restrictions in the functional calculus.

5.1. Combinatory logic

Combinatory logic was invented by Schönfinkel and Curry in the 1920's shortly before Church introduced the lambda-calculus. The idea was to build the foundations of logic on a formal system in which logical formulas could be handled in a *variable-free* manner.

As mentioned in Chapter 1, the systems of combinatory logic and λ-calculus that aimed at providing a foundations of mathematics and logic turned out to be inconsistent, due to the presence of arbitrary fixed-points—see [7, App. B] or [55, Chap. 17]. Nevertheless, one may distinguish a useful subsystem of the original system of combinators dealing only with pure functions, and this system will be called *combinatory logic* below.

The objects of study in combinatory logic are the *combinatory terms.*

5.1.1. Definition. The set $\mathcal{C}$ of *combinatory terms* is defined by the grammar:

$$\mathcal{C} ::= V \mid \mathbf{K} \mid \mathbf{S} \mid (\mathcal{C}\mathcal{C})$$

where V is the same set of variables as used in Λ. The notational conventions concerning parentheses are the same as for lambda-terms.

5.1.2. Definition. The reduction relation $\to_w$ on combinatory terms, called *weak reduction* is defined by the following rules:

- $\mathbf{K}FG \to_w F$;
- $\mathbf{S}FGH \to_w FH(GH)$;
- If $F \to_w F'$ then $FG \to_w F'G$ and $GF \to_w GF'$.

The symbol $\twoheadrightarrow_w$ denotes the smallest reflexive and transitive relation containing $\to_w$, and $=_w$ denotes the least equivalence relation containing $\to_w$.

A *w-normal form* is a combinatory term F such that $F \not\to_w G$, for all combinatory terms G.

5.1.3. Example.

- Let $\mathbf{I} = \mathbf{SKK}$. Then, for all F, we have $\mathbf{I}F \to_w \mathbf{K}F(\mathbf{K}F) \to_w F$.
- The term $\mathbf{SII}(\mathbf{SII})$ reduces to itself.
- Let $\mathbf{W} = \mathbf{SS}(\mathbf{KI})$. Then, for all F, G, we have $\mathbf{W}FG \twoheadrightarrow_w FGG$.
- Let $\mathbf{B} = \mathbf{S}(\mathbf{KS})\mathbf{K}$. Then, for all F, G, H, we have $\mathbf{B}FGH \twoheadrightarrow_w F(GH)$.
- Let $\mathbf{C} = \mathbf{S}(\mathbf{BBS})(\mathbf{KK})$. Then $\mathbf{C}FGH \twoheadrightarrow_w FHG$, for all F, G, H.
- $\mathbf{K}$, $\mathbf{S}$, $\mathbf{K}\ \mathbf{S}$, $\mathbf{S}\ \mathbf{K}$, and $\mathbf{S}\ \mathbf{KK}$ are w-normal forms.

The following gives the first hint that combinatory logic is essentially simpler than λ-calculus in some respects.

5.1.4. Definition. For $F \in \mathcal{C}$ define the set $\mathrm{FV}(F)$ of *free variables* of F by:

$$\begin{array}{lcl}
\mathrm{FV}(x) & = & \{x\}; \\
\mathrm{FV}(H\ G) & = & \mathrm{FV}(H) \cup \mathrm{FV}(G); \\
\mathrm{FV}(\mathbf{S}) & = & \{\}; \\
\mathrm{FV}(\mathbf{K}) & = & \{\}.
\end{array}$$

For $F, G \in \mathcal{C}$ and $x \in V$ define substitution of G for x in F by:

$$\begin{array}{lcl} x\{x := G\} & = & G; \\ y\{x := G\} & = & y \quad \text{if } x \neq y; \\ (H\ E)\{x := G\} & = & H\{x := G\}\ E\{x := G\}; \\ \mathbf{S}\{x := G\} & = & \mathbf{S}; \\ \mathbf{K}\{x := G\} & = & \mathbf{K}. \end{array}$$

Note that there are no bound variables, and no need for renaming in substitutions.

The following is similar to the the Church-Rosser property for λ-calculus.

5.1.5. Theorem (Church-Rosser property). *If $F \twoheadrightarrow_w F_1$ and $F \twoheadrightarrow_w F_2$ then $F_1 \twoheadrightarrow_w G$ and $F_2 \twoheadrightarrow_w G$, for some G.*

Proof. See the exercises. □

One can then infer Corollaries similar to 1.35–37 in Chapter 1.

There is an obvious similarity between terms of lambda-calculus and combinatory terms. A translation

$$(\ )_\Lambda : \mathcal{C} \to \Lambda$$

is easy to define. We just identify **K** and **S** with the corresponding lambda-terms:

5.1.6. Definition.

- $(x)_\Lambda = x$, for $x \in V$;
- $(\mathbf{K})_\Lambda = \lambda xy.x$;
- $(\mathbf{S})_\Lambda = \lambda xyz.xz(yz)$;
- $(FG)_\Lambda = (F)_\Lambda(G)_\Lambda$.

5.1.7. Proposition. *If $F \twoheadrightarrow_w G$ then $(F)_\Lambda \twoheadrightarrow_\beta (G)_\Lambda$.*

Proof. By induction on the derivation of $F \twoheadrightarrow_w G$. □

5.1.8. Remark. It is not in general the case that $(F)_\Lambda \twoheadrightarrow_\beta (G)_\Lambda$ implies $F \twoheadrightarrow_w G$. Counter-example: $(\mathbf{S}\ (\mathbf{K}\ \mathbf{I})\ \mathbf{K})_\Lambda \to_\beta (\mathbf{K})_\Lambda$ but $\mathbf{S}\ (\mathbf{K}\ \mathbf{I})\ \mathbf{K} \not\twoheadrightarrow_w \mathbf{K}$.

It is less obvious how to make a translation backward, because we have to define lambda abstraction without bound variables. One of the possible methods is as follows.

5.1.9. Definition. For each $F \in \mathcal{C}$ and each $x \in V$ we define the term $\lambda^*x.F \in \mathcal{C}$.

- $\lambda^* x.x = \mathbf{I}$;
- $\lambda^* x.F = \mathbf{K}F$, if $x \notin \mathrm{FV}(F)$;
- $\lambda^* x.FG = \mathbf{S}(\lambda^* x.F)(\lambda^* x.G)$, otherwise.

The following shows that the definition of abstraction behaves (partly) as expected.

5.1.10. PROPOSITION. $(\lambda^* x.F)G \twoheadrightarrow_w F\{x := G\}$

PROOF. Exercise 5.6.3. □

Using the operator $\lambda^* x$, we can define a translation

$$(\ )_{\mathcal{C}} : \Lambda \to \mathcal{C}$$

as follows.

5.1.11. DEFINITION.

- $(x)_{\mathcal{C}} = x$, for $x \in V$;
- $(MN)_{\mathcal{C}} = (M)_{\mathcal{C}}(N)_{\mathcal{C}}$;
- $(\lambda x.M)_{\mathcal{C}} = \lambda^* x.(M)_{\mathcal{C}}$.

5.1.12. REMARK. It is natural to expect, dually to Proposition 5.1.7, that one could use Proposition 5.1.10 to prove that

$$M \twoheadrightarrow_\beta N \Rightarrow (M)_{\mathcal{C}} \twoheadrightarrow_w (N)_{\mathcal{C}} \qquad (*)$$

However, this property does *not hold.* For instance $\lambda x.\mathbf{I}\,\mathbf{I} \to_\beta \lambda x.\mathbf{I}$, but $(\lambda x.\mathbf{I}\,\mathbf{I})_{\mathcal{C}} = \mathbf{S}\,(\mathbf{K}\,\mathbf{I})\,(\mathbf{K}\,\mathbf{I}) \not\twoheadrightarrow_w \mathbf{K}\,\mathbf{I} = (\lambda x.\mathbf{I})_{\mathcal{C}}$.

If one attempts to prove $(*)$ by induction on the derivation of $M \twoheadrightarrow_\beta N$, one runs into difficulties in the case $M \to_\beta N \Rightarrow \lambda x.M \to_\beta \lambda x.N$. The problem is that the corresponding principle

$$F \to_w G \Rightarrow \lambda^* x.F \to_w \lambda^* x.G \qquad (\xi)$$

fails. The references at the end of the chapter contain more information about this problem.

The following shows that the translations between Λ and $\mathcal{C}$ are inverses in a weak sense.

5.1.13. PROPOSITION. *For all $M \in \Lambda$, we have $((M)_{\mathcal{C}})_\Lambda =_\beta M$.*

PROOF. Exercise 5.6.4. □

Because of Propositions 5.1.7 and 5.1.13, we can think of $(\ )_\Lambda$ as of a homomorphic embedding of the combinatory logic into the lambda-calculus. In what follows, we often abuse the notation by using the names $\mathbf{S}, \mathbf{K}$, etc. for the λ-terms $(\mathbf{K})_\Lambda$, $(\mathbf{S})_\Lambda$, etc.

The following property is sometimes expressed by saying that $\mathbf{K}$ and $\mathbf{S}$ make a *basis* for untyped λ-calculus.

5.1.14. COROLLARY. *Every closed lambda term M is beta-equal to a term obtained from $\mathbf{K}$ and $\mathbf{S}$ solely by application.*

PROOF. The desired term is $((M)_\mathcal{C})_\Lambda$. □

Unfortunately, the embedding $(\ )_\Lambda : \mathcal{C} \to \Lambda$ is not an isomorphism. Put differently, the left inverse operator $(\ )_\mathcal{C}$ is only a projection (retraction). Indeed, we have already seen that the statement dual to Proposition 5.1.7 fails, and the same holds for the statement dual to Proposition 5.1.13.

5.1.15. EXAMPLE. $((\mathbf{K})_\Lambda)_\mathcal{C} = \mathbf{S}(\mathbf{KK})\mathbf{I} \neq_w \mathbf{K}$.

It follows that "weak" equality is actually a *strong* property!

5.2. Typed combinators

Since combinatory terms can be seen as a subset of lambda-terms, they can also inherit the structure of simply-typed lambda-calculus. Of course, there are two ways to do this.

5.2.1. DEFINITION. Define the typability relation $\vdash$ on $C \times \mathcal{C} \times \Pi$ by:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}$$

$$\frac{}{\Gamma \vdash \mathbf{K} : \sigma \to \tau \to \sigma}$$

$$\frac{}{\Gamma \vdash \mathbf{S} : (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M N : \tau}$$

for all types σ, τ and ρ and arbitrary context Γ.

The other formulation of simply typed combinatory logic uses combinatory terms à la Church.

5.2.2. Definition. Define the set $\mathcal{C}_\Pi$ of *combinatory terms à la Church* by the grammar:

$$\mathcal{C}_\Pi ::= V \mid \mathbf{K}_{\sigma,\tau} \mid \mathbf{S}_{\sigma,\tau,\rho} \mid (\mathcal{C}_\Pi\, \mathcal{C}_\Pi)$$

Define the typability relation $\vdash$ on $C \times \mathcal{C}_\Pi \times \Pi$ by:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}$$

$$\frac{}{\Gamma \vdash \mathbf{K}_{\sigma,\tau}\, \sigma \to \tau \to \sigma}$$

$$\frac{}{\Gamma \vdash \mathbf{S}_{\sigma,\tau,\rho} : (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\, N : \tau}$$

Following the path of Chapter 3 we could derive the usual properties, e.g., the free variables lemma, a version of the generation lemma, and so on, for each of the two combinatory logics with types. In some cases, e.g. the proof of subject reduction, the proof is simpler since reduction does not entail any substitutions, in contrast to the case of β-reduction. Moreover, we might also prove an equivalence result analogous to Propositions 3.3.2–3.3.3. However, for the sake of brevity, we shall not do so.

To distinguish between the typing relation for simply typed combinatory logic and the one for simply typed λ-calculus, we shall use $\vdash_{\mathcal{C}}$ and $\vdash_\Lambda$ for the two, respectively. In the remainder of the notes it will be clear from context whether the typing relations refer to à la Curry or à la Church systems, both in connection with combinatory logic and λ-calculus.

It is not difficult to see that our embedding $(\ )_\Lambda$ preserves types. In addition, the same is true for the translation $(\ )_{\mathcal{C}}$, but this requires the following lemma:

5.2.3. Lemma. *Let $\Gamma, x : \rho \vdash_{\mathcal{C}} F : \tau$. Then $\Gamma \vdash_{\mathcal{C}} \lambda^* x.F : \rho \to \tau$.*

Proof. By induction on F. □

5.2.4. Proposition.

1. *If $\Gamma \vdash_{\mathcal{C}} F : \tau$ then $\Gamma \vdash_\Lambda (F)_\Lambda : \tau$.*
2. *If $\Gamma \vdash_\Lambda M : \tau$ then $\Gamma \vdash_{\mathcal{C}} (M)_{\mathcal{C}} : \tau$.*

Proof. (i): By induction on the derivation of $\Gamma \vdash_{\mathcal{C}} F : \tau$. (ii): By induction on the derivation of $\Gamma \vdash_\Lambda M : \tau$, using Lemma 5.2.3 □

5.2.5. Corollary. *The simply-typed version of the calculus of combinators has the strong normalization property.*

Proof. By strong normalization of simply typed λ-calculus and Proposition 5.1.7. □

5.3. Hilbert-style proofs

Recall from Chapter 2 and 4 that, so far, our formal proofs have been of the *natural deduction* style. Apart from the sequent calculus style of presentation, which will be introduced later, there is yet another style of presentation of logics, known as the *Hilbert-style.* In fact, this is the traditional approach to the definition of a formal proof. A Hilbert-style proof system consists of a set of axioms and only a few proof rules.

Below we describe such a system for the implicational fragment of propositional intuitionistic logic. This system has only one proof rule, called *modus ponens*, which is sometimes translated to English as *"detachment rule"*:

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

There will be two axiom schemes. That is, all formulas that fit the patterns below are considered axioms:

(A1) $\varphi \to \psi \to \varphi$;

(A2) $(\varphi \to \psi \to \vartheta) \to (\varphi \to \psi) \to \varphi \to \vartheta$.

Note that there are in fact infinitely many axioms. But this does not bother us as long as they can be effectively described. Formal proofs in Hilbert-style systems are traditionally defined as sequences of formulas.

5.3.1. Definition. A formal *proof* of a formula φ from a set Γ of assumptions is a a finite sequence of formulas $\psi_1, \psi_2, \ldots, \psi_n$, such that $\psi_n = \varphi$, and for all $i = 1, \ldots, n$, one of the following cases takes place:

- ψ_i is an axiom, or
- ψ_i is an element of Γ, or
- there are $j, \ell < i$ such that $\psi_j = \psi_\ell \to \psi_i$ (i.e., ψ_i is obtained from ψ_j, ψ_ℓ using *modus ponens*).

We write $\Gamma \vdash_H \varphi$ if such a proof exists. The notation $\vdash_H$ obeys the usual conventions.

5.3.2. Example. Here is a proof of $\varphi \to \varphi$ from the empty set.

1. $(\varphi \to (\psi \to \varphi) \to \varphi) \to ((\varphi \to (\psi \to \varphi)) \to (\varphi \to \varphi))$ (axiom A2);
2. $\varphi \to (\psi \to \varphi) \to \varphi$ (axiom A1);
3. $(\varphi \to (\psi \to \varphi)) \to (\varphi \to \varphi)$ (*modus ponens*: detach 2 from 1);
4. $\varphi \to (\psi \to \varphi)$ (axiom A1);

5. $\varphi \to \varphi$ (*modus ponens*: detach 4 from 3);

5.3.3. EXAMPLE. And here is a proof of ϑ from $\{\varphi \to \psi, \psi \to \vartheta, \varphi\}$:

1. $\varphi \to \psi$ (assumption);
2. φ (assumption);
3. ψ (*modus ponens*: detach 2 from 1);
4. $\psi \to \vartheta$ (assumption);
5. ϑ (*modus ponens*: detach 3 from 4).

The following important property of Hilbert-style proof systems is called the *Deduction Theorem.*

5.3.4. PROPOSITION (Herbrand, 1930). $\Gamma, \varphi \vdash_H \psi$ *iff* $\Gamma \vdash_H \varphi \to \psi$.

PROOF. The proof from right to left requires one application of *modus ponens* and weakening. For the other direction, proceed by induction on the size of the proof. □

Note how easy the proof of $\varphi \to \varphi$ becomes with the availability of the deduction theorem, as compared to having to do the direct proof explicitly.

5.3.5. PROPOSITION. *For every* Γ *and* φ*:* $\Gamma \vdash_N \varphi$ *iff* $\Gamma \vdash_H \varphi$.

PROOF. The right to left part is an easy induction. The converse is also easy, using the Deduction Theorem. □

We conclude that our Hilbert-style system is complete in the sense of both Heyting algebras and Kripke models.

5.3.6. THEOREM. $\Gamma \models \varphi$ *iff* $\Gamma \vdash_H \varphi$.

PROOF. Immediate from the completeness for natural deduction and the previous proposition. □

5.3.7. REMARK. By adding axioms to handle the other connectives, one can obtain complete Hilbert-style proof systems for full propositional intuitionistic logic and for classical propositional logic. It is perhaps interesting that a complete proof systems for classical propositional calculus is obtained by adding only the axiom schemes

- $\bot \to \varphi$;
- $((\varphi \to \psi) \to \varphi) \to \varphi$.

(Recall that $\vee$ and $\wedge$ can be defined in classical logic by $\rightarrow$ and $\bot$.)

The following is a version of the Curry-Howard Isomorphism for Hilbert-style proof systems and combinatory logics. We work with combinatory terms à la Church.

5.3.8. PROPOSITION.

(i) *If* $\Gamma \vdash_{\mathcal{C}} F : \varphi$ *then* $|\Gamma| \vdash_H \varphi$.

(ii) *If* $\Gamma \vdash_H \varphi$ *then there exists* $F \in \mathcal{C}$ *such that* $\Delta \vdash_{\mathcal{C}} F : \varphi$, *where* $\Delta = \{(x_\varphi : \varphi) \mid \varphi \in \Gamma\}$.

PROOF. (i): by induction on the derivation of $\Gamma \vdash_{\mathcal{C}} M : \varphi$.
(ii): by induction on the derivation of $\Gamma \vdash_H \varphi$. □

The Curry-Howard isomorphism in the case of Hilbert-style proofs and combinatory terms is realized by a correspondence between proofs and and Church-style combinatory terms. Here we have the following pairs of equivalent notions:

application	*modus ponens*
variable	assumption
constants **K** and **S**	axioms

5.4. Relevance and linearity

Neither intuitionistic nor classical logic have any objections against the axiom scheme $\varphi \rightarrow \psi \rightarrow \varphi$, which expresses the following rule of reasoning: "an unnecessary assumption can be forgotten". This rule is however dubious when we are interested in the *relevance* of assumptions with respect to the conclusion. Logicians and philosophers have studied various variants of intuitionistic logic in which restrictions are made concerning the manipulation of assumptions. The classical references here are [1] and [2], but the idea of relevant logics dates back to early 50's. Hindley [54] attributes the idea to Moh and Church. Just like *no* use of an assumptions may be regarded as a dubious phenomenon, *multiple* use of an assumption may also raise important doubts. The most ancient reference to a logic in which this was taken into account, given by Hindley [54], is a work of Fitch from 1936.

With the Curry-Howard isomorphism at hand, we can easily identify the corresponding fragments of (the implicational fragment of) intuitionistic propositional logic, by characterizing lambda-terms with respect to the number of occurrences of bound variables within their scopes.

5.4.1. Definition.

1. The set of $\lambda\mathbf{I}$-*terms* is defined by the following induction:
   - Every variable is a $\lambda\mathbf{I}$-term;
   - An application MN is a $\lambda\mathbf{I}$-term iff both M and N are $\lambda\mathbf{I}$-terms;
   - An abstraction $\lambda x.M$ is a $\lambda\mathbf{I}$-term iff M is a $\lambda\mathbf{I}$-term and $x \in \mathrm{FV}(M)$.
2. The set of **BCK**-*terms* is defined as follows:
   - Every variable is a **BCK**-term;
   - An application MN is a **BCK**-term iff both M and N are **BCK**-terms, and $FV(M) \cap \mathrm{FV}(N) = \{\}$;
   - An abstraction $\lambda x.M$ is a **BCK**-term iff M is a **BCK**-term.
3. A term is called *linear* iff it is both a $\lambda\mathbf{I}$-term and a **BCK**-term.

Of course, $\lambda\mathbf{I}$-terms correspond to reasoning where each assumption is used *at least once*, but all assumptions are reusable. The **BCK**-terms represent the idea of *disposable* assumptions that are thrown away after a use, so they cannot be reused. A strict control over all assumptions, with each one being used *exactly* once, is maintained in proofs corresponding to linear terms. The three classes of lambda-terms determine three fragments of IPC($\to$):

Relevant logic	$\lambda\mathbf{I}$-terms
BCK-logic	**BCK**-terms
BCI-logic	linear terms

The above table can be taken as a formal definition of these three logics, in that the λ-calculi simply *are* the logics. Below we give more traditional Hilbert-style formulations of the logics.

5.4.2. Definition.

1. The *relevant* propositional calculus is a Hilbert-style proof system with the *modus ponens* as the only rule, and the following axiom schemes:

   A_S) $(\varphi \to \psi \to \vartheta) \to (\varphi \to \psi) \to \varphi \to \vartheta$;

   A_B) $(\psi \to \vartheta) \to (\varphi \to \psi) \to \varphi \to \vartheta$;

   A_C) $(\varphi \to \psi \to \vartheta) \to \psi \to \varphi \to \vartheta$;

   A_I) $\varphi \to \varphi$.

2. The **BCK** propositional calculus is a Hilbert-style proof system with the *modus ponens* as the only rule, and the axiom schemes (A_B) and (A_C) and

   A_K) $\varphi \to \psi \to \varphi$.

3. The **BCI** propositional calculus is a Hilbert-style proof system with the *modus ponens* as the only rule, and the axiom schemes (A_B) and (A_C) and (A_I).

5.4.3. Warning. The expression "linear logic" denotes a system which is a strict extension of the **BCI**-logic. (But linear logic is based on the same principle as **BCI**-logic: every assumption is used exactly once.)

Of course the axioms (A_K) and (A_S) are exactly our axioms (A1) and (A2) of the full IPC($\to$). The other axioms can also be seen as types of combinators (see Example 5.1.3). We have:

- $\mathbf{B} = \lambda xyz.x(yz) : (\psi \to \vartheta) \to (\varphi \to \psi) \to \varphi \to \vartheta$;
- $\mathbf{C} = \lambda xyz.xzy : (\varphi \to \psi \to \vartheta) \to \psi \to \varphi \to \vartheta$.

Clearly, our three logics correspond to fragments of $\mathcal{C}$ generated by the appropriate choices of combinators. This explains the abbreviations $\lambda\mathbf{I}$, **BCK** and **BCI**. The full untyped lambda-calculus is sometimes called $\lambda\mathbf{K}$-calculus.

We have still to justify that the above definitions are equivalent to those obtained by appropriately restricting occurrences of variables in lambda-terms. First we have the obvious part (remember that we identify combinatory terms with their translations via $(\ )_\Lambda$).

5.4.4. Lemma.

1. *The combinators* **S**, **B**, **C**, **I** *are* $\lambda\mathbf{I}$*-terms, and so are all terms obtained from* **S**, **B**, **C**, **I** *by applications;*
2. *The combinators* **B**, **C**, **K** *are* **BCK***-terms, and so are all terms obtained from* **B**, **C**, **K** *by applications;*
3. *The combinators* **B**, **C**, **I** *are* **BCI***-terms, and so are all terms obtained from* **B**, **C**, **I** *by applications.*

Thus, the embedding $(\ )_\Lambda$ translates the appropriate fragments of $\mathcal{C}$ into the appropriate fragments of Λ. But the inverse translation $(\ )_\mathcal{C}$ cannot be used anymore, as it requires **S**, and **K** to be available. We need first to redefine the combinatory abstraction λ^*.

5.4.5. Definition.

1. For each term in $\mathcal{C}$ and each $x \in V$ we define the term $\lambda^\circ x.M \in \mathcal{C}$.
   - $\lambda^\circ x.x = \mathbf{I}$;
   - $\lambda^\circ x.F = \mathbf{K}F$, whenever $x \notin \mathrm{FV}(F)$;
   - $\lambda^\circ x.FG = \mathbf{S}(\lambda^\circ x.F)(\lambda^\circ x.G)$, if $x \in \mathrm{FV}(F) \cap \mathrm{FV}(G)$;
   - $\lambda^\circ x.FG = \mathbf{C}(\lambda^\circ x.F)G$, if $x \in \mathrm{FV}(F)$ and $x \notin \mathrm{FV}(G)$;
   - $\lambda^\circ x.FG = \mathbf{B}F(\lambda^\circ x.G)$, if $x \notin \mathrm{FV}(F)$ and $x \in \mathrm{FV}(G)$.
2. Now define a translation $[\ ]_\mathcal{C} : \Lambda \to \mathcal{C}$, as follows:
   - $[x]_\mathcal{C} = x$, for $x \in V$;
   - $[MN]_\mathcal{C} = [M]_\mathcal{C}[N]_\mathcal{C}$;
   - $[\lambda x.M]_\mathcal{C} = \lambda^\circ x.[M]_\mathcal{C}$.

The translation $[\ ]_\mathcal{C} : \Lambda \to \mathcal{C}$ has all the good properties of $(\ )_\mathcal{C}$. That is, Propositions 5.1.10, 5.1.13 and 5.2.4 remain true. (For the proof note first that $(\lambda^\circ x.F)_\Lambda =_\beta (\lambda^* x.F)_\Lambda$.) In addition we have:

5.4.6. Proposition.

1. *If M is a $\lambda\mathbf{I}$-term, then $[M]_\mathcal{C}$ is built solely from the combinators* $\mathbf{S}$, $\mathbf{B}$, $\mathbf{C}$ *and* $\mathbf{I}$.
2. *If M is a* $\mathbf{BCK}$*-term then $[M]_\mathcal{C}$ is built solely from the combinators* $\mathbf{B}$, $\mathbf{C}$ *and* $\mathbf{K}$.
3. *If M is a linear term then $[M]_\mathcal{C}$ is built solely from the combinators* $\mathbf{B}$, $\mathbf{C}$ *and* $\mathbf{I}$.

Proof. Easy. Uses the following property: $FV([M]_\mathcal{C}) = \mathrm{FV}(M)$. □

It follows that the translation $[\ ]_\mathcal{C}$ can be seen as an embedding of each of the appropriate fragments of $\mathcal{C}$ into the corresponding fragment of simply typed lambda calculus. We can conclude with the following summary:

5.4.7. Theorem.

- *A formula φ is a theorem of relevant logic if and only if it is a type of a $\lambda\mathbf{I}$-term;*
- *A formula φ is a theorem of* $\mathbf{BCK}$*-logic if and only if it is a type of a* $\mathbf{BCK}$*-term.*
- *A formula φ is a theorem of* $\mathbf{BCI}$*-logic if and only if it is a type of a linear term.*

Proof. Immediate from Proposition 5.4.6, and the appropriate modifications of Propositions 5.1.13 and 5.2.4. □

5.5. Historical remarks

Combinatory logic was introduced by Curry in some early papers [21, 22, 23] and is also studied at length in some newer books [24, 25], which are still very readable.

Hilbert-style proofs are often used in text books on logic that are not concerned with proof normalization in particular, or proof theory in general. It is interesting to note that the deduction theorem, which provides a translation from natural deduction proofs to Hilbert-style proofs, and the abstraction operator, which provides a translation from typed combinatory terms to typed λ-terms, were discovered independently in work on Hilbert systems and work on combinatory logic, although they are essentially the same result. This is just one example of a number of results that have been discovered independently in work on logical systems and functional calculi.

Incidentally, the correspondence between β-equality and weak equality is not as tight as one might hope for. However, if one adds a certain—somewhat involved—set of rules, called A_β, to the rules for weak equality, the resulting relation is equivalent to β-equality in the sense that the above translations between Λ and $\mathcal{C}$ preserve equality and are each other's inverses. In particular, the extended equality on combinators is closed under rule (ξ).

The correspondence between β-equality and weak equality is more elegant in the *extensional* versions of these calculi. More precisely, if one adds the principle of extensionality

$$P\,x =_\beta P'\,x \;\;\&\;\; x \notin \mathrm{FV}(P\,P') \;\Rightarrow\; P =_\beta P' \qquad (\mathrm{ext})_\beta$$

to $=_\beta$ and the similar principle $(\mathrm{ext})_w$ to weak equality, then the resulting calculi are equivalent in the above sense.

Adding rule (ext) to $=_\beta$ is equivalent to adding so-called *η-equality* (see Chapter 6), and adding rule (ext) to $=_w$ is equivalent to adding a certain set of equational axioms, called $A_{\beta\eta}$. More about all this can be found, e.g., in [7].

5.6. Exercises

5.6.1. Exercise. Find a combinator $\mathbf{2} \in \mathcal{C}$ such that $\mathbf{2}FA \twoheadrightarrow_w F(FA)$, for all F and A in $\mathcal{C}$.

5.6.2. Exercise. Prove the Church-Rosser property for weak reduction using the Tait & Martin-Löf method from Chapter 1. Note that the proof for combinatory logic is somewhat simpler due to the fact that *non-overlapping redexes* remain non-overlapping during reduction of other redexes.

5.6.3. Exercise. Prove Proposition 5.1.10.

5.6.4. Exercise. Prove Proposition 5.1.13.

5.6.5. Exercise. Give a Hilbert-style proof of the formula
$(\varphi \to \psi) \to (\psi \to \vartheta) \to \varphi \to \vartheta$.

5.6.6. Exercise. Give a detailed proof of the Deduction Theorem. Use your proof to give an abstraction operator, like λ^*.

5.6.7. Exercise. Describe a notion of reduction for Hilbert-style proofs, corresponding to weak reduction on combinatory terms.

5.6.8. Exercise. Consider the following variant of the calculus of combinators: there are typed constants $\mathbf{K}_{\sigma,\tau}$, and $\mathbf{S}_{\tau,\tau,\tau}$, with typing and reduction rules as usual, and in addition, there are additional constants $\mathbf{I}_\tau : \tau \to \tau$ with the reduction rule $\mathbf{I}_\tau F \to F$. (The identity combinator cannot now be defined as **SKK** because not all typed forms of **S** are available.) By embedding into Church-style combinatory logic, show that this variant satisfies subject reduction and strong normalization properties.

5.6.9. Exercise. (Based on [16]) Consider the terms : $\mathbf{K} = \lambda xy.x$, $\mathbf{S}^\circ = \lambda ixyz.i(i((x(iz))(i(y(iz)))))$ and $\mathbf{I} = \lambda x.x$. Show that these terms form a basis for lambda-calculus in the sense of Corollary 5.1.14, but their types (whatever choice is taken) do not make a complete Hilbert-style axiom system for IPC($\to$).
Hint: One cannot derive the formula $(p \to p \to q) \to p \to q$.

5.6.10. Exercise. Adopt your solutions of Exercises 5.6.1 and 5.6.3 to the case of the translation $[\ ]_C$ of Section 5.4.

CHAPTER 6

Type-checking and related problems

In this chapter we discuss some decision problems related to simply-typed lambda calculus and intuitionistic propositional logic. We are mostly interested in decision problems arising from the analysis of the ternary predicate "$\Gamma \vdash M : \tau$" in the Curry-style version of simply-typed lambda calculus. But the following definition makes sense for every type-assignment system deriving judgements of this form (including Church-style systems).

6.0.11. DEFINITION.

1. The *type checking* problem is to decide whether $\Gamma \vdash M : \tau$ holds, for a given context Γ, a term M and a type τ.
2. The *type reconstruction* problem, also called *typability* problem, is to decide, for a given term M, whether there exist a context Γ and a type τ, such that $\Gamma \vdash M : \tau$ holds, i.e., whether M is typable.
3. The *type inhabitation* problem, also called *type emptiness* problem, is to decide, for a given type τ, whether there exists a closed term M, such that $\vdash M : \tau$ holds. (Then we say that τ is *non-empty* and has an *inhabitant* M).

An obvious motivation to consider type-checking and type reconstruction problems comes of course from programming language design, especially related to the language ML, see [72, 27]. But there were earlier results concerning this problem, due to Curry, Morris and Hindley. See [54, pp. 33–34], for historical notes.

If we look at the type reconstruction problem from the point of view of the Curry-Howard isomorphism, it becomes a problem of determining whether a given "proof skeleton" can be actually turned into a correct proof by inserting the missing formulas. It may be surprising that this kind of questions are sometimes motivated by proof-theoretic research. As noted in [54, pp. 103–104], the main ideas of a type-reconstruction algorithm can be traced as far as the 20's.[1] See [114], for a fresh work, where the "skeleton instantiation" problem is discussed, without any relation to types.

For the type inhabitation problem, the Curry-Howard isomorphism gives an immediate translation into the language of logic:

6.0.12. PROPOSITION. *The type inhabitation problem for the simply-typed lambda calculus is recursively equivalent to the validity problem in the implicational fragment of intuitionistic propositional logic.*

PROOF. Obvious. □

The above proposition remains true for other typed languages, for which the Curry-Howard isomorphism makes sense.

From a programmer's point of view, the type inhabitation problem can be seen as follows: An empty type (a type which cannot be assigned to any term) means a specification that cannot be fulfilled by any program phrase. Solving type inhabitation means (in the contexts of modular programming) the ability to rule out such specifications at compile time.

6.1. Hard and complete

This short section is to recall a few basic notions from complexity theory. The reader is referred to standard textbooks, like [57], for a more comprehensive discussion.

6.1.1. DEFINITION. The notation LOGSPACE, PSPACE and P, refers respectively to the classes of languages (decision problems) solvable by deterministic Turing Machines in logarithmic space, polynomial space, and polynomial time (measured w.r.t. the input size).

6.1.2. DEFINITION. We say that a language L_1 is *reducible* to a language L_2 *in logarithmic space* (or LOGSPACE-*reducible*) iff there is a Turing Machine, that works in logarithmic space (we count only the work tapes, not the input or output tapes) and computes a total function f, such that

$$w \in L_1 \quad \text{iff} \quad f(w) \in L_2,$$

for all inputs w. Two languages are LOGSPACE-*equivalent* iff there are LOGSPACE-reductions each way.

That is, to decide if $w \in L_1$ one can ask if $f(w) \in L_2$, and the cost of the translation is only shipping and handling. Note that a logarithmic space reduction takes at most polynomial time, so this notion is slightly more general than that of a polynomial time reduction.

[1]The good old Polish school again ...

6.1.3. Definition. We say that a language L is *hard* for a complexity class $\mathcal{C}$, iff every language $L' \in \mathcal{C}$ is reducible to L in logarithmic space. If we have $L \in \mathcal{C}$ in addition, then we say that L is *complete* in the class $\mathcal{C}$, or simply $\mathcal{C}$*-complete*.

6.2. The 12 variants

The type reconstruction problem is often abbreviated by "$? \vdash M : ?$", and the type inhabitation problem is written as "$\vdash M : ?$". This notation naturally suggests other related problems, as one can choose to replace various parts of our ternary predicate by question marks, and choose the context to be empty or not. A little combinatorics shows that we have actually 12 problems. Out of these 12 problems, four are completely trivial, since the answer is always "yes":

- $? \vdash ? : ?$;
- $\Gamma \vdash ? : ?$;
- $\vdash ? : ?$;
- $? \vdash ? : \tau$.

Thus we end up with eight non-trivial problems, as follows:

1) $\Gamma \vdash M : \tau$ (type checking);
2) $\vdash M : \tau$ (type checking for closed terms);
3) $? \vdash M : \tau$ (type checking without context);
4) $? \vdash M : ?$ (type reconstruction);
5) $\vdash M : ?$ (type reconstruction for closed terms);
6) $\Gamma \vdash M : ?$ (type reconstruction in a context);
7) $\vdash ? : \tau$ (inhabitation);
8) $\Gamma \vdash ? : \tau$ (inhabitation in a context).

Most of these problems can easily be shown LOGSPACE-equivalent to one of our three main problems: (1), (4) or (7). Some of these LOGSPACE reductions are just inclusions. Indeed, problem (2) is a special case of (1) and of (3), problem (5) is a special case of (4) and (6), and problem (7) is a special case of (8). Others are the subject of Exercises 6.8.1 and 6.8.3. An exception is problem (3). Problems (1), (2) and (4)–(6) reduce to (3) in logarithmic space, but we do not know of any simple LOGSPACE reduction the other way.[2] However, such reductions exists between all problems (1)–(6), because all they turn out to be P-complete.

Let us make one more remark here: On a first look it might seem that determining whether a given term has a given type in a given environment could be easier than determining whether it has any type at all. This impression however is generally wrong, as type reconstruction is easily reducible to type checking, see Exercise 6.8.1. This reduction is "generic", i.e., it works for all reasonable typed calculi.

It is quite unlikely to have a reduction from (7) or (8) to any of (1)–(6), because the inhabitation problems are PSPACE-complete, and that would imply P = PSPACE. We do not know about a simple reduction the other way.

6.3. (First-order) unification

The following is a general definition of (first-order) unification. In what follows we will need only a special case, where the first-order signature is fixed to consist of only one symbol: the binary function symbol "$\to$".

6.3.1. Definition.

1. A *first-order signature* is a finite family of function, relation and constant symbols. Each function and relation symbol comes with a designated non-zero arity. (Constants are sometimes treated as zero-ary functions.) In this section we consider only *algebraic signatures*, i.e., signatures without relation symbols.

2. An *algebraic term* over a signature Σ, or just *term* is either a variable or a constant in Σ, or an expression of the form $(ft_1 \dots t_n)$, where f is an n-ary function symbol, and $t_1, \dots, t_n$ are algebraic terms over Σ.[3] We usually omit outermost parentheses.

The formal definition of an algebraic term involves a prefix application of function symbols. Of course, there is a tradition to write some binary function symbols in the infix style, and we normally do it this way. Our most beloved signature is one that has the (infix) arrow as the only symbol. It is not difficult to see that algebraic terms over this signature can be identified with simple types, or with implicational formulas if you prefer.

In general, algebraic terms over Σ can be identified with finite labelled trees satisfying the following conditions:

- Leaves are labelled by variables and constant symbols;

[2] A solution was given by Henning Makholm, see Chapter 14, Exercise 6.8.3.

[3] Do not confuse algebraic terms with lambda-terms.

- Internal nodes with n daughters are labelled by n-ary function symbols.

6.3.2. Definition.

1. An *equation* is a pair of terms, written "$t = u$". A *system of equations* is a finite set of equations. Variables occurring in a system of equations are called *unknowns*.
2. A *substitution* is a function from variables to terms which is the identity almost everywhere. Such a function S is extended to a function from terms to terms by $S(ft_1 \dots t_n) = fS(t_1) \cdots S(t_n)$ and $S(c) = c$.[4]
3. A substitution S is a solution of an equation "$t = u$" iff $S(t) = S(u)$ (meaning that $S(t)$ and $S(u)$ is the same term). It is a solution of a system E of equations iff it is a solution of all equations in E.

Thus, for instance, the equation $f(gxy)x = fz(fyy)$ has a solution S with $S(x) = fyy$, $S(y) = y$ and $S(z) = g(fyy)y$ (and many other solutions too), while the equation $f(gxy)c = fz(fyy)$, where c is a constant, has no solution. This is because no substitution can turn fyy into c. Another example with no solution is $f(gxy)x = fx(fyy)$, but this time the reason is different: if S were a solution then $S(x)$ would be a proper subterm of itself.

The problem of determining whether a given system of equations has a solution is called the *unification problem*. It is not difficult to see that there is no loss of generality in considering single equations rather than systems of equations (Exercise 6.8.5). The unification algorithm (we say "the", because all these algorithms are actually based on similar ideas, and differ only in details) is due to J.A. Robinson [93], and was motivated by the first-order resolution rule. But, as pointed out by Hindley [54, pp. 43–44], there were also earlier works. Discussions of unification algorithms can be found in various textbooks, for instance in [74].

We choose to sketch a version of the algorithm that is "algebraic" in style. For this, we need the folllowing definition.

6.3.3. Definition.

1. A system of equations is in a *solved form* iff it has the following properties:
   - All equations are of the form "$x = t$", where x is a variable;
   - A variable that occurs at a left-hand side of an equation does not occur at the right-hand side of any equation;
   - A variable may occur in only one left-hand side.

[4]Thus, a substitution is a valuation in the algebra of all terms over Σ.

A variable not occurring as a left-hand side of any equation is called *undefined*.

2. A system of equations is *inconsistent* iff it contains an equation of either of the forms:

   - "$gu_1 \dots u_p = ft_1 \dots t_q$", where f and g are two different function symbols;
   - "$c = ft_1 \dots t_q$", or "$ft_1 \dots t_q = c$", where c is a constant symbol and f is an n-ary function symbol;
   - "$c = d$", where c and d are two different constant symbols;
   - "$x = ft_1 \dots t_q$", where x is a variable, f is an n-ary function symbol, and x occurs in one of $t_1, \dots, t_q$.

3. Two systems of equations are *equivalent* iff they have the same solutions.

It is easy to see that an inconsistent system has no solutions and that a solved system E has a solution S_0 defined as follows:

- If a variable x is undefined then $S_0(x) = x$;
- If "$x = t$" is in E, then $S_0(x) = t$.

The core of Robinson's algorithm can be seen as follows:

6.3.4. LEMMA. *For every system E of equations, there is an equivalent system E' which is either inconsistent or in a solved form. In addition, the system E' can be obtained by performing a finite number of the following operations:*

a) Replace "$x = t$" and "$x = s$" (where t is not a variable) by "$x = t$" and "$t = s$";

b) Replace "$t = x$" by "$x = t$";

c) Replace "$ft_1 \dots t_n = fu_1 \dots u_n$" by "$t_1 = u_1$", ... , "$t_n = u_n$";

d) Replace "$x = t$" and "$r = s$" by "$x = t$" and "$r[x := t] = s[x := t]$";

e) Remove an equation of the form "$t = t$".

PROOF. As long as our system is not solved, and not inconsistent, we can always apply one of the operations (a)–(d). We leave it as Exercise 6.8.8 to show that this process terminates (unless (b) or (d) is used in a silly way). □

6.3.5. Corollary. *The unification problem is decidable.* □

In fact, the above algorithm can be optimized to work in polynomial time (Exercise 6.8.10), provided we only need to check whether a solution exists, and we do not need to *write it down* explicitly, cf. Exercise 6.8.6. The following result is from Dwork *et al* [33].

6.3.6. Theorem. *The unification problem is P-complete with respect to Log-space reductions.* □

Suppose that we have a system of equations E, which is transformed to an equivalent solved system E'. The solution S_0 of E' defined as above, is a most general solution of E' and E, because every other solution must be a specialization of S_0. Formally we have the following definition.

6.3.7. Definition.

- If P and R are substitutions then $P \circ R$ is a substitution defined by $(P \circ R)(x) = P(R(x))$.
- We say that a substitution S is an *instance* of another substitution R (written $R \leq S$) iff $S = P \circ R$, for some substitution P.
- A solution R of a system E is *principal* iff the following equivalence holds for all substitutions S:

$$S \text{ is a solution of } E \quad \text{iff} \quad R \leq S.$$

6.3.8. Proposition. *If a system of equations has a solution then it has a principal one.*

Proof. For a given system of equations E, let E' be an equivalent system in a solved form, and let S_0 be as described above. Then S_0 is a principal solution of E. □

6.4. Type reconstruction algorithm

We now show how type-reconstruction can be reduced to unification. This is a LOGSPACE-reduction, and it can easily be modified to work for all the problems (1)–(6) of Section 6.2. Since there is also a LOGSPACE-reduction the other way, the main result of this section may be stated as:

6.4.1. Theorem. *Type-reconstruction in simply-typed lambda calculus is P-complete.*

The first work where this result was *explicitly* stated was probably the M.Sc. Thesis of Jurek Tyszkiewicz [110]. Our proof of the theorem consists of two reductions. The first one is from type-reconstruction to unification.

Let M be a lambda-term. Choose a representative of M so that no bound variable occurs free in M and no bound variable is bound twice. In what follows we work with this representative rather than with M as an equivalence class. By induction on the construction of M, we define:

- a system of equations E_M (over the signature consisting only of the binary function symbol $\to$);
- a type τ_M.

Of course, the idea is as follows: E_M will have a solution iff M is typable, and τ_M is (informally) a pattern of a type for M.

6.4.2. Definition.

- If M is a variable x, then $E_M = \{\}$ and $\tau_M = \alpha_x$, where α_x is a fresh type variable.
- If M is an application PQ then $\tau_M = \alpha$, where α is a fresh type variable, and $E_M = E_P \cup E_Q \cup \{\tau_P = \tau_Q \to \alpha\}$.
- If M is an abstraction $\lambda x.P$, then $E_M = E_P$ and $\tau_M = \alpha_x \to \tau_P$.

It is not difficult to see that the above construction can be done in logarithmic space. The main property of our translation is as follows.

6.4.3. Lemma.

1. *If $\Gamma \vdash M : \rho$, then there exists a solution S of E_M, such that $\rho = S(\tau_M)$ and $S(\alpha_x) = \Gamma(x)$, for all variables $x \in FV(M)$.*
2. *Let S be a solution of E_M, and let Γ be such that $\Gamma(x) = S(\alpha_x)$, for all $x \in FV(M)$. Then $\Gamma \vdash M : S(\tau_M)$.*

Proof. Induction with respect to M. □

It follows that M is typable iff E_M has a solution. But E_M has then a principal solution, and this has the folllowing consequence. (Here, $S(\Gamma)$ is a context such that $S(\Gamma)(x) = S(\Gamma(x))$.)

6.4.4. Definition. A pair (Γ, τ), consisting of a context (such that the domain of Γ is $FV(M)$) and a type, is called the *principal pair* for a term M iff the following holds:

- $\Gamma \vdash M : \tau$;
- If $\Gamma' \vdash M : \tau'$ then $\Gamma' \supseteq S(\Gamma)$ and $\tau' = S(\tau)$, for some substitution S.

(Note that the first condition implies $S(\Gamma) \vdash M : S(\tau)$, for all S.) If M is closed (in which case Γ is empty), we say that τ is the *principal type* of M.

6.4.5. COROLLARY. *If a term M is typable, then there exists a principal pair for M. This principal pair is unique up to renaming of type variables.*

PROOF. Immediate from Proposition 6.3.8. □

6.4.6. EXAMPLE.

- The principal type of **S** is $(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$. The type $(\alpha \to \beta \to \alpha) \to (\alpha \to \beta) \to \alpha \to \alpha$ can also be assigned to **S**, but it is not principal.
- The principal type of all the Church numerals is $(\alpha \to \alpha) \to \alpha \to \alpha$. But the type $((\alpha \to \beta) \to \alpha \to \beta) \to (\alpha \to \beta) \to \alpha \to \beta$ can also be assigned to each numeral.

To complete the proof of Theorem 6.4.1, we should also give the other reduction. We only give a brief hint of how this should be done. First we reduce the general case of unification to our special arrow-only case (Exercise 6.8.7) and one equation only (Exercise 6.8.5). Then, for a given equation "$\tau = \sigma$", we consider the term $\lambda xy.x(yt_\tau)(yt_\sigma)$, where x, y are new variables and terms t_τ and t_σ are as in Exercise 6.8.2.

6.5. Eta-reductions

We cannot hide this from the reader any more—we finally have to confess that other notions of reduction than beta-reduction are considered in lambda-calculi. In particular, we have *eta-reduction*, based on the following principle:

$$\lambda x.Mx \to_\eta M, \quad \text{whenever } x \notin FV(M).$$

Formally, we have the following definition:

6.5.1. DEFINITION. We define the relation $\to_\eta$ as the least relation satisfying

- If $x \notin FV(M)$ then $\lambda x.Mx \to_\eta M$;
- If $P \to_\eta P'$ then $\lambda x.P \to_\eta \lambda x.P'$;
- If $P \to_\eta P'$ then $PQ \to_\eta P'Q$ and $QP \to_\eta QP'$.

The notation $\twoheadrightarrow_\eta$ and $=_\eta$ has the expected meaning. We also use the symbols $\to_{\beta\eta}$, $\twoheadrightarrow_{\beta\eta}$, $=_{\beta\eta}$ to refer to reductions combining β- and η-reductions.

The motivation for this notion of reduction (and equality) is as follows: Two functions should be considered equal if and only if they return equal results for equal arguments. Indeed, we have

6.5.2. PROPOSITION. *Let $=_{ext}$ be the least equivalence relation such that:*

- *If $M =_\beta N$ then $M =_{ext} N$;*
- *If $Mx =_{ext} Nx$, and $x \notin FV(M) \cup FV(M)$ then $M =_{ext} N$;*
- *If $P =_{ext} P'$ then $PQ =_{ext} P'Q$ and $QP =_{ext} QP'$.*

Then $=_{ext}$ is identical to $=_{\beta\eta}$.

PROOF. Exercise 6.8.15. Note that the above definition does not contain the condition "If $P =_{ext} P'$ then $\lambda x.P =_{ext} \lambda x.P'$". This is not a mistake, because this property (called "rule ξ") follows from the others. □

We do not take $\rightarrow_\eta$ as our standard notion of reduction, because we do not want our calculus of functions to be extensional. After all, we want to be able to distinguish between two algorithms, even if their input-output behaviour is the same.

The notions of eta- and beta-eta-reduction and equality have most of the good properties of beta-reduction and equality. In particular, the Church-Rosser theorem remains true, leftmost reductions are normalizing,[5] and on the typed level we still have both subject reduction and strong normalization properties. (Note that strong normalization for eta alone is immediate, as each eta-reduction step reduces the length of a term.) Of course, eta-reduction makes sense also for the Church-style typed terms.

6.5.3. DEFINITION. We now define the notion of a Church-style term in an *η-long normal form* (or just *long normal form*). Recall that we write M^σ as an informal way of saying that the Church-style term M has type σ in some fixed context. The definition is by induction:

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

Thus a term in a long normal form is a term in normal form where all function variables are "fully applied" to arguments.

6.5.4. LEMMA. *For every Church-style term M^σ in beta normal form there exists a term M_1^σ in η-long normal form, such that $M_1^\sigma \twoheadrightarrow_\eta M$.*

PROOF. Easy. □

[5]But the proof is somewhat more involved than for beta-reduction.

6.6. Type inhabitation

In this section we prove a result of Statman [103], that the inhabitation problem for the finitely-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. The decidability was already known to to Gentzen in 1935, and we will discuss his (syntactic) proof in the next chapter. There are also semantic methods based on the existence of finite models.

First observe that a type is inhabited if and only if there exists a closed Church-style term of that type. Thus it suffices to consider Church-style terms. First we prove that our problem is in PSPACE.

6.6.1. LEMMA. *There is an alternating polynomial time algorithm (and thus also a deterministic polynomial space algorithm) to determine whether a given type τ is inhabited in a given context Γ.*

PROOF. If a type is inhabited then, by Lemma 6.5.4, it is inhabited by a term in a long normal form. To determine whether there exists a term M in a long normal form, 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 answer in parallel the questions if τ_i are inhabited in Γ.

This alternating (or recursive) procedure is repeated as long as there are new questions of the form $\Gamma \vdash ? : \tau$. Note 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 $M[x/y]$ 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 context Γ either stays the same or it expands. Thus the number of steps (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. □

To show PSPACE-hardness, we define a reduction from the satisfiability problem for classical second-order propositional formulas (QBF). We refer the reader to [57] for details about this problem. (But do not use the Polish translation.)

Assume that a second-order propositional 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 a basis Γ_Φ from the following types:

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

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

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

The following lemma is proven by a routine induction w.r.t. the length of formulas. Details are left to the reader.

6.6.2. 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 $v(\varphi) = 1$.*

PROOF. Exercise 6.8.18. □

From Lemma 6.6.2 we obtain PSPACE-hardness, since the reduction can be performed in logarithmic space. This, together with Lemma 6.6.1 implies the main result of this section.

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

6.7. Equality of typed terms

As we have already observed, to verify whether two typable lambda-terms are beta-equal or not, it suffices to reduce them to normal form. One thing that often is overlooked here is that the complexity of this decision procedure depends on the size of particular type derivations (or Church-style terms) and *not* directly on the size of the pure lambda-terms to be verified. In the case of simply-typed lambda calculus, the cost of type-reconstruction is is a minor fraction of the total cost, even if we insist on the possibly exponential types to be written down. Indeed, we have the following theorem of Statman:

6.7.1. THEOREM (R. Statman). *The problem to decide whether two given Church-style terms M and N of a given type τ are beta-equal is of non-elementary complexity. That is, for each r, the decision procedure takes more than $\exp_r(n)$ steps on inputs of size n. (Recall that $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{exp_k(n)}$.)*

The simplest known proof of this result is by H. Mairson [68]. This proof, like the original proof of Statman, uses validity for a simple higher-order logic as an intermediate step. It might be interesting to have a direct coding of Turing Machines into lambda-terms.

6.8. Exercises

6.8.1. EXERCISE. Show that:

a) problem (4) reduces to problem (5);

b) problem (5) reduces to problem (1);

c) problem (1) reduces to problem (2);

d) problem (8) reduces to problem (7)

in logarithmic space.

6.8.2. EXERCISE. Assume a context Γ consisting of type assumptions of the form $(x_\alpha : \alpha)$. Define terms t_τ such that $\Gamma \vdash t_\tau : \sigma$ holds if and only if $\tau = \sigma$.

6.8.3. EXERCISE. Show that problem (6) reduces to problem (4) in logarithmic space. *Hint:* first do Exercise 6.8.2.

6.8.4. EXERCISE. What's wrong with the following reduction of problem (3) to problem (4)? To answer $? \vdash M : \tau$ ask $? \vdash \lambda yz.y(zM)(zt_\tau) : ?$.

6.8.5. Exercise. Show that for every system of equations there is an equivalent single equation.

6.8.6. Exercise. Show that the size of a shortest possible solution of a given system of equations may be exponential in the size of that system. (Construct systems of equations of size $\mathcal{O}(n)$, such that all their solutions are of size at least exponential in n.) Can a solution be always *represented* in polynomial space?

6.8.7. Exercise. Prove that the general form of the unification problem reduces in logarithmic space to unification over the signature consisting only of an arrow.

6.8.8. Exercise. Complete the proof of Lemma 6.3.4.

6.8.9. Exercise. Show examples of loops in the unification algorithm that may be caused by using rule (d) in a silly way, or removing the restriction in rule (a) that t is not a variable.

6.8.10. Exercise. Design a polynomial time algorithm to decide whether a given equation has a solution.

6.8.11. Exercise. Modify the algorithm of Section 6.4 to obtain an algorithm for problem (3).

6.8.12. Exercise. Prove the following *converse principal type theorem*: If φ is a non-empty type, then there exists a closed term M such that φ is the principal type of M. *Hint:* Use the technique of Exercise 6.8.2. (In fact, if N is a closed term of type τ, then we can require M to be beta-reducible to N.)

6.8.13. Exercise. Show that every principal pair of a BCK-term in normal form has the following properties:

- Each type variable occurs at most twice;
- If it does, then one occurrence is positive (to the left of an even number of arrows) and the other negative.

Also show that if (Γ, τ) is a principal pair of a BCK-term M in normal form then M is an erasure of a Church style term in long normal form that has the type τ in Γ.

6.8.14. Exercise. (S. Hirokawa [56])
Prove that for a given pair (Γ, τ), there is at most one BCK-term M in normal form such that (Γ, τ) is its principal pair.

6.8.15. Exercise. Prove Proposition 6.5.2.

6.8.16. Exercise. What is the natural deduction counterpart of eta-reduction?

6.8.17. Exercise. Show examples of types that have exactly n normal inhabitants, for any number $n \in \mathbb{N} \cup \{\aleph_0\}$.

6.8.18. Exercise. Prove Lemma 6.6.2

6.8.19. Exercise. (C.-B. Ben-Yelles)
Show that it is decidable whether a type has a finite or an infinite number of different normal inhabitants.

6.8.20. Exercise. Let $\varphi = \tau_1 \to \cdots \to \tau_n \to \alpha$ and let α be the only type variable occurring in φ. Prove (by induction) that φ is an inhabited type if and only if at least one of the τ_i's is *not* inhabited.

6.8.21. Exercise. (R. 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. *Hint:* First do Exercise 6.8.20.

6.8.22. Exercise. (T. Prucnal, W. Dekkers [30]) 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.

CHAPTER 7

Sequent calculus

We have seen two different formalisms for presenting systems of formal logic: natural deduction and Hilbert-style. Each of these has its advantages. For instance, in Hilbert-style proofs, there are no problems pertaining to the management of assumptions, whereas in natural deduction, the proofs are easier to discover, informally speaking.

As we have seen earlier, both classical and intuitionistic propositional calculus are decidable; that is, there is an algorithm which decides, for any φ, whether or not φ is classically valid (according to the truth table semantics), and similarly, there is an algorithm which decides, for any φ, whether or not φ is intuitionistically valid (according to the Heyting algebra semantics or Kripke models). By the soundness and completeness results, this result means that there are algorithms that decide, for any φ, whether or not φ is provable in our proof systems for classical and intuitionistic propositional calculus, respectively.

This result suggests that we should be able to develop decision algorithms that do not make an excursion via semantics; that is, we should be able to read the inference rules bottom-up and turn this reading into algorithms that decide whether formulas have proofs in the systems.

Anyone who has tried at least once to write down an actual Hilbert-style proof for even a simple formula will understand why this approach is not satisfactory in practice. If we want to prove a formula ψ, using the modus ponens rule

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

the formula φ has to be somehow chosen or guessed. And there is no bound for the space we make this choice from: the formula φ can be *anything* at all. Any approach to automatic theorem proving based on this rule seems doomed to failure.

The same problem appears in natural deduction proofs, since we also have the modus ponens rule there. (In addition, we have another unpleasant property in the Hilbert-style system: formulas occurring in proofs are very long, so even if we know what to choose, such proofs are still inconvenient.)

In this chapter we introduce a third kind of formalism, known as *sequent calculus,* for presenting systems of formal logic. Sequent calculus was introduced in the 1930's, by Gerhard Gentzen [39], who also introduced natural deduction.[1]

Despite similar syntax, sequent calculus and natural deduction are quite different and serve different purposes. While natural deduction highlights the most fundamental properties of connectives by its introduction and elimination rule for each connective, sequent calculus is more "practically" oriented: if one reads the rules of sequent calculus from bottom to top, the rules simplify the process of proof construction. Instead of introduction and elimination rules, there are only introduction rules. Some of these rules introduce connectives in the conclusion parts of judgements—in fact, these rules are identical to the introduction rules from natural deduction. But there are also rules introducing connectives in the assumption parts of judgements. These rules replace the elimination rules of natural deduction.

The development of sequent calculus systems has been successful not only for theoretical purposes: many practical approaches to automated theorem proving are based on some form of sequent calculi or their relatives. In particular, the *resolution rule* can be seen as such a relative.

7.1. Classical sequent calculus

Although we are now mostly concerned with intuitionistic proof systems, we introduce a classical version of sequent calculus first. Intuitionistic sequent calculus is obtained from the classical one by a restriction which sheds some light on the relationship between these two equally fundamental logics.

As mentioned below, many variations on the definition of sequent calculus systems are possible; we use the systems studied by Prawitz [85, App. A], since these minimize the noise in the relationship with natural deduction. A number of variants can be found in, e.g., [109] and [38].

7.1.1. DEFINITION. A (classical) *sequent* is a pair of sets of formulas, written $\Gamma \vdash \Sigma$, where the right-hand side is not empty.[2] A *proof* in the sequent calculus is a tree labelled with sequents in such a way that mothers and daughters match the proof rules below.

We write $\Gamma \vdash^{+}_{LC} \Sigma$ iff $\Gamma \vdash \Sigma$ has a proof, and we write $\Gamma \vdash_{LC} \Sigma$ iff $\Gamma \vdash \Sigma$ has a proof which does not use the rule *Cut*.

[1]Stanisław Jaśkowski independently introduced natural deduction systems—see Prawitz' book [85], where more information about the origins of natural deduction and sequent calculus systems can be found.

[2]Sequents with empty right-hand sides are permitted in many presentations. The meaning of an empty right-hand side is the same as of a right-hand side consisting only of $\bot$, so our restriction is not essential.

We use similar conventions as in the case of natural deduction. For instance, we write Γ, Δ for $\Gamma \cup \Delta$, and φ for $\{\varphi\}$.

Axiom:

$$\Gamma, \varphi \vdash \varphi, \Delta$$

Rules:

$$\frac{\Gamma, \varphi \vdash \Sigma}{\Gamma, \varphi \wedge \psi \vdash \Sigma}(\mathrm{L}\wedge)\ \frac{\Gamma, \psi \vdash \Sigma}{\Gamma, \varphi \wedge \psi \vdash \Sigma} \qquad \frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}(\mathrm{R}\wedge)$$

$$\frac{\Gamma, \varphi \vdash \Sigma \quad \Gamma, \psi \vdash \Sigma}{\Gamma, \varphi \vee \psi \vdash \Sigma}(\mathrm{L}\vee) \qquad \frac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}(\mathrm{R}\vee)\frac{\Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma, \psi \vdash \Sigma}{\Gamma, \varphi \to \psi \vdash \Delta \cup \Sigma}(\mathrm{L}\to) \qquad \frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \to \psi, \Delta}(\mathrm{R}\to)$$

$$\frac{}{\Gamma, \bot \vdash \Sigma}(\mathrm{L}\bot)$$

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma, \varphi \vdash \Sigma}{\Gamma \vdash \Delta \cup \Sigma}(\mathit{Cut})$$

The rule (L$\bot$) has no premise and may be regarded as an axiom. The remaining rules, except the cut rule, are called *logical rules* since they define the meaning of the logical connectives $\wedge$, $\vee$, and $\to$. The logical rules consists of left and right introduction rules for every connective. The right rules are identical to the introduction rules from natural deduction; the left rules will play the role of the elimination rules from natural deduction.

The cut rule is the only one that is neither a left nor a right rule. The formula φ in the cut rule is called the *cut* formula. One can recognize a similarity between cut and *modus ponens*.

The intuitive meaning of $\Gamma \vdash \Sigma$ is that the assumptions in Γ imply *one of* the conclusions in Σ, i.e., that $\varphi_1 \wedge \ldots \wedge \varphi_n$ implies $\psi_1 \vee \ldots \vee \psi_m$, where $\Gamma = \{\varphi_1, \ldots, \varphi_n\}$ and $\Sigma = \{\psi_1, \ldots, \psi_m\}$. The rules for conjunction and disjunction clearly reflect this idea.

7.1.2. Remark. In order to facilitate comparison with natural deduction we have taken $\bot$ as primitive—as we have seen earlier, negation can then be defined by $\neg\varphi = \varphi \to \bot$. One often finds in the literature $\neg$ taken as primitive instead. In this case, the rule (L$\bot$) is replaced by the two rules

$$\frac{\Gamma \vdash \Delta, \varphi}{\Gamma, \neg\varphi \vdash \Delta}(\mathrm{L}\neg) \qquad\qquad \frac{\Gamma, \varphi \vdash \Sigma}{\Gamma \vdash \neg\varphi, \Sigma}(\mathrm{R}\neg)$$

which are derived rules in the above system.

7.1.3. Warning. In many presentations of sequent calculus, sequents are pairs of *sequences* (with possible repetitions) rather than sets. In such systems one must in addition to the axiom, the cut rule, and the logical rules, adopt so-called *structural* rules, namely *weakening* rules that allow addition of formulas to the left and right of $\vdash$, *contraction* rules that allow contraction of two identical formulas into one on the left and right of $\vdash$, and *exchange* rules that allow changing the order of two consecutive formulas on the left or on the right of $\vdash$. In this case one takes the axiom in the form $\varphi \vdash \varphi$. Such a system occurs, e.g., in [46].

One may also use *multi-sets* instead of sets and sequences. In this case, the exchange rules are not needed.

7.1.4. Remark. It is not difficult to see that in the presence of weakening, the rules (*Cut*) and (L$\to$) could as well be written as follows:

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma, \varphi \vdash \Delta}{\Gamma \vdash \Delta}(\mathit{Cut}) \qquad\qquad \frac{\Gamma \vdash \varphi, \Sigma \quad \Gamma, \psi \vdash \Sigma}{\Gamma, \varphi \to \psi \vdash \Sigma}(\mathrm{L} \to)$$

We prefer the other presentation of these rules for uniformity with the intuitionistic fragment, to be defined in the next section.

The following shows that sequent calculus is complete with respect to the ordinary semantics of classical logic.

7.1.5. Proposition. *If* $\Sigma = \{\varphi_1, \ldots \varphi_n\}$ *then we have* $\Gamma \vdash^+_{LC} \Sigma$ *if and only if the entailment* $\Gamma \models \varphi_1 \vee \cdots \vee \varphi_n$ *is classically valid.*[3] *In particular,* $\vdash^+_{LC} \Sigma$ *iff* $\varphi_1 \vee \cdots \vee \varphi_n$ *is a classical tautology.*

The proof is omitted. Gentzen [39] proved the completeness of the sequent calculus system by proving that the system is equivalent to another logical system. For the purposes of that proof, the cut rule is very convenient. Gentzen's *Hauptsatz* then states that the cut rule is a derived rule, and hence the cut rule is in fact not necessary for completeness; that is, every application of the cut rule can be eliminated from a given proof. This results is also known as the *Cut Elimination Theorem.* We shall have more to say about this result in the context of intuitionistic logic below.

[3]That is, iff each valuation satisfying all the formulas in Γ must also satisfy $\varphi_1 \vee \cdots \vee \varphi_n$.

7.1.6. EXAMPLE. Here is a sequent calculus proof of Peirce's law:

$$(\mathrm{R}\to)\dfrac{(\mathrm{L}\to)\dfrac{(\mathrm{R}\to)\dfrac{p \vdash p, q}{\vdash p, p \to q} \qquad p \vdash p}{(p \to q) \to p \vdash p}}{\vdash ((p \to q) \to p) \to p}$$

Note that we sometimes have two formulas at the right-hand side.

7.1.7. EXAMPLE. And here is another example that uses only sequents with one-element right-hand sides:

$$(\mathrm{R}\to)\dfrac{(\mathrm{R}\to)\dfrac{(\mathrm{R}\to)\dfrac{(\mathrm{L}\to)\dfrac{p,\ p \to q \vdash p \qquad \dfrac{(\mathrm{L}\to)\dfrac{p \vdash p \qquad p,\ q \vdash q}{p,\ p \to q \vdash q} \qquad p,\ p \to q,\ r \vdash r}{p,\ p \to q,\ q \to r \vdash r}(\mathrm{L}\to)}{p,\ p \to q,\ p \to q \to r \vdash r}}{p \to q,\ p \to q \to r \vdash p \to r}}{p \to q \to r \vdash (p \to q) \to p \to r}}{\vdash (p \to q \to r) \to (p \to q) \to p \to r}$$

7.2. Intuitionistic sequent calculus

The intuitionistic sequent calculus is obtained from the classical system by a simple syntactic restriction. We just require that only one formula occurs at the right-hand side of a sequent. That is, the above classical rules are modified so that

- Σ has always exactly one element;
- Δ is always empty.

7.2.1. DEFINITION. An *intuitionistic sequent* is one of the form $\Gamma \vdash \varphi$, where φ is a single formula. We write $\Gamma \vdash_L^+ \varphi$ iff $\Gamma \vdash \varphi$ has a sequent calculus proof using only intuitionistic sequents, i.e., using only the below rules. We write $\Gamma \vdash_L \varphi$ if there is such a proof that does not use the rule *Cut*.

Axiom:

$$\Gamma, \varphi \vdash \varphi$$

Rules:

$$\frac{\Gamma, \varphi \vdash \sigma}{\Gamma, \varphi \wedge \psi \vdash \sigma}(\mathrm{L}\wedge) \quad \frac{\Gamma, \psi \vdash \sigma}{\Gamma, \varphi \wedge \psi \vdash \sigma} \qquad \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}(\mathrm{R}\wedge)$$

$$\frac{\Gamma, \varphi \vdash \sigma \quad \Gamma, \psi \vdash \sigma}{\Gamma, \varphi \vee \psi \vdash \sigma}(\mathrm{L}\vee) \qquad \frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}(\mathrm{R}\vee)\frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}$$

$$\frac{\Gamma \vdash \varphi \quad \Gamma, \psi \vdash \sigma}{\Gamma, \varphi \to \psi \vdash \sigma}(\mathrm{L}{\to}) \qquad \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}(\mathrm{R}{\to})$$

$$\frac{}{\Gamma, \bot \vdash \sigma}(\mathrm{L}\bot)$$

$$\frac{\Gamma \vdash \varphi \quad \Gamma, \varphi \vdash \sigma}{\Gamma \vdash \sigma}(\mathit{Cut})$$

The following shows that intuitionistic natural deduction and intuitionistic sequent calculus are equivalent.

7.2.2. Proposition. *$\Gamma \vdash_L^+ \varphi$ iff $\Gamma \vdash_N \varphi$.*

Proof. We prove each direction by induction on the derivation of the sequent.

For the *left-to-right* direction, the main problem is how to express the left rules of sequent calculus in terms of the elimination rules of natural deduction, and how to express the cut rule in terms of modus ponens.

1. The derivation of $\Gamma \vdash_L^+ \varphi$ is

$$\Gamma', \varphi \vdash \varphi$$

Then

$$\Gamma', \varphi \vdash \varphi$$

is also a derivation of $\Gamma \vdash_N \varphi$.

2. The derivation of $\Gamma \vdash_L^+ \varphi$ ends in

$$\frac{\Gamma', \psi_1 \vdash \varphi}{\Gamma', \psi_1 \wedge \psi_2 \vdash \varphi}$$

By the induction hypothesis we have a natural deduction derivation of $\Gamma', \psi_1 \vdash \varphi$. By Lemma 2.6, $\Gamma', \psi_1 \wedge \psi_2, \psi_1 \vdash \varphi$, so $\Gamma', \psi_1 \wedge \psi_2 \vdash \psi_1 \to \varphi$. Since also $\Gamma', \psi_1 \wedge \psi_2 \vdash \psi_1 \wedge \psi_2$, and hence $\Gamma', \psi_1 \wedge \psi_2 \vdash \psi_1$, we get $\Gamma', \psi_1 \wedge \psi_2 \vdash \varphi$, by modus ponens. Thus $\Gamma \vdash_N \varphi$.

3. The derivation of $\Gamma \vdash_L^+ \varphi$ ends in

$$\frac{\Gamma', \psi_1 \vdash \varphi \qquad \Gamma', \psi_2 \vdash \varphi}{\Gamma', \psi_1 \vee \psi_2 \vdash \varphi}$$

By the induction hypothesis we have derivations in natural deduction of $\Gamma', \psi_1 \vdash \varphi$ and $\Gamma', \psi_2 \vdash \varphi$. By Lemma 2.6, $\Gamma', \psi_1, \psi_1 \vee \psi_2 \vdash \varphi$ and $\Gamma', \psi_2, \psi_1 \vee \psi_2 \vdash \varphi$. Since also $\Gamma', \psi_1 \vee \psi_2 \vdash \psi_1 \vee \psi_2$, we get $\Gamma', \psi_1 \vee \psi_2 \vdash \varphi$. Thus $\Gamma \vdash_N \varphi$.

4. The derivation of $\Gamma \vdash_L^+ \varphi$ ends in

$$\frac{\Gamma' \vdash \psi_1 \qquad \Gamma', \psi_2 \vdash \varphi}{\Gamma', \psi_1 \to \psi_2 \vdash \varphi}$$

By the induction hypothesis we have derivations in natural deduction $\Gamma' \vdash \psi_1$ and $\Gamma', \psi_2 \vdash \varphi$. By Lemma 2.6, $\Gamma', \psi_1 \to \psi_2 \vdash \psi_1$ and $\Gamma', \psi_1 \to \psi_2, \psi_2 \vdash \varphi$. As before, $\Gamma', \psi_1 \to \psi_2 \vdash \psi_1 \to \psi_2$, so $\Gamma', \psi_1 \to \psi_2 \vdash \psi_2$. Also $\Gamma', \psi_1 \to \psi_2 \vdash \psi_2 \to \varphi$, so $\Gamma', \psi_1 \to \psi_2 \vdash \varphi$. Thus $\Gamma \vdash_N \varphi$.

5. The derivation of $\Gamma \vdash_L^+ \varphi$ is

$$\frac{}{\Gamma', \bot \vdash \varphi}$$

Then

$$\frac{\dfrac{}{\Gamma', \bot \vdash \bot}}{\Gamma', \bot \vdash \varphi}$$

is a derivation of $\Gamma \vdash_N \varphi$.

6. The derivation of $\Gamma \vdash_L^+ \varphi$ ends in

$$\frac{\Gamma \vdash \psi \qquad \Gamma, \psi \vdash \varphi}{\Gamma \vdash \varphi}$$

By the induction hypothesis we have derivations in natural deduction of $\Gamma \vdash \psi$ and $\Gamma, \psi \vdash \varphi$. Then $\Gamma \vdash \psi \to \varphi$, and then $\Gamma \vdash \varphi$. Thus $\Gamma \vdash_N \varphi$.

The remaining cases—the right rules—are trivial.

For the *right-to-left* direction the problem is to express the elimination rules of natural deduction in terms of the left rules of sequent calculus; the cut rule turns out to be useful for this.

As above the cases where the derivation consists of a use of the axiom or ends in an introduction rule are trivial.

1. The derivation of $\Gamma \vdash_N \varphi$ ends in

$$\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}$$

   By the induction hypothesis we have a sequent calculus derivation of $\Gamma \vdash \varphi \wedge \psi$. By the axiom and the left rule for $\wedge$ we get $\Gamma, \varphi \wedge \psi \vdash \varphi$. Then by the cut rule $\Gamma \vdash \varphi$. Thus $\Gamma \vdash_L^+ \varphi$.

2. The derivation of $\Gamma \vdash_N \varphi$ ends in

$$\frac{\Gamma, \psi_1 \vdash \varphi \qquad \Gamma, \psi_2 \vdash \varphi \qquad \Gamma \vdash \psi_1 \vee \psi_2}{\Gamma \vdash \varphi}$$

   By the induction hypothesis we have sequent calculus derivations of $\Gamma, \psi_1 \vdash \varphi$, of $\Gamma, \psi_2 \vdash \varphi$, and of $\Gamma \vdash \psi_1 \vee \psi_2$. By the left rule for $\vee$ we get $\Gamma, \psi_1 \vee \psi_2 \vdash \varphi$. Then by the cut rule $\Gamma \vdash \varphi$. Thus $\Gamma \vdash_L^+ \varphi$.

3. The derivation of $\Gamma \vdash_N \varphi$ ends in

$$\frac{\Gamma \vdash \psi \to \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi}$$

   By the induction hypothesis we have sequent calculus derivations of $\Gamma \vdash \psi \to \varphi$ and $\Gamma \vdash \psi$. By the axiom $\Gamma, \varphi \vdash \varphi$, so by the left rule for $\to$ we have that $\Gamma, \psi \to \varphi \vdash \varphi$. Then by the cut rule $\Gamma \vdash \varphi$. Thus $\Gamma \vdash_L^+ \varphi$.

4. The derivation of $\Gamma \vdash_N \varphi$ ends in

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}$$

   By the induction hypothesis we have a sequent calculus derivation of $\Gamma \vdash \bot$. By the left rule for $\bot$ we have $\Gamma, \bot \vdash \varphi$. Then by the cut rule $\Gamma \vdash \varphi$. Thus $\Gamma \vdash_L^+ \varphi$. □

7.3. Cut elimination

In both directions of the proof of Proposition 7.2.2 we introduce detours. In the left-to-right direction we express the left rule for, say $\wedge$, by a sequence of rules in which a $\wedge$-introduction is immediately followed by a $\wedge$-elimination (this is reflected by a redex of form $\pi_i(< M_1, M_2 >)$ in the λ-term corresponding to the proof). In general we expressed each left rule of the sequent calculus system by a natural deduction proof in which a sequent occurrence was both the conclusion of an introduction rule and the major[4] premise of the corresponding elimination rule (in general, such sequent occurrences are reflected by redexes in the λ-term corresponding to the proof).

In the right-to-left direction we used the cut rule to express elimination rules in terms of left rules.

We know that we can get rid of the detours in the natural deduction proofs; that is, we can transform any natural deduction proof into one in which no sequent occurrence is both the conclusion of an introduction rule and the major premise of the corresponding elimination rule. This corresponds to the fact that, by the weak normalization theorem, we can eliminate all redexes in a term of the simply typed λ-calculus with pairs and sums.

The following theorem states that we can also do without the cuts.

7.3.1. THEOREM (Cut elimination). *For all φ and Γ the conditions $\Gamma \vdash_L^+ \varphi$ and $\Gamma \vdash_L \varphi$ are equivalent.*

The proof is somewhat tedious, especially when presented in terms of proof trees—see, e.g., [46] or [109]. Therefore we postpone the proof to Section 7.6 where a more convenient notation for proofs is developed. Here we merely reflect on some of the more interesting aspects of the proof, and consider some consequences of the theorem.

First observe that there is no uniform way to eliminate an application of the cut rule, i.e., there is no fixed sequence of other rules that is equivalent to a cut. Each cut has to be eliminated differently, and this depends on the shape of the *cut formula* and the way it was constructed above the cut.

In addition, in an attempt to eliminate a cut with a complex cut formula, i.e., one of the form $\varphi \to \psi$, we may actually create new cuts, as can be seen from the following example. Consider a proof that ends with an application of a cut rule of the form:

[4]In $\vee E$, $\to E$, and $\wedge E$ the major premise is the leftmost one, the rightmost one, and the single one, respectively.

$$\dfrac{(\mathrm{R}\to)\dfrac{\begin{matrix}(1)\\ \vdots\\ \Gamma,\varphi\vdash\psi\end{matrix}}{\Gamma\vdash\varphi\to\psi} \qquad \dfrac{\begin{matrix}(2)\\ \vdots\\ \Gamma\vdash\varphi\end{matrix}\qquad \begin{matrix}(3)\\ \vdots\\ \Gamma,\psi\vdash\vartheta\end{matrix}}{\Gamma,\ \varphi\to\psi\vdash\vartheta}(\mathrm{L}\to)}{\Gamma\vdash\vartheta}(Cut)$$

We can eliminate this cut at the cost of introducing two new ones. This makes sense, because the new cut formulas are simpler. The new proof is as follows:

$$\dfrac{(Cut)\dfrac{\begin{matrix}(2)\\ \vdots\\ \Gamma\vdash\varphi\end{matrix}\qquad \begin{matrix}(1)\\ \vdots\\ \Gamma,\varphi\vdash\psi\end{matrix}}{\Gamma\vdash\psi} \qquad \begin{matrix}(3)\\ \vdots\\ \vdots\\ \Gamma,\psi\vdash\vartheta\end{matrix}}{\Gamma\vdash\vartheta}(Cut)$$

Note that in our example the cut formula $\varphi\to\psi$ was introduced just before the cut by the rules (R$\to$) and (L$\to$).

The strategy of the cut elimination proof is as follows. The main cases, when the cut formula is introduced by the appropriate left and right rules directly before cut, is treated as in our example: by replacing the cut by new "simpler" cuts. Other cuts are "permuted upward" so that each cut is eventually either applied to an axiom (an easy case), or another main case is obtained. This requires an induction over two parameters: the depths of cuts and the complexity of cut formulas.[5]

7.3.2. Remark. The cut elimination theorem also holds for the classical sequent calculus; that is, for all Σ and Γ the conditions $\Gamma\vdash^{+}_{LC}\Sigma$ and $\Gamma\vdash_{LC}\Sigma$ are equivalent.

7.3.3. Lemma (Subformula property). *The cut-free sequent calculus $\vdash_L$ has the following property: Each formula occurring in a proof of $\Gamma\vdash\varphi$ is either a subformula of φ or a subformula of a formula occurring in Γ.*

Proof. By induction on the derivation of $\Gamma\vdash\varphi$. □

There are a number of consequences of the subformula property. One is that finding a sequent calculus proof of a given formula (or finding out that no such proof exists) is incomparably easier than finding such a proof in the Hilbert, or natural deduction system. As we reconstruct the proof

[5]The similarity between this approach and the proof method of weak normalization is not at all incidental.

by building the tree upward, the search space at each step is limited to subformulas of the formulas occurring at the present stage. This process cannot continue indefinitely, as the number of available formulas is bounded, and we will eventually repeat already considered sequents.

7.3.4. COROLLARY (Gentzen). *It is decidable, for input φ, whether $\vdash_L^+ \varphi$.*

Another consequence is the conservativity of fragments of the calculus determined by a choice of connectives. The subformula property implies that a cut-free proof of a sequent can only mention connectives occurring in that sequent. Thus, e.g., a formula $((p \land q) \to r) \leftrightarrow (p \to (q \to r))$ is provable in a system containing only rules for implication and conjunction.

7.3.5. COROLLARY. IPC *is conservative over its implicational fragment.*

We end this section with another proof of the disjunction property (Proposition 2.5.7).

7.3.6. COROLLARY. *If $\vdash_L^+ \varphi \lor \psi$ then either $\vdash_L^+ \varphi$ or $\vdash_L^+ \psi$.*

PROOF. If there is a proof of $\vdash_L^+ \varphi \lor \psi$, then there is a cut-free one. And a cut-free proof of a disjunction must end up with an application of rule ($\lor$I). Thus, either $\vdash_L \varphi$ or $\vdash_L \psi$ must have been proved first. □

7.4. Term assignment for sequent calculus

Natural deduction proofs correspond to typed λ-terms and Hilbert-style proofs correspond to typed combinators. What do sequent calculus proofs correspond to?

There are several answers. The traditional one—see, e.g., [84, 118]—is that we can assign lambda-terms to sequent calculus proofs; that is, we can devise an alternative version of simply typed λ-calculus—with the same term language, but with different typing rules—which is to sequent calculus what the traditional formulation of simply typed λ-calculus is to natural deduction.

This is carried out below. We begin with the implicational fragment.

7.4.1. DEFINITION (Sequent calculus style $\lambda\!\to$). The type and term language of the sequent calculus style $\lambda\!\to$ is as for $\lambda\!\to$ (à la Curry). The typing rules are as follows:

Axiom:

$$\Gamma, x{:}\varphi \vdash x : \varphi$$

Rules:

$$\frac{\Gamma \vdash M : \varphi \quad \Gamma, x{:}\psi \vdash N : \sigma}{\Gamma, y{:}\varphi \to \psi \vdash N[x := yM] : \sigma}(\mathrm{L}{\to}) \qquad \frac{\Gamma, x{:}\varphi \vdash M : \psi}{\Gamma \vdash \lambda x.M : \varphi \to \psi}(\mathrm{R}{\to})$$

$$\frac{\Gamma \vdash M : \varphi \quad \Gamma, x{:}\varphi \vdash N : \sigma}{\Gamma \vdash (\lambda x.N)M : \sigma}(\mathit{Cut})$$

We also write $\vdash_L^+$ and $\vdash_L$ for derivability in this system with and without cut, respectively. We thus have binary and ternary version of both $\vdash_L^+$ and $\vdash_L$; the binary version refers to the sequent calculus formulation of IPC($\to$) in Definition 7.2.1, and the ternary version refers to the present sequent calculus style formulation of $\lambda{\to}$.

As usual we have that the system with terms agrees with the system without terms.

7.4.2. Proposition.

(i) *If* $\Gamma \vdash_L^+ M : \varphi$ *then* $|\Gamma| \vdash_L^+ \varphi$.

(ii) *If* $\Gamma \vdash_L^+ \varphi$ *then there exists* $M \in \Lambda$ *such that* $\Delta \vdash_L^+ M : \varphi$, *where* $\Delta = \{(x_\varphi : \varphi) \mid \varphi \in \Gamma\}$.

The above sequent calculus system assigns types to certain λ-terms. Are these the same λ-terms as those that receive types by the usual simply typed λ-calculus à la Curry? The answer is *no!* For instance, there is no way to assign a type to $(\lambda x.\lambda y.x)\ (\lambda z.z)\ (\lambda z.z)$ in the above system.

However, the proof of Proposition 7.2.2 implicitly defines a translation from terms typable in simply typed λ-calculus (corresponding to natural deduction proofs) to terms typable in the above system (corresponding to sequent calculus proofs), and vice versa.

On the other hand, if we restrict attention to λ-terms in normal form, then the set of terms typable in traditional simply typed λ-calculus coincides with the set of terms typable in sequent calculus.

7.4.3. Proposition. *For every term* M *in normal form,* $\Gamma \vdash_L^+ M : \varphi$ *iff* $\Gamma \vdash M : \varphi$ *in simply typed* λ*-calculus.*

Proof. First show that a term M is in normal form iff either

- M is a variable, or
- $M = \lambda x.N$, where N is a normal form, or
- $M = N[y := xP]$, where N and P are normal forms.

Then the property follows easily. □

In the correspondence between simply typed λ-calculus and natural deduction, λ-terms in normal form correspond to normal deductions (i.e., deductions where no sequent is at the same time the conclusion of an introduction rule and the major premise of the corresponding elimination rule).

In the sequent calculus variant of simply typed λ-calculus, λ-terms in normal form correspond to cut-free proofs in sequent calculus.

7.4.4. Proposition. *If* $\Gamma \vdash_L^+ M : \varphi$*, then* M *is in normal form if and only if* $\Gamma \vdash_L M : \varphi$*.*

Proof. Obvious. □

Thus simply typable λ-terms in normal form correspond to both normal proofs in natural deduction and cut-free sequent calculus proofs. We therefore have the correspondence

$$\textit{Normal deductions} \quad \Longleftrightarrow \quad \textit{Cut-free proofs}$$

However, note that a deduction may use the cut rule even if the corresponding λ-term is in normal form (cf. Exercise 7.7.5): the substitution $N[x := y\,M]$ may delete the term M which may contain redexes. In this case we just know that there is another typing that does not use the cut rule of the same term.

7.4.5. Remark. As mentioned above, the proof of Proposition 7.2.2 implicitly defines a translation from terms typable in traditional simply typed λ-calculus (corresponding to natural deduction proofs) to terms typable in the above system (corresponding to sequent calculus proofs), and vice versa.

From what has been said above one might expect that the translations map normal forms to normal forms. However this is not the case. The reason for this is that in the proof of Proposition 7.2.2 we aimed at the simplest possible way to get from natural deduction proofs to sequent calculus proofs; in particular, we translated the left rules of sequent calculus into natural deduction proofs containing detours, and we made use of the cut rule in translating elimination rules into left rules.

7.5. The general case

We briefly show how the development of the preceding section can be generalized to the full propositional language.

Recall the extension of $\lambda\!\to$ à la Curry with pairs and sums:

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

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

$$\frac{\Gamma \vdash L : \psi \vee \varphi \quad \Gamma, x : \psi \vdash M : \rho \quad \Gamma, y : \varphi \vdash N : \rho}{\Gamma \vdash \mathrm{case}(L; x.M; y.N) : \rho}$$

For completeness, extend the language with an operator ε for falsity, with the following rule:

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \varepsilon(M) : \sigma}$$

and with no reduction rule (as there is no $\bot$-introduction rule).

First we generalize the construction in the proof of Proposition 7.4.3

7.5.1. LEMMA. *A term M is in normal form iff either*

- *M is a variable, or*
- *$M = \lambda x.P$, or*
- *$M = P[y := xQ]$, or*
- *$M =< P, Q >$, or*
- *$M = \mathrm{in}_1(P)$, or*
- *$M = \mathrm{in}_2(P)$, or*
- *$M = P[y := \pi_1(x)]$, or*
- *$M = P[y := \pi_2(x)]$, or*
- *$M = P[y := \mathrm{case}(x; v.Q; w.R)]$, or*
- *$M = \varepsilon(P)$,*

where P, Q, and R are normal forms.

PROOF. Easy. □

7.5.2. Definition (Sequent calculus style $\lambda\!\!\rightarrow$ for the full language). The sequent calculus style $\lambda\!\!\rightarrow$ for the full propositional language is as for $\lambda\!\!\rightarrow$ à la Curry with pairs and sums. The typing rules are are those of Definition 7.4.1 and in addition the following:

$$(\mathrm{L}\wedge)\ \frac{\Gamma, x{:}\varphi_i \vdash M : \sigma}{\Gamma, y{:}\varphi_1 \wedge \varphi_2 \vdash M[x := \pi_i(y)] : \sigma} \qquad \frac{\Gamma \vdash M : \varphi \quad \Gamma \vdash N : \psi}{\Gamma \vdash < M, N >: \varphi \wedge \psi}(\mathrm{R}\wedge)$$

$$(\mathrm{L}\vee)\frac{\Gamma, x{:}\varphi \vdash M : \sigma \quad \Gamma, y{:}\psi \vdash N : \sigma}{\Gamma, z{:}\varphi \vee \psi \vdash \mathrm{case}(z; x.M; y.N) : \sigma} \qquad \frac{\Gamma \vdash M : \varphi_i}{\Gamma \vdash \mathrm{in}_i(M) : \varphi_1 \vee \varphi_2}(\mathrm{R}\vee)$$

$$\frac{}{\Gamma, x{:}\bot \vdash \varepsilon(x) : \sigma}(\mathrm{L}\bot)$$

It is a routine matter to verify that the generalized version of Proposition 7.4.2 holds.

We would like now to generalize Proposition 7.4.3, but there is a problem. Some typable lambda-terms in normal form do not correspond to any terms typable in the new system. For instance, if M, N, P and Q are normal forms, then the term $\pi_1(\mathrm{case}(z; x. < M, N >; y. < P, Q >))$ is in normal form. But it has no type in the above system, even if the term is typable in $\lambda\!\!\rightarrow$ with pairs and sums (to see this, observe that no rule could possibly be the last one used.)

One way to remedy this problem is to modify the term assignment for the (Cut) rule to:

$$\frac{\Gamma \vdash M : \varphi \quad \Gamma, x{:}\varphi \vdash N : \sigma}{\Gamma \vdash N[x := M] : \sigma}(\mathit{Cut})$$

Then our example term can be typed, but only using the cut-rule, so the correspondence between normal proofs and cut-free proofs has been lost.

Incidentally, this difficulty does not occur for implication and conjunction, but only for the disjunction and falsity. The reason is that the elimination rules for these connectives are different. Recall that every elimination rule has a *main* premise involving the eliminated connective. In case of implication and conjunction, the conclusion of the elimination rule (more precisely: the right-hand side of the conclusion) is a subformula of the main premise. In case of disjunction and falsity this is not the case.

Our example term corresponds to the following sequence of proof steps: conjunction introduction (pairing) followed by disjunction elimination (case), followed by conjunction elimination (projection). Due to the "irregular" behaviour of disjunction elimination, the last projection should actually be applied to the pair(s) created at the beginning. But the case instruction makes this impossible. It is a stranger who entered here by mistake due to an improperly closed door (the "bad" elimination, as Girard calls it) and does her own work quite unrelated to the form of the main premise. A solution is either to ignore her or to open the door even more and let her go out. Technically, these two alternatives mean that we should either relax the existing reduction rules to allow for reduction of introduction/elimination pairs, even if the latter does not immediately follow the former, or we should introduce *commuting conversions*, i.e., reduction rules to permute eliminations.

After Girard [46] we take the second option.

7.5.3. Definition. In $\lambda\!\rightarrow$ with pairs and sums let $\to_c$ denote the union of $\to_\beta$, of the β-reductions for pairs and sums (see Page 75), and of the compatible closure of the relation defined by the following rules:

- $\pi_1(\text{case}(M; x.P; y.Q)) \to \text{case}(M; x.\pi_1(P); y.\pi_1(Q))$;
- $\pi_2(\text{case}(M; x.P; y.Q)) \to \text{case}(M; x.\pi_2(P); y.\pi_2(Q))$;
- $(\text{case}(M; x.P; y.Q))N \to \text{case}(M; x.PN; y.QN)$;
- $\varepsilon(\text{case}(M; x.P; y.Q)) \to \text{case}(M; x.\varepsilon(P); y.\varepsilon(Q))$;
- $\text{case}(\text{case}(M; x.P; y.Q); z.N; v.R) \to$
  $\text{case}(M; x.\text{case}(P; z.N; v.R); y.\text{case}(Q; z.N; v.R))$;
- $\varepsilon(\pi_1(M)) \to \varepsilon(M)$;
- $\varepsilon(\pi_2(M)) \to \varepsilon(M)$;
- $\varepsilon(\varepsilon(M)) \to \varepsilon(M)$;
- $(\varepsilon(M))N \to \varepsilon(M)$;
- $\text{case}(\varepsilon(M); x.P; y.Q) \to \varepsilon(M)$.

Also, let NF_c denote the set of normal forms with respect to $\to_c$.

Now we can state a version of Proposition 7.4.3.

7.5.4. Proposition. *For every $M \in \text{NF}_c$: $\Gamma \vdash_L^+ M : \varphi$ iff $\Gamma \vdash M : \varphi$ in $\lambda\!\rightarrow$ with pairs and sums.*

For the full system we also have

7.5.5. Proposition. *For every deduction $\Gamma \vdash_L^+ M : \varphi$, M is in c-normal form if and only if $\Gamma \vdash_L M : \varphi$.*

PROOF. Obvious. □

7.5.6. REMARK. The notion of η-reduction is often understood as follows. An elimination followed by an introduction of the same connective should be ignored. We can write the following eta-rule for $\wedge$:

$$< \pi_1(M), \pi_2(M) > \to_\eta M.$$

The above rule, although looking very convincing, hides an unpleasant surprise to be discovered in Exercise 7.7.6.

For function types, as we have already observed before, the meaning of the eta rule is the postulate of extensionality for functions, In case of $\wedge$, eta-reduction has the following meaning: Every object of a product type is actually a pair.

This leads to the idea of the following "generalized extensionality" principle: Every object should be assumed to be in a "canonical form". The canonical form for an object of type $\sigma \vee \tau$ is a variant, i.e., it is either an $\mathrm{in}_1(M)$ or an $\mathrm{in}_2(N)$. Thus suggest the following eta rule for disjunction:

$$\mathrm{case}(M; x.\mathrm{in}_1(x); y.\mathrm{in}_2(y)) \to M.$$

7.5.7. WARNING. We use logical symbols $\vee$ and $\wedge$ to denote also the corresponding types. Similar symbols are often used to denote *intersection* and *union* types, which have quite a different meaning (see e.g. [6]). Our $\wedge$ is actually a *product* rather than intersection, and our $\vee$ is a variant type rather than set-theoretic or lattice union.

7.6. Alternative term assignment

The sequent calculus systems with terms in the preceding two sections reveal interesting connections about normal natural deduction proofs and cut-free sequent calculus proofs.

However, for a fine-grained analysis of cut-elimination the term assignment is not satisfactory. The problem is that different proofs correspond to the same term so that reductions on proofs is not exactly mirrored by reductions on terms. In this section we follow another well-known approach—see, e.g., [15, 116, 38]—and introduce another, more explicit, way of assigning terms to sequent calculus proofs. We shall use the term assignment to prove the Cut Elimination Theorem.

Yet another approach to term assignment appears in [52].

7.6.1. Definition (Alternative term assignment to sequent calculus). We consider the language of propositional formulas and the following term language:

$$\begin{array}{lcl} M & ::= & x \mid \mathrm{in}_1(M) \mid \mathrm{in}_2(M) \mid \lambda x.M \mid \ < M, M' > \\ & \mid & \varepsilon(x) \\ & \mid & \mathrm{case}(x; x'.M'; x''.M'') \\ & \mid & \mathrm{let}\ x = x'\ M \ \mathrm{in}\ M' \\ & \mid & \mathrm{let}\ x = \pi_1(x')\ \mathrm{in}\ M' \\ & \mid & \mathrm{let}\ x = \pi_2(x')\ \mathrm{in}\ M' \\ & \mid & \mathrm{let}^{\varphi}\ x = M'\ \mathrm{in}\ M' \end{array}$$

Note that in the first three kinds of let-expression, the form of N in the expression "let $x = N$ in M" is restricted to certain forms.

The inference rules of the system are as follows:

Axiom:

$$\Gamma, x{:}\varphi \vdash x : \varphi$$

Rules:

$$(\mathrm{L}{\to})\ \frac{\Gamma \vdash M : \varphi \quad \Gamma, x{:}\psi \vdash N : \sigma}{\Gamma, y{:}\varphi \to \psi \vdash \mathrm{let}\ x = y\ M\ \mathrm{in}\ N : \sigma} \qquad \frac{\Gamma, x{:}\varphi \vdash M : \psi}{\Gamma \vdash \lambda x.M : \varphi \to \psi}(\mathrm{R}{\to})$$

$$(\mathrm{L}\wedge)\ \frac{\Gamma, x{:}\varphi_i \vdash M : \sigma}{\Gamma, y{:}\varphi_1 \wedge \varphi_2 \vdash \mathrm{let}\ x = \pi_i(y)\ \mathrm{in}\ M : \sigma} \qquad \frac{\Gamma \vdash M : \varphi \quad \Gamma \vdash N : \psi}{\Gamma \vdash < M, N >: \varphi \wedge \psi}(\mathrm{R}\wedge)$$

$$(\mathrm{L}\vee)\frac{\Gamma, x{:}\varphi \vdash M : \sigma \quad \Gamma, y{:}\psi \vdash N : \sigma}{\Gamma, z{:}\varphi \vee \psi \vdash \mathrm{case}(z; x.M; y.N) : \sigma} \qquad \frac{\Gamma \vdash M : \varphi_i}{\Gamma \vdash \mathrm{in}_i(M) : \varphi_1 \vee \varphi_2}(\mathrm{R}\vee)$$

$$\frac{}{\Gamma, x{:}\bot \vdash \varepsilon(x) : \sigma}(\mathrm{L}\bot)$$

$$\frac{\Gamma \vdash M : \varphi \quad \Gamma, x{:}\varphi \vdash N : \sigma}{\Gamma \vdash \mathrm{let}^{\varphi}\ x = M\ \mathrm{in}\ N : \sigma}(\mathit{Cut})$$

In λ-terms with pairs and sums there are a number of *constructors* and a number *destructors*. The constructors are $< \bullet, \bullet >$, $\mathrm{in}_i(\bullet)$, and $\lambda x.\bullet$; these build up *values*, informally speaking. The destructors are $\mathrm{case}(\bullet; x.M; x'.M')$, $\bullet\ N$, and $\pi_i(\bullet)$. These inspect and dissect values, informally speaking. In the λ-calculus with pairs and sums one can freely apply a destructor to any term. The main difference to the above term language is that now the combination of destructors and constructors are expressed via an explicit rule, namely cut.

This makes it very explicit where intermediate results, e.g. a pair of which one takes a projection, are constructed. In functional programming there are various techniques to eliminate intermediate data structures from functional programs, notably Wadler's deforestation [115]. Marlow [69] studies deforestation of a functional programming language which is similar to the term assignment for sequent calculus proofs.

The following rules remove cuts from sequent calculus proofs represented by the alternative syntax of Definition 7.6.1.

7.6.2. Definition. On the term language introduced in Definition 7.6.1, we introduce the relation $\twoheadrightarrow_a$ as the transitive, reflexive, compatible closure of the relation defined by the following rules, which are divided into three groups: *main cases, absent constructor in left term,* and *absent constructor in right term.*

The main cases are:

$$\begin{array}{ll}
\text{let}^{\varphi_1\wedge\varphi_2}\ y = < M_1, M_2 > \ \text{in let}\ x = \pi_i(y)\ \text{in}\ M & \to \\
\qquad\qquad \text{let}^{\varphi_i}\ x = M_i\ \text{in}\ M & \\
\text{let}^{\varphi_1\vee\varphi_2}\ y = \text{in}_i(M)\ \text{in case}(y; x_1.M_1; x_2.M_2) & \to \\
\qquad\qquad \text{let}^{\varphi_i}\ x_i = M\ \text{in}\ M_i & \\
\text{let}^{\varphi_1\to\varphi_2}\ y = \lambda x.M\ \text{in let}\ z = y\ N\ \text{in}\ L & \to \\
\qquad\qquad \text{let}^{\varphi_1}\ x = N\ \text{in let}^{\varphi_2}\ z = M\ \text{in}\ L &
\end{array}$$

Absent constructor from left hypothesis:

$$\begin{array}{ll}
\text{let}^{\varphi}\ x = y\ \text{in}\ N & \to N\{x := y\} \\
\text{let}^{\varphi}\ x = \text{let}\ y = \pi_i(z)\ \text{in}\ M\ \text{in}\ N & \to \text{let}\ y = \pi_i(z)\ \text{in let}^{\varphi}\ x = M\ \text{in}\ N \\
\text{let}^{\varphi}\ x = \text{case}(z; y_1.M_1; y_2.M_2)\ \text{in}\ N & \to \\
\qquad \text{case}(z; y_1.\text{let}^{\varphi}\ x = M_1\ \text{in}\ N; y_2.\text{let}^{\varphi}\ x = M_1\ \text{in}\ N) & \\
\text{let}^{\varphi}\ x = \text{let}\ y = z\ M\ \text{in}\ K\ \text{in}\ N & \to \text{let}\ y = z\ M\ \text{in let}^{\varphi}\ x = K\ \text{in}\ N \\
\text{let}^{\varphi}\ x = \text{let}^{\psi}\ y = M\ \text{in}\ K\ \text{in}\ N & \to \text{let}^{\psi}\ y = M\ \text{in let}^{\varphi}\ x = K\ \text{in}\ N \\
\text{let}^{\varphi}\ x = \varepsilon(y)\ \text{in}\ N & \to \varepsilon(y)
\end{array}$$

Absent constructor from right hypothesis:

$$\begin{array}{ll}
\text{let}^{\varphi}\ x = N\ \text{in}\ y & \to \ y\{x := N\} \\
\text{let}^{\varphi}\ x = N\ \text{in}\ < M_1, M_2 > & \to \\
\qquad < \text{let}^{\varphi}\ x = N\ \text{in}\ M_1, \text{let}^{\varphi}\ x = N\ \text{in}\ M_2 > & \\
\text{let}^{\varphi}\ x = N\ \text{in in}_i(M) & \to \ \text{in}_i(\text{let}^{\varphi}\ x = N\ \text{in}\ M) \\
\text{let}^{\varphi}\ x = N\ \text{in}\ \lambda y.M & \to \ \lambda y.\text{let}^{\varphi}\ x = N\ \text{in}\ M \\
\text{let}^{\varphi}\ x = N\ \text{in}\ \varepsilon(y) & \to \ \varepsilon(y) \\
\text{let}^{\varphi}\ x = N\ \text{in let}^{\psi}\ y = K\ \text{in}\ L & \to \\
\qquad \text{let}^{\psi}\ y = (\text{let}^{\varphi}\ x = N\ \text{in}\ K)\ \text{in}\ (\text{let}^{\varphi}\ x = N\ \text{in}\ L) &
\end{array}$$

7.6.3. Definition.

1. Define the *degree* $d(\varphi)$ of a formula φ by:

$$d(\bot) = d(\alpha) = 0, \qquad \text{for } \alpha \in PV;$$

   and

$$d(\varphi \wedge \psi) = d(\varphi \vee \psi) = d(\varphi \to \psi) = 1 + \max\{d(\varphi), d(\psi)\}.$$

2. Define the *degree* $d(M)$ of a term M as the maximal degree of any φ in any $\text{let}^{\varphi}\ x = K \text{ in } L$ in M.

3. Define the height $h(M)$ of a term M as the height of M viewed as a tree.

7.6.4. Lemma. *Let $d = d(\varphi)$ and assume*

$$\Gamma \vdash^{+}_{L} let^{\varphi}\ x = M\ in\ N : \psi$$

where $d(M) < d$ and $d(N) < d$. Then $let^{\varphi}\ x = M\ in\ N \twoheadrightarrow_a P$ for some P with $\Gamma \vdash^{+}_{L} P : \psi$ and $d(P) < d$.

Proof. By induction on $h(M) + h(N)$. Split into cases according to the form of M and N. □

7.6.5. Proposition. *If $\Gamma \vdash^{+}_{L} M : \varphi$ and $d(M) > 0$ then $M \twoheadrightarrow_a N$ for some N with $\Gamma \vdash^{+}_{L} N : \varphi$ and $d(M) > d(N)$.*

Proof. By induction on M using the lemma. □

7.6.6. Theorem (Gentzen). *If $\Gamma \vdash^{+}_{L} M : \varphi$ then $M \twoheadrightarrow_a N$ where $\Gamma \vdash^{+}_{L} N : \varphi$ and $d(N) = 0$, i.e., N represents a cut-free proof.*

Proof. By induction on $d(M)$ using the Proposition. □

What does the system introduced above correspond to, computationally speaking? The rules are similar to the rules that one finds in systems for *explicit substitution*—see, e.g., [14]. It would be interesting to investigate this in greater detail—this has been done recently [113].

7.7. Exercises

7.7.1. Exercise. Give sequent calculus proofs for the formulas of Example 2.2. To prove the odd-numbered formulas use only sequents with single formulas at right-hand sides.

7.7.2. Exercise. Show that all cut-free proofs for the even-numbered formulas of Example 2.2 must involve sequents with more than one formula at the right-hand side.

7.7.3. Exercise. Design a sequent calculus allowing empty right-hand sides of sequents. Does it make sense now to have a right rule for $\bot$?

7.7.4. Exercise. Prove Proposition 7.4.3. On the basis of this proof describe algorithms translating a normal deduction into a cut-free proof and conversely.

7.7.5. Exercise. Give examples of cuts that are assigned terms in normal form, according to the term assignment of Section 7.4.

7.7.6. Exercise. Show that the Curry-style variant of lambda-calculus with $\wedge$ does not have the subject reduction property for η-reductions. Show that the eta rule for $\vee$ has the subject reduction property.

7.7.7. Exercise. Design a Church-style calculus with $\wedge$ and $\vee$ and show that subject reduction property holds for that calculus.

7.7.8. Exercise. Explain the difference between the above two results.

7.7.9. Exercise. Can you design a reasonable eta-rule for disjunction aiming at erasing elimination-introduction pairs? Why not?

7.7.10. Exercise. Define an eta-rule for $\bot$.

7.7.11. Exercise. Let A denote the term language of Definition 7.6.1, except that in A, cut terms have the form let* $x = M$ in N (the φ is omitted). Let L denote the set of λ-terms with pairs and sums and ε.

Let $\vdash_L$ denote typability in $\lambda\!\rightarrow$ with pairs and sums and $\bot$, and let $\vdash_A$ denote typability in the sense of Definition 7.6.1 with the obvious modification to the cut rule to accomodate the change in the syntax of cut terms.

Let $\twoheadrightarrow_A$ denote the reduction relation from Definition 7.6.2, and let $\twoheadrightarrow_L$ denote the transitive, reflexive closure of $\rightarrow_\beta$ plus the reductions on pairs and sums. $=_A$ and $=_L$ are the obvious closures.

Use the proof of Proposition 7.2.2 to give translations $t_L : A \rightarrow L$ and $t_A : L \rightarrow A$ between A and L. Note that these can also be viewed as translations on type-free terms.

Which of the following properties hold for your translations?

1. $\Gamma \vdash_L M : \varphi \Leftrightarrow \Gamma \vdash_A t_A(M) : \varphi$;
2. $\Gamma \vdash_A M : \varphi \Leftrightarrow \Gamma \vdash_L t_L(M) : \varphi$;
3. $M \twoheadrightarrow_L N \Leftrightarrow t_A(M) \twoheadrightarrow_A t_A(N)$;
4. $M \twoheadrightarrow_A N \Leftrightarrow t_L(M) \twoheadrightarrow_L t_L(N)$;
5. $t_L(t_A(M)) =_L M$;
6. $t_A(t_L(M)) =_A M$.

What happens if you add commuting conversion to the relations $\rightarrow_L$, $=_L$, etc.?

CHAPTER 8

Classical logic and control operators

In the previous chapters we have encountered the Curry-Howard isomorphism in various incarnations; each of these state a correspondence between some system of typed terms and a system of formal logic.

Until now these systems of formal logic have been *constructive;* that is, in none of them have we found the *principle of the excluded middle* or the *double negation elimination principle* that one finds in *classical logics.*

This is by no means a coincidence. Until around 1990 there was a widespread consensus to the effect that "there is no Curry-Howard isomorphism for classical logic." However, at that time Tim Griffin made a path-breaking discovery which have convinced most critics that classical logics have something to offer the Curry-Howard isomorphism.

In this chapter we introduce classical propositional logic, we study how one can assign terms to classical proofs, and we present a system for classical proof normalization. The connection between classical and intuitionistic logic is also elaborated in some detail. Griffin's discovery is then presented at the end of the chapter.

8.1. Classical propositional logic, implicational fragment

Although the bulk of the previous chapters have been concerned with formulations of intuitionistic propositional logic we have occasionally come across classical propositional logic.

For instance, in Chapter 2, we briefly studied the algebraic semantics of classical logic, and in the preceding chapter, we introduced sequent calculus for intuitionistic logic as the restriction of classical sequent calculus to one-formula right hand sides.

In the case of natural deduction, there are several ways to obtain classical propositional logic from intuitionistic propositional logic. The following gives one way of doing this for the implicational fragment.

8.1.1. REMARK. In order to avoid confusion and lengthy remarks it is convenient in this chapter to have available a systematic way of assigning names to subsets of the language of propositions and to logical system and typed λ-calculi.

In this chapter, $L(\to)$ denotes the set of *implicational formulas,* i.e., the language generated by the grammar:

$$L(\to) \ni \varphi ::= \bot \mid \alpha \mid \varphi \to \varphi'$$

The *full propositional language* $L(\to, \vee, \wedge)$ is the language generated by the grammar:

$$L(\to, \vee, \wedge) \ni \varphi ::= \bot \mid \alpha \mid \varphi \to \varphi' \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi'$$

We shall occasionally be concerned with the set $L^-(\to)$ of *pure implicational formulas,* i.e., the language generated by

$$L^-(\to) \ni \varphi ::= \alpha \mid \varphi \to \varphi'$$

Similarly, the *pure full propositional language* $L^-(\to, \vee, \wedge)$ is the language generated by the grammar:

$$L^-(\to, \vee, \wedge) \ni \varphi ::= \alpha \mid \varphi \to \varphi' \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi'$$

We will follow similar naming conventions for logical systems and typed λ-calculi.

8.1.2. DEFINITION (Classical propositional logic, implicational fragment). Let φ, ψ range over implicational formulas, i.e., over $L(\to)$. As usual, Γ and Δ denote contexts for which we use the standard conventions.

The natural deduction presentation of the implicational fragment CPC$(\to)$ of classical propositional logic is defined by the following axiom and rules:

Axiom:

$$\Gamma, \varphi \vdash \varphi$$

Rules:

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

$$\frac{\Gamma, \varphi \to \bot \vdash \bot}{\Gamma \vdash \varphi}\ (\neg\neg\mathrm{E})$$

8.1.3. PROPOSITION. *Let φ be an implicational formula. Then $\Gamma \vdash \varphi$ iff $\Gamma \models \varphi$ according to the truth-table semantics. In particular $\vdash \varphi$ iff φ is a tautology.*

PROOF. The proof is left as an exercise. □

8.1.4. REMARK. A small variation of the system is obtained by changing rule $(\neg\neg\mathrm{E})$ to

$$\frac{\Gamma \vdash (\varphi \to \bot) \to \bot}{\Gamma \vdash \varphi}\ (\neg\neg\mathrm{E}')$$

It is an easy exercise to see that this change does not affect the set of provable sequents, i.e., $\Gamma \vdash \varphi$ can be derived in the original system iff $\Gamma \vdash \varphi$ can be derived in the modified system.

Since $\neg\varphi$ is defined as $\varphi \to \bot$ a shorter way to express the formula in the hypothesis is $\neg\neg\varphi$, which explains the name *double negation elimination.*

8.1.5. REMARK. In CPC($\to$) one can prove every formula that can be proved in IPC($\to$), the implicational fragment of intuitionistic propositional logic. The latter system contains the axiom and the rules $(\to\mathrm{I})$, $(\to\mathrm{E})$ and $(\bot\mathrm{E})$, so the only problem is to show that $(\bot\mathrm{E})$ holds as a derived rule in CPC($\to$). That is, that we have to show that in CPC($\to$),

$$\Gamma \vdash \bot \;\Rightarrow\; \Gamma \vdash \varphi$$

for any φ. In fact, this is easy. If $\Gamma \vdash \bot$ then also $\Gamma, \varphi \to \bot \vdash \bot$ by an easy weakening lemma, and then $\Gamma \vdash \varphi$ by $(\neg\neg\mathrm{E})$.

8.1.6. REMARK. Another way to define CPC($\to$) is to consider the axiom along with rules $(\to\mathrm{I})$ and $(\to\mathrm{E})$ and then the following two rules:

$$\frac{\Gamma, \varphi \to \psi \vdash \varphi}{\Gamma \vdash \varphi}(\mathrm{P}) \qquad\qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}(\bot\mathrm{E})$$

The left-most rule is called *Peirce's law,* and the right-most one is called *ex falso sequitur quod libet* (from absurdity follows whatever you like).

An alternative is to consider instead the following two rules:

$$\frac{\Gamma, \varphi \to \bot \vdash \varphi}{\Gamma \vdash \varphi}(\mathrm{P}\bot) \qquad\qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}(\bot\mathrm{E})$$

In other words, in the presence of $(\bot\mathrm{E})$, Peirce's law and the special case $\psi = \bot$ are equivalent.

It is an exercise to show that these two systems derive the same sequents as the system introduced in Definition 8.1.2.

8.1.7. REMARK. In the case of Hilbert-style proofs there are also several ways to obtain classical logic from intuitionistic logic. For instance, if one takes absurdity as primitive, one can add to the two axioms (A1) and (A2) of Section 5.3 the principle of double negation elimination:

$$((\varphi \to \bot) \to \bot) \to \varphi.$$

Another possibility is to add Peirce's law in the form

$$((\varphi \to \psi) \to \varphi) \to \varphi,$$

together with ex-falso in the form:

$$\bot \to \varphi.$$

8.1.8. REMARK. If one takes negation as primitive in a Hilbert-style system, one can add to the axioms (A1) and (A2) the third axiom:

$$(\neg\varphi \to \neg\psi) \to (\neg\varphi \to \psi) \to \varphi,$$

which is read: "if $\neg\varphi$ implies both ψ and $\neg\psi$, then it is contradictory to assume $\neg\varphi$, so φ holds."

In case negation is taken as primitive one cannot simply add

$$\neg\neg\varphi \to \varphi \tag{$*$}$$

This may seem a bit strange since above we suggested to add exactly this axiom when $\neg$ is defined in terms of $\bot$. The point is, however, that in defining $\neg$ in terms of $\bot$ we get certain extra axioms for free. For instance, we have above the rule

$$(\varphi \to \psi) \to (\varphi \to (\psi \to \bot)) \to (\varphi \to \bot)$$

and the corresponding axiom

$$(\varphi \to \psi) \to (\varphi \to \neg\psi) \to \neg\varphi$$

does not follow from axioms (A1) and (A2) and the double negation axiom ($*$).

Similar remarks apply to natural deduction systems in which negation is taken as primitive.

Apart from classical propositional logic and intuitionistic propositional logic, there are many other similar propositional logics, although none of them are as fundamental as these two. One can show that classical propositional logic is a *maximal* logic in the sense that, for any axiom scheme φ, either φ is a theorem of classical propositional logic, or addition of φ to classical propositional logic would render the system inconsistent. Such properties are usually known as *Hilbert-Post completeness.*

8.2. The full system

In the previous section we were concerned with the implicational fragment of classical propositional logic. What is required to obtain a definition of the whole system with conjunction and disjunction?

One approach is to add to the language the two connectives $\wedge$ and $\vee$ and adopt the introduction and elimination rules of the system of Section 2.2.

8.2.1. Definition (Classical propositional logic). Let φ, ψ range over formulas in the full propositional language, i.e., over $L(\to, \vee, \wedge)$. As usual, Γ and Δ denote contexts for which we use the standard conventions.

The natural deduction presentation of $\mathrm{CPC}(\to, \vee, \wedge)$, classical propositional logic, is defined by same rules as in Definition 8.1.2 with the addition of the following well-known rules.

$$\frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\,(\wedge\mathrm{I}) \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\,(\wedge\mathrm{E})\,\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\,(\vee\mathrm{I})\,\frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi} \qquad \frac{\Gamma, \varphi \vdash \sigma \quad \Gamma, \psi \vdash \sigma \quad \Gamma \vdash \varphi \vee \psi}{\Gamma \vdash \sigma}\,(\vee\mathrm{E})$$

The above addition does not change anything concerning implicational formulas.

8.2.2. Proposition. *If* $\varphi \in L(\to)$ *and* $\vdash \varphi$ *in* $\mathrm{CPC}(\to, \vee, \wedge)$, *then* $\vdash \varphi$ *in* $\mathrm{CPC}(\to)$.

Proof. Left as an exercise. □

In other words, the full system is conservative over the implicational fragment.

However, there is a more economical approach. In contrast to the situation in intuitionistic propositional logic, we may *define* conjunction and disjunction as derived connectives.

8.2.3. Definition. Let $\varphi \wedge \psi$ and $\varphi \vee \psi$ abbreviate the following formulas, respectively:

$$\begin{aligned} \varphi \wedge \psi &= (\varphi \to \psi \to \bot) \to \bot; \\ \varphi \vee \psi &= (\varphi \to \bot) \to (\psi \to \bot) \to \bot. \end{aligned}$$

8.2.4. Remark. The above definition of $\varphi \vee \psi$ is not standard; one usually takes $\varphi \vee \psi = (\varphi \to \bot) \to \psi$. This definition lacks the double negation on ψ. From a logical point of view this is of little importance, since in classical logic we can pass back and forth between a formula and its double negation. However, the abbreviations in Definition 8.2.3 are more systematic since they arise as special case of a general scheme for representing data types in typed λ-calculi, as we shall see later.

The following states that our definition of $\wedge$ and $\vee$ have the intended behaviour.

8.2.5. Proposition. *Let $\varphi \in L(\to, \vee, \wedge)$ and and let $\varphi' \in L(\to)$ be the implicational formula obtained by replacing every occurrence of $\wedge$ and $\vee$ by their defining formulas according to Definition 8.2.3. Then $\Gamma \vdash \varphi'$ iff $\Gamma \models \varphi$.*

Proof. By completeness $\Gamma \vdash \varphi'$ iff $\Gamma \models \varphi'$. By elementary calculations with truth tables, $\Gamma \models \varphi'$ iff $\Gamma \models \varphi$. □

8.3. Terms for classical proofs

In what way can we extend the Curry-Howard isomorphism to classical propositional logic? In one sense this is easy: we just add a new term constructor $\Delta x.M$ (or $\Delta x{:}\varphi.M$ in the Church variant of the calculus) in the conclusion of the double negation elimination rule.

8.3.1. Remark. It is convenient in this chapter to have available a systematic way of assigning names to subsets of the language of λ-terms with pairs, etc.

Recall that Λ denotes the set of λ-terms, i.e., the language generated by the grammar:

$$\Lambda \ni M ::= x \mid \lambda x.M \mid M\ M'$$

The set of λ-terms, extended with pairs and sums, $\Lambda(\pi_i, \mathrm{in}_i)$ is the language generated by the grammar:

$$\begin{array}{lcl} \Lambda(\pi_i, \mathrm{in}_i) \ni M & ::= & x \mid \lambda x.M \mid M\ M' \mid \ < M, M' > \ \mid \pi_i(M) \mid \\ & & \mathrm{in}_i(M) \mid \mathrm{case}(M; x'.M'; x''.M'') \end{array}$$

We shall occasionally be concerned with the set Λ_ε generated by

$$\Lambda_\varepsilon \ni M ::= x \mid \lambda x.M \mid M\ M' \mid \varepsilon(M)$$

Similarly, we have the language generated by the grammar:

$$\begin{array}{lcl} \Lambda_\varepsilon(\pi_i, \mathrm{in}_i) \ni M & ::= & x \mid \lambda x.M \mid M\ M' \mid \ < M, M' > \ \mid \pi_i(M) \\ & \mid & \mathrm{in}_i(M) \mid \mathrm{case}(M; x'.M'; x''.M'') \mid \varepsilon(M) \end{array}$$

8.3.2. Definition (type-free and simply typed λ_Δ-calculus, $\lambda_\Delta(\to)$). The term language Λ_Δ of type-free λ_Δ-calculus is defined by the grammar:

$$\Lambda_\Delta \ni M ::= x \mid \lambda x.M \mid M\ M' \mid \Delta x.M$$

The simply typed λ_Δ-calculus has as type language the set $L(\to)$. The inference system of simply typed λ_Δ-calculus arises from the system of simply typed λ-calculus à la Curry by addition of the rule

$$\frac{\Gamma, x : \varphi \to \bot \vdash M : \bot}{\Gamma \vdash \Delta x.M : \varphi}$$

8.3.3. Remark. In the Church variant the obvious modifications are made to the term language and to the inference rules. In particular, the double negation elimination rule becomes

$$\frac{\Gamma, x : \varphi \to \bot \vdash M : \bot}{\Gamma \vdash \Delta x{:}\varphi \to \bot . M : \varphi}$$

We then have the usual correspondence (where binary $\vdash$ means derivability in CPC($\to$) and ternary $\vdash$ means typability in simply λ_Δ-calculus à la Curry):

8.3.4. Proposition.

(i) *If* $\Gamma \vdash M : \varphi$ *then* $|\Gamma| \vdash \varphi$*, where* $|\Gamma| = \{\varphi \mid \exists x : (x : \varphi) \in \Gamma\}$.

(ii) *If* $\Gamma \vdash \varphi$ *then there exists an* M *such that* $\Gamma' \vdash M : \varphi$*, where* $\Gamma' = \{x_\varphi : \varphi \mid \varphi \in \Gamma\}$.

The assignment of $\Delta x.M$ to the double negation elimination rule only extends the Curry-Howard isomorphism to classical logic in a very naive sense: we still have no idea what the computational significance of Δ is. However, we shall learn more about this in Section 8.7.

8.4. Classical proof normalization

As we have seen in previous chapters, reduction on proofs is a rather important concept in the proof theory of intuitionistic logic; for classical logic the same is the case, and we now present certain reductions on classical, propositional proofs that have appeared in the literature.

The following gives reductions on λ_Δ-terms à la Church. The corresponding reductions on λ_Δ-terms à la Curry are obtained by erasing all type annotations.

8.4.1. Definition. Define the relation $\to_\Delta$ on Λ_Δ à la Church as the smallest compatible relation containing the following rules:

$$\begin{array}{lll}
(\Delta x{:}\neg(\varphi \to \psi).M)\ N & \to_\Delta & \Delta z{:}\neg\psi.M\{x := \lambda y{:}\varphi \to \psi\ .\ z\ (y\ N)\}; \\
\Delta x{:}\neg\varphi.x\ M & \to_\Delta & M \qquad \text{provided } x \notin \mathrm{FV}(M); \\
\Delta x{:}\neg\varphi.x\ \Delta y{:}\neg\varphi.N & \to_\Delta & \Delta z{:}\neg\varphi.N\{x, y := z\}.
\end{array}$$

where $\{\bullet := \bullet\}$ denotes substitution on λ_Δ-terms defined in the obvious way, and the notion of compatibility is taken in the obvious way relative to the set Λ_Δ. We use the notation NF_Δ etc with the usual meaning.

From the point of view of reduction on proofs, the first of these rules decreases the complexity of formulas to which we apply the double negation elimination rule. The second rule may be regarded as a form of η-rule for Δ. The third rule reduces applications of the double negation elimination rule nested in a certain trivial way.

8.4.2. Warning. The reader should be warned that one finds in the literature different typed λ-calculi corresponding to classical logic that are not merely simple variations of the above one. This is in particular true for the system studied by Parigot—see, e.g, [82]—which is one of the most widely cited approaches.

Also, different authors use different symbols for the term corresponding to variants of the double negation elimination rule, e.g. μ (Parigot [79, 80, 82, 81], Ong [78]), γ (Rezus [91, 92]), and $\mathcal{C}$ (Griffin [49] and Murthy [75, 76]); the Δ is taken from Rehof and Sørensen [89].

8.4.3. Remark. In the literature one finds different sets of reduction rules for classical proofs, although almost every set contains a variant of the first of the above rules.

The following gives a characterization of the normal forms of $\beta\Delta$-reduction, i.e., a characterization of classical proofs without detours.

8.4.4. Definition. Let $\mathcal{N}$ be the smallest class of λ_Δ-terms closed under the rule: $M_1, \dots, M_n \in \mathcal{N} \Rightarrow \lambda x_1. \dots \lambda x_n.\Delta y_1. \dots \Delta y_m.z\ M_1 \dots M_n \in \mathcal{N}$ where $n, m \geq 0$ and z may be an x_i, a y_j, or some other variable.

8.4.5. Proposition. *If* $\Gamma \vdash M : \varphi$ *in simply typed* λ_Δ*-calculus, and* $M \in \mathrm{NF}_{\beta\Delta}$*, then* $M \in \mathcal{N}$.

Proof. By induction on the derivation of $\Gamma \vdash M : \varphi$. □

The proposition states that any normal proof proceeds by first making certain assumptions (reflected by the variable z), then decomposing those assumptions into simpler formulas by elimination rules (reflected by the applications $z\ M_1 \dots M_n$), then using on the resulting formulas some applications of the double negation elimination rule (reflected by $\Delta y_i.\bullet$), and finally by building up from the result more complicated formulas again by introduction rules (reflected by the abstractions $\lambda x_i.\bullet$).

The following shows that the reduction rules are sufficiently strong that the characterization of their normal forms entails consistency of the system.

8.4.6. Corollary. $\nvdash \bot$.

Proof. If $\vdash \bot$ then $\vdash M : \bot$ for some closed M. By the strong normalization theorem—which will be proved later—we can assume that M is in normal form. Then by induction on the derivation of $M \in \mathcal{N}$ show that $\vdash M : \bot$ is impossible. □

8.5. Definability of pairs and sums

We saw in the second section above that one can define conjunction and disjunction in classical propositional logic. Intuitively speaking, at the level of terms, this suggest that we should be able to define pairs and sums. In this section we show that this is indeed possible. For simplicity we work in this section with λ_Δ-terms à la Curry.

First we introduce the extension that corresponds to the full system CPC$(\rightarrow, \vee, \wedge)$.

8.5.1. Definition. The term language $\Lambda_\Delta(\pi_i, \mathrm{in}_i)$ of type-free λ_Δ-calculus extended with pairs and sums is defined by the grammar:

$$\begin{array}{rcl}\Lambda_\Delta(\pi_i, \mathrm{in}_i) \ni M & ::= & x \mid \lambda x.M \mid M\,M' \mid\ < M, M' > \mid \pi_i(M) \\ & \mid & \mathrm{in}_i(M) \mid \mathrm{case}(M; x'.M'; x''.M'') \mid \Delta x.M\end{array}$$

The simply typed λ_Δ-calculus extended with pairs and sums, denoted by $\lambda_\Delta(\rightarrow, \vee, \wedge)$, has as type language the set $L(\rightarrow, \vee, \wedge)$. The inference system of $\lambda_\Delta(\rightarrow, \vee, \wedge)$ arises from that of $\lambda_\Delta(\rightarrow)$ by addition of the usual rules à la Curry for typing pairs and sums.

The following shows how to *define* pairs and sums.

8.5.2. Definition. Define the following abbreviations.

$$\begin{array}{lcl}< P, Q > & = & \lambda z.z\,P\,Q; \\ \pi_i(P) & = & \Delta k.P\ (\lambda x_1.\lambda x_2.k\ x_i); \\ \mathrm{in}_i(P) & = & \lambda y_1.\lambda y_2.y_i\ P; \\ \mathrm{case}(P; x_1.Q_1; x_2.Q_2) & = & \Delta k.P\ (\lambda x_1.kQ_1)\ (\lambda x_2.kQ_2).\end{array}$$

By some elementary calculations, one can then prove the following.

8.5.3. Proposition. *Let $M \in \Lambda_\Delta(\pi_i, in_i)$ and let $M' \in \Lambda_\Delta(\pi_i, in_i)$ be the term obtained by expansion according to the abbreviations in the preceding definition. Let $\varphi \in L(\rightarrow, \vee, \wedge)$ and let $\varphi' \in L(\rightarrow)$ be the formula obtained by replacing every occurrence of $\wedge$ and $\vee$ by their defining formulas according to Definition 8.2.3.*

1. *If $\Gamma \vdash M : \varphi$ in simply typed λ_Δ-calculus extended with pairs and sums, then $\Gamma \vdash M' : \varphi'$ in the simply λ_Δ-calculus.*

2. *If $M \rightarrow_{\beta\Delta} N$ (using reductions for pairs and sums) then $M' \twoheadrightarrow_{\beta\Delta} N'$, where N' is the expansion of N according to the preceding definition.*

The definition of pairs is the standard one from type-free λ-calculus (see Chapter 1), while the projection construction is different from that normally employed in type-free λ-calculus, viz. $M\ (\lambda x_1.\lambda x_2.x_i)$. This latter definition does not work because $\lambda x_1.\lambda x_2.x_i$ has type $\varphi_1 \to \varphi_2 \to \varphi_i$ instead of the type $\varphi_1 \to \varphi_2 \to \bot$, which M expects. Changing the definition of conjunctive types to solve the problem is not possible; it leads to the type of a pair being dependent on which component a surrounding projection picks.[1] The operator Δ solves the problem by means of an application which turns the type of x_i into $\bot$ regardless of i. When the projection is calculated, the k reaches its Δ and can be removed by the second reduction rule for Δ:

$$\begin{array}{lcl} \pi_1(< M_1, M_2 >) & \equiv & \Delta k.(\lambda f.f\ M_1\ M_2)\ \lambda x_1.\lambda x_2.k\ x_1 \\ & \to_\beta & \Delta k.(\lambda x_1.\lambda x_2.k\ x_1)\ M_1\ M_2 \\ & \twoheadrightarrow_\beta & \Delta k.k\ M_1 \\ & \to_\Delta & M_1 \end{array}$$

As mentioned earlier, the definition for disjunctive formulas above is *not* standard in logic. The standard definition is $(\varphi \to \bot) \to \psi$ instead of $(\varphi \to \bot) \to (\psi \to \bot) \to \bot$. However, when one tries to prove the derived inference rules for this translation it turns out that the corresponding constructions for injection and case analysis are very different from those defining pairs and projections. Specifically, to have the desired reduction rule hold derived one would need to add extra power to Δ. The present definition and corresponding defined constructions can be motivated by corresponding definition in second-order logic, which we will encounter in a later chapter.

8.6. Embedding into intuitionistic propositional logic

In this section we shall show that classical logic can be embedded into intuitionistic logic in a certain sense.

Translations like the following have been studied since the 1930s by Kolmogorov, Gentzen, Gödel, and Kuroda.

8.6.1. Definition. Define the translation k from implicational formulas to implicational formulas by:

$$\begin{array}{lcl} k(\alpha) & = & \neg\neg\alpha \\ k(\bot) & = & \neg\neg\bot \\ k(\varphi \to \psi) & = & \neg\neg(k(\varphi) \to k(\psi)) \end{array}$$

We aim to show that if φ is classically provable, then $k(\varphi)$ is intuitionistically provable. We do this by giving a translation of classical proofs of φ into intuitionistic proofs of $k(\varphi)$. More precisely, the translation is stated on terms representing these proofs.

[1]If one is willing to settle for a weaker notion of pairs where both component must have the same type, then this problem vanishes. This shows that pairs with components of the same type can be represented in the simply typed λ-calculus.

8.6.2. Definition. Define the translation t from λ_Δ-terms to λ-terms by:

$$\begin{array}{lcl}
t(x) & = & \lambda k.x\ k \\
t(\lambda x.M) & = & \lambda k.k\ \lambda x.t(M) \\
t(M\ N) & = & \lambda k.t(M)\ (\lambda m.m\ t(N)\ k) \\
t(\Delta x.M) & = & \lambda k.(\lambda x.t(M))\ (\lambda h.h\ \lambda j.\lambda i.i\ (j\ k))\ \lambda z.z
\end{array}$$

The following shows that k defines an embedding of classical logic into intuitionistic logic.

8.6.3. Proposition. *If $\Gamma \vdash M : \varphi$ in simply typed λ_Δ-calculus, then $k(\Gamma) \vdash t(M) : k(\varphi)$ in simply typed λ-calculus.*

This gives another proof of consistency of classical propositional logic: if classical logic is inconsistent, so is intuitionistic logic.

8.6.4. Proposition. *$\nvdash \perp$ in classical propositional logic.*

Proof. If $\vdash M : \perp$ in classical propositional logic, then $\vdash t(M) : \neg\neg\perp$ in intuitionistic logic, and then $\vdash t(M)\ \lambda z.z : \perp$ in intuitionistic logic, a contradiction. □

The above proof gives a conservativity result: if $\perp$ is provable in classical logic, $\perp$ is provable already in intuitionistic logic. The construction can be generalized to other formulas than $\perp$; in fact, this way one can prove that any formula of form $\forall x \exists y : P(x, y)$, where P is a primitive recursive predicate, is provable in classical arithmetic (i.e., Peano Arithmetic) iff it is provable in intuitionistic arithmetic (i.e., Heyting Arithmetic). Formulas of this form are quite important since they include, e.g., every assertion that some algorithm terminates ("for any input x there is a terminating computation y"). In other words, as concerns provability of termination of algorithms there is no difference between intuitionistic and classical logic.

On the one hand, this means that constructivists should raise no objection to the use of classical logic in this special case since any classical proof can be converted into an intuitionistic one of the same formula. Conversely, classical logicians cannot claim that any logical strength is lost by the restriction to intuitionistic logic.

The following shows that the translation t internalizes Δ conversion by β-conversion.

8.6.5. Proposition. *If $M =_{\beta\Delta} N$ then $t(M) =_\beta t(N)$.*

By analyzing the connection between Δ and β in some more detail, one can prove:

8.6.6. Proposition. *If $M \in \infty_{\beta\Delta}$ then $t(M) \in \infty_\beta$.*

8.6.7. Corollary. *The relation $\rightarrow_{\beta\Delta}$ is strongly normalizing on typable terms.*

Proof. By the preceding proposition, Proposition 8.6.3, and strong normalization of simply typed λ-calculus. □

8.7. Control operators and CPS translations

So far we have not revealed what the computational meaning of Δ is; in this section we finally release the suspension: Δ is a control operator!

Control operators appear in functional programming languages like Scheme (Call/cc), ML (exceptions), Lisp (catch and throw).

Let us illustrate, by way of example, how control operators can be used to program efficiently and concisely. We shall consider the problem of writing a function `M` which takes a binary tree of integer nodes and returns the result of multiplying all the node values. Of course, this problem raises the efficiency issue of what to do when a node value of 0 is encountered.
Our example programs will be written in syntactically sugared SCHEME, the sugar being that instead of `(define` M N`)` we write M `=` N and instead of `(lambda (x)` M`)` we write λ `x.`M, and for `(letrec ([f t]) t')` we write `(let f = t in t')`.

Our first solution is the straightforward purely functional one which trades efficiency off for elegance. We suppose given auxiliary functions `mt?` testing for the empty tree, `num` selecting the node value of the root node, `lson` and `rson` returning the left and right subtrees.

8.7.1. Example. (Functional, elegant, inefficient version)
```
M1 = λ t.(if (mt?  t)
        1
        (* (num t) (* (M1 (lson t)))(M1 (rson t)))))
```

One can optimize `M1` so as to stop multiplying as soon as a node value of 0 is encountered. This can be done in purely functional style, by means of tests. Our next solution embodies that strategy. Here we assume a constructor `EX` and a test function `EX?` such that `(EX` M`)` tags M with the marker `EX`, and `EX?` tests for the presence of the tag. Furthermore, we assume a 0 test function `zero?`. The reader will probably agree that elegance (or at the least conciseness) has now been traded off for efficiency.

8.7.2. Example. (Functional, inelegant, efficient version)
```
M2 = λ t.(if (mt?  t)
        1
        (if (zero?  (num t))
              (EX 0)
              (let ((l (M2 (lson t))))
                   (if (EX? l)
                        (let ((r (M2 (rson t))))
                             (if (EX? r)
                                  r
                                  (* (num t) (* l r))))))))
```

The function `M2` will return an integer, the product of all the tree nodes, if no node contains 0, or the value `(EX 0)` if any node contains 0. We may see the `EX` tag as a kind of *exception* marker which is propagated explicitly up the recursion tree. In this vein one could view the `EX` constructor as an injection function taking an integer to an element of a sum type of the form *int* $\vee$ *ex*. Now, the `catch`/`throw` mechanism is well suited to handle exactly this kind of problem where an exceptional value is propagated. Efficiency is enhanced by `catch` and `throw` because all the propagation is done in a single step (or jump, as we might say). This leaves us with a relatively elegant and efficient but non functional version, as shown in the next example program.

8.7.3. Example. (Non functional, elegant, efficient version)

```
M3 = λ t.  catch j in
       (let L = λ t'.(if (mt?  t')
              1
              (if (zero?  (num t'))
                   (throw j 0)
                   (* (num t')
                   (* (L (lson t'))(L (rson t'))))))
       in (L t))
```

It is an interesting fact that the mechanism used in Example 8.7.3 can be internalized in the purely functional part of the language by the so-called CPS-transformation technique. Applying that translation to the program with catch and throw gives:

8.7.4. Example. (CPS version of `M3`)

```
M4 = λ t.  λ k.
       (if (mt?  t)
              (k 1)
              (if (zero?  (num t))
                   0
                   ((M4 (lson t))
                        (λ l.((M4 (rson t))
                        (λ r.(k (* (num t) (* l r)))))))))
```

The non-functional program can be written as follows in the type-free λ_Δ-calculus.

```
M = λt.Δ j.j
      (Y (λf.λt'. (if (mt? t')
             1
             (if (zero? (num t'))
                  ε(j 0)
                  (* (num t')
                  (* (f (lson t'))(f (rson t'))))))) t),
```

where $\mathbf{Y}$ denotes Church's fixpoint combinator and $\varepsilon(M)$ abbreviates $\Delta x.M$, for $x \notin \mathrm{FV}(M)$.

It is instructive to verify that, e.g., for T $\equiv$ `<2,<0,nil,nil>,nil>`, we have MT $= 0$, noticing how an "exception" is raised as the node value 0 is encountered.

In conclusion, the Δ may be regarded as a control operator similar to call/cc of Scheme and exceptions of ML, and the double negation embedding of classical logic into intuitionisitic logic corresponds to well-known CPS-translations of terms with control operators into pure, control-operator-free languages.

8.8. Historical remarks

Felleisen and his co-workers studied λ-calculi with control operators in an untyped setting. Their aim was to provide a foundation of type-free functional programming langauges with control operators similarly to the way λ-calculus may be regarded as a foundation of type-free pure functional programming languages.

Felleisen devised a control calculus, an extension of the λ-calculus, and carried out what could aptly be called *Plotkin's program* (see [83]) for the study of the relation between calculi and programming languages.

Griffin discovered in 1990, in an attempt to incorporate control operators into the world of typed λ-calculi, that Felleisen's $\mathcal{C}$-operator could be typed by the classical double negation elimination rule [49]. Using this rule does, however, lead to certain difficulties because typing is not in general preserved under reduction ("Failure of Subject Reduction.") This defect was repaired by Griffin via a so-called computational simulation.

Later, Murthy overcame the same difficulties by changing the type system into a so-called pseudo-classical logic. Applying conservativity results of classical logics over corresponding minimal logics Murthy showed in [75] that for a certain class of classically provable formulas the Realizability Interpretation remains sound. This was done using CPS-translations of control operator calculi into pure λ-calculi.

Since the seminal work of Griffin and Murthy, numerous systems have appeared that connect classical logic and control operators; indeed, the study of classical logic in connection with the Curry-Howard isomorphism now constitutes a separate field.

8.9. Exercises

8.9.1. Exercise. Prove Proposition 8.1.3.

8.9.2. Exercise.

1. Show that Peirce's law can be derived from the special case of Peirce's law and ex-falso.
2. Show that double negation elimination can be derived from the special case of Peirce's law and ex-falso.
3. Show that the special case of Peirce's law and ex-falso can both be derived from double negation elimination.

8.9.3. Exercise. Prove that $\Gamma \vdash \varphi$ can be derived in the system of Definition 8.1.2 iff $\Gamma \vdash \varphi$ can be derived in the system of Remark 8.1.4.

8.9.4. Exercise. Let φ be some implicational formula such that $\not\vdash \varphi$ in CPC($\to$), and consider the system Z which arises from CPC($\to$) by adding all instances (substituting implicational formulas for propositional variables) of φ as axioms. Show that $\vdash \bot$ in Z.

8.9.5. Exercise. prove Proposition 8.2.2.

8.9.6. Exercise. Show that in CPC($\to, \vee, \wedge$) one can derive $\vdash \alpha \vee \neg\alpha$.

8.9.7. Exercise. The rule

$$(\Delta x{:}\neg(\varphi \to \psi).M)\ N \to_\Delta \Delta z{:}\neg\psi.M\{x := \lambda y{:}\varphi \to \psi\ .\ z\ (y\ N)\}$$

can only reduce Δ's inside an application. The following aggressive variant does not wait for the application:

$$(\Delta x{:}\neg(\varphi \to \psi).M) \to_\Delta \lambda a{:}\varphi\ .\ \Delta z{:}\neg\psi.M\{x := \lambda y{:}\varphi \to \psi\ .\ z\ (y\ a)\}$$

Corresponding to these two different rules, give reduction rules for the constructs $\pi_i(\Delta x{:}\neg(\varphi \wedge \psi).M)$ and $\Delta x{:}\neg(\varphi \wedge \psi).M$.

Can you give corresponding rules for disjunction?

8.9.8. Exercise. Show that $\to_{\beta\Delta}$ on λ_Δ-terms à la Curry is Church-Rosser, or show that $\to_{\beta\Delta}$ on λ_Δ-terms à la Curry is not Church-Rosser.

Same question for $\to_{\beta\Delta}$ on terms à la Church.

8.9.9. Exercise. Does the following hold? For any $M \in \Lambda_\Delta$, $M \in \mathcal{N}$ iff $M \in \mathrm{NF}_{\beta\Delta}$.

CHAPTER 9

First-order logic

In this chapter we extend our consideration to formulas with quantifiers and generalize the proof systems and interpretations seen in earlier chapters to the first-order case.

9.1. Syntax of first-order logic

The objects investigated by propositional logic are compound statements, built from some atomic statements (represented by propositional variables) by means of logical connectives. The goal is to understand relations between compound statements depending on their structure, rather than on the actual "meaning" of the atoms occurring in these statements. But mathematics always involves reasoning about *individual objects*, and statements about properties of objects can not always be adequately expressed in the propositional language. The famous syllogism is an example:

All humans are mortal;
Socrates is a human;
Therefore Socrates is mortal.

To express this reasoning in the language of formal logic, we need to quantify over individual objects (humans), and of course we need predicates (relations) on individual objects. The logical systems involving these two features are known under the names "*predicate logic*", "*predicate calculus*", "*quantifier calculus*" or "*first-order logic*". This section describes a variant of the first-order syntax. Such syntax is always defined with respect to a fixed first-order signature Σ, which is typically assumed to be finite. Recall from Chapter 6 that a signature is a family of function, relation and constant symbols, each with a fixed arity. Also recall that *algebraic terms* over Σ are individual variables, constants and expressions of the form $(f t_1 \ldots t_n)$, where f is an n-ary function symbol, and $t_1, \ldots, t_n$ are algebraic terms over Σ.

9.1.1. Definition.

1. An *atomic formula* is an expression of the form $(rt_1 \dots t_n)$, where r is an n-ary relation symbol, and $t_1, \dots, t_n$ are algebraic terms over Σ.
2. The set Φ_Σ of *first-order formulas* over Σ, is the least set such that:
   - All atomic formulas and $\bot$ are in Φ_Σ;
   - If $\varphi, \psi \in \Phi_\Sigma$ then $(\varphi \to \psi), (\varphi \vee \psi), (\varphi \wedge \psi) \in \Phi_\Sigma$;
   - If $\varphi \in \Phi_\Sigma$ and x is an individual variable, then $\forall x\, \varphi, \exists x\, \varphi \in \Phi_\Sigma$.
3. As usual, we abbreviate $(\varphi \to \bot)$ as $\neg\varphi$ and $((\varphi \to \psi) \wedge (\psi \to \varphi))$ as $(\varphi \leftrightarrow \psi)$.
4. A formula is *open* iff it contains no quantifiers.

9.1.2. Convention. The parentheses-avoiding conventions used for propositional formulas apply as well to first-order formulas. However, there is no general consensus about how quantifiers should be written. One convention, which is common in logic, is that a quantifier is an operator of highest priority, so that "$\forall x\, \varphi \to \psi$" stands for "$(\forall x\, \varphi) \to \psi$". The other convention seems to have originated in type theory and is that the quantifier scope extends as much to the right as possible. In order to avoid confusion, we will sometimes use extra parentheses and sometimes we will use an explicit dot notation. Namely, we will write "$\forall x.\varphi \to \psi$" for "$\forall x(\varphi \to \psi)$" and "$\forall x\varphi. \to \psi$" instead of "$(\forall x\varphi) \to \psi$". (Needless to say, authors sometimes forget about such conventions. The reader should be warned that each of the two authors has got used to a different style.)

9.1.3. Definition.

1. If t is an algebraic term then $FV(t)$ stands for the set of all variables occurring in t.
2. The set $FV(\varphi)$ of free variables of a formula φ is defined by induction:
   - $FV(rt_1 \dots t_n) = FV(t_1) \cup \dots \cup FV(t_n)$;
   - $FV(\varphi \to \psi) = FV(\varphi \vee \psi) = FV(\varphi \wedge \psi) = FV(\varphi) \cup FV(\psi)$;
   - $FV(\forall x\, \varphi) = FV(\exists x\, \varphi) = FV(\varphi) - \{x\}$.
3. A *sentence*, also called a *closed formula*, is a formula without free variables.

9.1.4. Definition. The definition of a substitution of a term for an individual variable, denoted $\varphi[x := t]$, respects the quantifiers as variable-binding operators, and thus must involve variable renaming. Formally,

$x[x := t] = t$;
$y[x := t] = y$, if $y \neq x$;
$(ft_1 \dots t_n)[x := t] = ft_1[x := t] \dots t_n[x := t]$;
$(rt_1 \dots t_n)[x := t] = rt_1[x := t] \dots t_n[x := t]$;
$(\forall x\varphi)[x := t] = \forall x\varphi$;
$(\forall y\varphi)[x := t] = \forall y\varphi[x := t]$, if $y \neq x$, and $y \notin \mathrm{FV}(t)$ or $x \notin \mathrm{FV}(\varphi)$;
$(\forall y\varphi)[x := t] = \forall z\varphi[y := z][x := t]$, if $y \neq x$ and $y \in \mathrm{FV}(t)$ and $x \in \mathrm{FV}(\varphi)$;
$(\exists x\varphi)[x := t] = \exists x\varphi$;
$(\exists y\varphi)[x := t] = \exists y\varphi[x := t]$ if $y \neq x$, and $y \notin \mathrm{FV}(t)$ or $x \notin \mathrm{FV}(\varphi)$;
$(\exists y\varphi)[x := t] = \exists z\varphi[y := z][x := t]$ if $y \neq x$ and $y \in \mathrm{FV}(t)$ and $x \in \mathrm{FV}(\varphi)$.
where z is a fresh variable.

The *simultaneous substitution*, written $\varphi[x_1 := t_1, \dots, x_n := t_n]$, is the term

$$\varphi[x_1 := s_1] \cdots [x_n := s_n][y_1^1 := z_1^1] \cdots [y_{m_1}^1 := z_{m_1}^1] \cdots [y_1^n := z_1^n] \cdots [y_{m_n}^n := z_{m_n}^n],$$

where $z_1^i, \dots, z_{m_i}^i$ are all variables in t_i, the variables y_j^i are all fresh and different, and $s_i = t_i[z_1^i := y_1^i] \cdots [z_{m_i}^i := y_{m_i}^i]$, for all i.

9.1.5. CONVENTION. It is a common convention to write e.g., $\varphi(x, y, z)$ instead of φ, to stress that x, y, z may occur in φ. In this case, the notation like $\varphi(t, s, u)$ is used as a shorthand for $\varphi[x := t, y := s, z := u]$. We will also use this convention, but one should be aware that it is not a part of the syntax.

It is not customary to introduce alpha-conversion on first-order formulas. Typically, alpha-equivalent formulas are considered different, although they are equivalent with respect to all reasonable semantics, and one can be derived from another with all reasonable proof systems. However, for uniformity of our presentation, we prefer to allow for the alpha-conversion (defined in the obvious way) and identify alpha-convertible formulas from now on.

9.2. Intuitive semantics

The Brouwer-Heyting-Kolmogorov interpretation of propositional formulas (Chapter 2) extends to first-order logic as follows:

- *A construction of* $\forall x\, \varphi(x)$ *is a method (function) transforming every object* $\mathbf{a}$ *into a construction of* $\varphi(\mathbf{a})$.
- *A construction of* $\exists x\, \varphi(x)$ *is a pair consisting of an object* $\mathbf{a}$ *and a construction of* $\varphi(\mathbf{a})$.

Note that the BHK-interpretation should be taken with respect to some domain of "objects". These objects are syntactically represented by algebraic terms.

9.2.1. EXAMPLE. Consider the following formulas:

1. $\neg\exists x\,\varphi. \leftrightarrow \forall x\neg\varphi$;
2. $\neg\forall x\,\varphi. \leftrightarrow \exists x\neg\varphi$;
3. $(\psi \to \forall x\,\varphi(x)) \leftrightarrow \forall x(\psi \to \varphi(x))$, where $x \notin FV(\psi)$;
4. $(\psi \to \exists x\,\varphi(x)) \leftrightarrow \exists x(\psi \to \varphi(x))$, where $x \notin FV(\psi)$;
5. $\forall x(\varphi \to \psi) \to (\exists x\,\varphi. \to \exists x\,\psi)$;
6. $\forall x(\psi \vee \varphi(x)) \leftrightarrow \psi \vee \forall x\,\varphi(x)$, where $x \notin FV(\psi)$;
7. $\forall x(\varphi \to \psi). \to (\forall x\,\varphi. \to \forall x\,\psi)$;
8. $(\forall x\,\varphi(x). \to \psi) \to \exists x(\varphi(x) \to \psi)$, where $x \notin FV(\psi)$;
9. $\forall x\,\varphi(x). \to \varphi(t)$;
10. $\neg\neg\forall x(\varphi \vee \neg\varphi)$;
11. $\psi \to \forall x\,\psi$, where $x \notin FV(\psi)$;
12. $\exists x(\exists y\,\varphi(y). \to \varphi(x))$

Although all these formulas are all classical first-order tautologies,[1] one will have difficulties finding BHK-interpretations for some of them.

9.2.2. REMARK. It should be no surprise when we say that universal quantification is a generalization of conjunction. Indeed, the sentence "all cats have tails" is quite like an infinite conjunction of statements concerning each individual cat separately. In quite the same spirit one can say that existential quantification is a generalized disjunction. This idea is reflected by the algebraic semantics, where we interpret quantifiers as (possibly infinite) joins and meets, see Definitions 9.4.3 and 9.4.4.

But the BHK-interpretation as above hints for another correspondence: between universal quantification and implication, because in both cases we have a function as a construction. The analogy is so strong that in certain systems with quantifiers, implication is just syntactic sugar. We will see it in Chapters 10 and 13.

9.3. Proof systems

The three main approaches: natural deduction, sequent calculus and the Hilbert style extend to first-order logic by adding suitable rules and axioms to the rules and axiom schemes for propositional logic. The notation $\vdash_N$, $\vdash_L$, etc., is the obvious modification from the propositional case.

[1]We assume that the reader is familiar with classical first-order logic. A suggested textbook is e.g. [70].

Natural deduction

We extend the system of natural deduction with the following rules to introduce and eliminate quantifiers:

$$(\forall\mathrm{I})\frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall x\,\varphi}\ (x \notin FV(\Gamma)) \qquad (\forall\mathrm{E})\frac{\Gamma \vdash \forall x\,\varphi}{\Gamma \vdash \varphi[x := t]}$$

$$(\exists\mathrm{I})\frac{\Gamma \vdash \varphi[x := t]}{\Gamma \vdash \exists x\,\varphi} \qquad (\exists\mathrm{E})\frac{\Gamma \vdash \exists x\,\varphi \quad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}\ (x \notin FV(\Gamma, \psi))$$

The reader should be warned that our rules are such because we have agreed on alpha-conversion of formulas. Otherwise, one has to modify rules (∀I) and (∃E) to work for any alpha-variant of the quantifier bindings. Similar modifications would have to be done on the other proof systems to follow.

Sequent calculus

Here are classical sequent calculus rules for quantifiers. Note the symmetry between the two quantifiers.

$$(\forall\mathrm{L})\frac{\Gamma, \varphi[x := t] \vdash \Sigma}{\Gamma, \forall x\,\varphi \vdash \Sigma} \qquad (\forall\mathrm{R})\frac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \forall x\,\varphi, \Delta}\ (x \notin FV(\Gamma, \Delta))$$

$$(\exists\mathrm{L})\frac{\Gamma, \varphi \vdash \Sigma}{\Gamma, \exists x\,\varphi \vdash \Sigma}\ (x \notin FV(\Gamma, \Sigma)) \qquad (\exists\mathrm{R})\frac{\Gamma \vdash \varphi[x := t], \Delta}{\Gamma \vdash \exists x\,\varphi, \Delta}$$

To obtain intuitionistic sequent calculus we restrict ourselves to single formulas at the right-hand sides (Σ consists of a single formula and Δ is always empty).

$$(\forall\mathrm{L})\frac{\Gamma, \varphi[x := t] \vdash \sigma}{\Gamma, \forall x\,\varphi \vdash \sigma} \qquad (\forall\mathrm{R})\frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall x\,\varphi}\ (x \notin FV(\Gamma))$$

$$(\exists\mathrm{L})\frac{\Gamma, \varphi \vdash \sigma}{\Gamma, \exists x\,\varphi \vdash \sigma}\ (x \notin FV(\Gamma, \sigma)) \qquad (\exists\mathrm{R})\frac{\Gamma \vdash \varphi[x := t]}{\Gamma \vdash \exists x\,\varphi}$$

9.3.1. THEOREM (Cut elimination). *For all φ and Γ, if the sequent $\Gamma \vdash \varphi$ has a proof then it has a cut-free proof.*

The following was probably first shown by Rasiowa and Sikorski using the topological space semantics, see [88].

9.3.2. COROLLARY (Existence property). *If $\vdash \exists x\,\varphi$ then there exists a term t such that $\vdash \varphi[x := t]$.*

PROOF. The last rule in a cut-free proof of $\vdash \exists x\, \varphi$ must be ($\exists$R). $\square$

Note that if the signature consists of relation symbols only (and thus the only terms are variables) then $\vdash \exists x\, \varphi$ implies $\vdash \forall x\, \varphi$!

Hilbert-style proofs

It is difficult to find two different authors who would give identical Hilbert-style proof systems.[2] Our choice is as follows. We take as axioms all the propositional axiom schemes, and in addition all formulas of the form:

- $\forall x\, \varphi(x). \to \varphi(t)$;
- $\varphi(t) \to \exists x\, \varphi(x)$;
- $\psi \to \forall x\, \psi$, where $x \notin FV(\psi)$;
- $\exists x\, \psi. \to \psi$, where $x \notin FV(\psi)$;
- $\forall x(\varphi \to \psi). \to (\exists x\, \varphi. \to \exists x\, \psi)$;
- $\forall x(\varphi \to \psi). \to (\forall x\, \varphi. \to \forall x\, \psi)$.

As inference rules of our system we take *modus ponens* and the following *generalization rule*:

$$\frac{\varphi}{\forall x\, \varphi}$$

The use of generalization requires some caution (corresponding to the side conditions in rules ($\forall$I) and ($\forall$R)).

9.3.3. DEFINITION. A formal *proof* of a formula φ from a set Γ of assumptions is a a finite sequence of formulas $\psi_1, \psi_2, \ldots, \psi_n$, such that $\psi_n = \varphi$, and for all $i = 1, \ldots, n$, one of the following cases takes place:

- ψ_i is an axiom;
- ψ_i is an element of Γ;
- there are $j, \ell < i$ such that $\psi_j = \psi_\ell \to \psi_i$ (i.e., ψ_i is obtained from ψ_j, ψ_ℓ using *modus ponens*);
- there is $j < i$ such that $\psi_i = \forall x\, \psi_j$, for some $x \notin FV(\Gamma)$ (i.e., ψ_i is obtained from ψ_j by generalization).

[2]Unless they are *co-authors*, of course.

9.3.4. Warning. In many textbooks, the above definition of a proof does not include the restriction on applicability of the generalization rule. This does not matter as long as Γ is a set of sentences (what is typically assumed). In general however, this gives a different relation $\vdash_H^\circ$, such that

$$\Gamma \vdash_H^\circ \varphi \quad \text{iff} \quad \Gamma^\circ \vdash_H \varphi,$$

where Γ° is obtained from Γ by binding all free variables by universal quantifiers placed at the beginning of all formulas. For this relation, the deduction theorem (see below) would only hold for Γ consisting of sentences.

9.3.5. Lemma (Deduction Theorem).
The conditions $\Gamma, \varphi \vdash_H \psi$ and $\Gamma \vdash_H \varphi \to \psi$ are equivalent.

Proof. Easy. □

9.3.6. Theorem. *Natural deduction, sequent calculus, and the above Hilbert-style proof system are all equivalent, i.e., $\Gamma \vdash_N \varphi$ and $\Gamma \vdash_L \varphi$ and $\Gamma \vdash_H \varphi$ are equivalent to each other for all Γ and φ.*

Proof. Boring. □

Translations from classical logic

Double negation translations from classical to intuitionistic logic can be extended to the first-order case. We add the following clauses to the definition of the Kolmogorov translation of Chapter 8:

- $t(\forall x\, \varphi) := \neg\neg\forall x\, t(\varphi)$;
- $t(\exists x\, \varphi) := \neg\neg\exists x\, t(\varphi)$.

and we still have the following result:

9.3.7. Theorem. *A formula φ is a classical theorem iff $t(\varphi)$ is an intuitionistic theorem.*

Proof. Exercise 9.5.6 □

Since classical provability reduces to intuitionistic provability, and classical first-order logic is undecidable, we obtain undecidability of intuitionistic first-order logic as a consequence.

9.3.8. Corollary. *First-order intuitionistic logic is undecidable.*

In fact, the undecidability result holds already for a very restricted fragment of first-order intuitionistic logic, with $\forall$ and $\to$ as the only connectives and with no function symbols. In particular, there is no need for negation or falsity.

9.4. Semantics

We begin with classical semantics. Assume that our signature Σ consists of the function symbols $f_1, \dots, f_n$, relation symbols $r_1, \dots, r_m$ and constant symbols $c_1, \dots, c_k$.

9.4.1. Definition. A *structure* or *model* for Σ is an algebraic system $\mathcal{A} = \langle A, f_1^{\mathcal{A}}, \dots, f_n^{\mathcal{A}}, r_1^{\mathcal{A}}, \dots, r_m^{\mathcal{A}}, c_1^{\mathcal{A}}, \dots, c_k^{\mathcal{A}} \rangle$, where the $f_i^{\mathcal{A}}$'s and $r_i^{\mathcal{A}}$'s are respectively operations and relations over $\mathcal{A}$ (of appropriate arities) and the $c_i^{\mathcal{A}}$'s are distinguished elements of A.

9.4.2. Convention.

- Typical notational conventions are to forget about the superscript ${}^{\mathcal{A}}$ in e.g., $f^{\mathcal{A}}$, and to identify $\mathcal{A}$ and A. (Otherwise we may write $A = |\mathcal{A}|$.)
- We think of relations in A as of functions ranging over the set $\{0, 1\}$ rather than of sets of tuples.

9.4.3. Definition.

1. Let t be a term with all free variables among $\vec{x}$, and let $\vec{a}$ be a vector of elements of A of the same length as $\vec{x}$. We define the value $t^{\mathcal{A}}(\vec{a}) \in A$ by induction:
   - $(x_i)^{\mathcal{A}}(\vec{a}) = a_i$;
   - $(ft_1 \dots t_n)^{\mathcal{A}}(\vec{a}) = f^{\mathcal{A}}(t_1^{\mathcal{A}}(\vec{a}), \dots, t_n^{\mathcal{A}}(\vec{a}))$.
2. Let φ be a formula such that all free variables of φ are among $\vec{x}$, and let $\vec{a}$ be as before. We define the value $\varphi^{\mathcal{A}}(\vec{a}) \in \{0, 1\}$, as follows
   - $\bot^{\mathcal{A}}(\vec{a}) = 0$;
   - $(rt_1 \dots t_n)^{\mathcal{A}}(\vec{a}) = r^{\mathcal{A}}(t_1^{\mathcal{A}}(\vec{a}), \dots, t_n^{\mathcal{A}}(\vec{a}))$;
   - $(\varphi \vee \psi)^{\mathcal{A}}(\vec{a}) = \varphi^{\mathcal{A}}(\vec{a}) \cup \psi^{\mathcal{A}}(\vec{a})$;
   - $(\varphi \wedge \psi)^{\mathcal{A}}(\vec{a}) = \varphi^{\mathcal{A}}(\vec{a}) \cap \psi^{\mathcal{A}}(\vec{a})$;
   - $(\varphi \to \psi)^{\mathcal{A}}(\vec{a}) = \varphi^{\mathcal{A}}(\vec{a}) \Rightarrow \psi^{\mathcal{A}}(\vec{a})$;
   - $(\forall y\, \varphi(y, \vec{x}))^{\mathcal{A}}(\vec{a}) = \inf\{\varphi(y, \vec{x})^{\mathcal{A}}(b, \vec{a}) : b \in A\}$;
   - $(\exists y\, \varphi(y, \vec{x}))^{\mathcal{A}}(\vec{a}) = \sup\{\varphi(y, \vec{x})^{\mathcal{A}}(b, \vec{a}) : b \in A\}$,

   where the operations $\cup$, $\cap$, $\Rightarrow$, inf and sup, and the constant 0 are in the two-element Boolean algebra of truth values. (Of course, we have $a \Rightarrow b = (1 - a) \cup b$.) We write $\mathcal{A}, \vec{a} \models \varphi(\vec{x})$ iff $\varphi^{\mathcal{A}}(\vec{a}) = 1$, and we write $\mathcal{A} \models \varphi(\vec{x})$ iff $\mathcal{A}, \vec{a} \models \varphi(\vec{x})$, for all $\vec{a}$. We write $\Gamma \models \varphi$ iff for all $\mathcal{A}$ and $\vec{a}$ with $\mathcal{A}, \vec{a} \models \psi$, for all $\psi \in \Gamma$, we also have $\mathcal{A}, \vec{a} \models \varphi$.

The above definition can now be generalized so that values of formulas are not necessarily in $\{0,1\}$, but in the algebra $P(X)$ of all subsets of a certain set X. Relations over $\mathcal{A}$ may now be seen as functions ranging over $P(X)$ rather than $\{0,1\}$, i.e., the notion of a structure is more general. One can go further and postulate values of formulas in an arbitrary Boolean algebra $\mathcal{B}$. This will work as well, provided $\mathcal{B}$ is a *complete* algebra, i.e., all infinite sup's and inf's do exist in $\mathcal{B}$. (Otherwise, values of some quantified formulas could not be defined.)

One can show that these generalizations do not change the class of classically valid statements of the form $\Gamma \models \varphi$ (Exercise 9.5.7). We do not investigate this further, since classical logic serves us as an illustration only.

Algebraic semantics

An obvious idea how to adopt the above approach to intuitionistic logic is to replace complete Boolean algebras by complete Heyting algebras.

9.4.4. DEFINITION.

1. An *intuitionistic $\mathcal{H}$-structure* for Σ is a system
$$\mathcal{A} = \langle A, f_1^{\mathcal{A}}, \dots, f_n^{\mathcal{A}}, r_1^{\mathcal{A}}, \dots, r_m^{\mathcal{A}}, c_1^{\mathcal{A}}, \dots, c_k^{\mathcal{A}}\rangle,$$
where the $f_i^{\mathcal{A}}$'s and $r_i^{\mathcal{A}}$'s and $c_i^{\mathcal{A}}$'s are as before, and the $r_i^{\mathcal{A}}$'s are functions of appropriate arity from A to a complete Heyting algebra $\mathcal{H}$.

2. The values of terms and formulas are defined as in Definition 9.4.3, except that operations $\cup$, $\cap$, inf, sup and $\Rightarrow$ are in $\mathcal{H}$.

3. The notation $\mathcal{A}, \vec{a} \models \varphi(\vec{x})$ and $\mathcal{A} \models \varphi(\vec{x})$ is as in Definition 9.4.3.

4. The notation $\Gamma \models \varphi$ should be understood as follows: "For all $\mathcal{H}$ and all $\mathcal{H}$-structures $\mathcal{A}$ and vectors $\vec{a}$, with $\mathcal{A}, \vec{a} \models \Gamma$, we also have $\mathcal{A}, \vec{a} \models \varphi$".

5. The symbol $\models_{\mathcal{K}}$ is $\models$ restricted to any given class $\mathcal{K}$ of complete Heyting algebras.

An example of a complete Heyting algebra is the algebra of open sets of a topological space (in particular a metric space), where sup is set-theoretic $\bigcup$, and
$$\inf\{A_i : i \in I\} = \mathrm{Int}(\bigcap\{A_i : i \in I\}).$$

9.4.5. THEOREM.

The following conditions are equivalent for the intuitionistic first-order logic:

1. $\Gamma \vdash \varphi$, where "$\vdash$" is either "$\vdash_N$" or "$\vdash_L$" or "$\vdash_H$";

2. $\Gamma \models \varphi$*;*

3. $\Gamma \models_{\mathcal{K}} \varphi$*, where* $\mathcal{K}$ *is the class of (algebras of open sets of) all metric spaces.*

It seems to be still an open problem whether the class of all metric spaces can be replaced by a one-element class consisting only of $\mathbb{R}^2$. But it can be shown that there exists a single metric space of this property, see [88]. (Note however that $\mathbb{R}^2$ can still be used for counterexamples.)

Kripke semantics

An alternative way of relaxing the definition of classical semantics is to keep the classical notion of a model, but think of models as of possible worlds.

9.4.6. DEFINITION. A structure $\mathcal{A} = \langle A, f_1^{\mathcal{A}}, \ldots, f_n^{\mathcal{A}}, r_1^{\mathcal{A}}, \ldots, r_m^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_k^{\mathcal{A}} \rangle$ is a *substructure* of $\mathcal{B} = \langle B, f_1^{\mathcal{B}}, \ldots, f_n^{\mathcal{B}}, r_1^{\mathcal{B}}, \ldots, r_m^{\mathcal{B}}, c_1^{\mathcal{B}}, \ldots, c_k^{\mathcal{B}} \rangle$ iff the following hold:

- $A \subseteq B$;
- $r_i^{\mathcal{A}} \subseteq r_i^{\mathcal{B}}$, for all i;
- $f_i^{\mathcal{A}} \subseteq f_i^{\mathcal{B}}$, for all i;
- $c_i^{\mathcal{A}} = c_i^{\mathcal{B}}$, for all i.

Thus, $\mathcal{B}$ extends the possible world $\mathcal{A}$ by enriching the domain of available objects and by adding more information about known objects. We write $\mathcal{A} \subseteq \mathcal{B}$ to express that $\mathcal{A}$ is a substructure of $\mathcal{B}$.

9.4.7. DEFINITION. A *Kripke model* for first-order logic is a triple of the form $\mathcal{C} = \langle C, \leq, \{\mathcal{A}_c : c \in C\} \rangle$, where C is a non-empty set, $\leq$ is a partial order in C, and $\mathcal{A}_c$'s are structures such that

$$\text{if } c \leq c' \text{ then } \mathcal{A}_c \subseteq \mathcal{A}_{c'}.$$

Let now φ be a formula such that all free variables of φ are among $\vec{x}$, and let $\vec{\mathsf{a}}$ be a vector of elements of A_c of the same length as $\vec{x}$. We define the relation $\Vdash$ by induction as follows:

- $c, \vec{\mathsf{a}} \Vdash rt_1 \ldots t_n$ iff $\mathcal{A}_c, \vec{\mathsf{a}} \models rt_1 \ldots t_n$ (classically);
- $c, \vec{\mathsf{a}} \nVdash \bot$;
- $c, \vec{\mathsf{a}} \Vdash \varphi \vee \psi$ iff $c, \vec{\mathsf{a}} \Vdash \varphi$ or $c, \vec{\mathsf{a}} \Vdash \psi$;
- $c, \vec{\mathsf{a}} \Vdash \varphi \wedge \psi$ iff $c, \vec{\mathsf{a}} \Vdash \varphi$ and $c, \vec{\mathsf{a}} \Vdash \psi$;

- $c, \vec{\mathsf{a}} \Vdash \varphi \to \psi$ iff $c', \vec{\mathsf{a}} \Vdash \psi$, for all c' such that $c \leq c'$ and $c', \vec{\mathsf{a}} \Vdash \varphi$;
- $c, \vec{\mathsf{a}} \Vdash \exists y\, \varphi(y, \vec{x})$ iff $c, \mathsf{b}, \vec{\mathsf{a}} \Vdash \varphi(y, \vec{x})$, for some $\mathsf{b} \in \mathcal{A}_c$;
- $c, \vec{\mathsf{a}} \Vdash \forall y\, \varphi(y, \vec{x})$ iff $c', \mathsf{b}, \vec{\mathsf{a}} \Vdash \varphi(y, \vec{x})$, for all c' such that $c \leq c'$ and all $\mathsf{b} \in \mathcal{A}_{c'}$.

The symbol $\Vdash$ is now used in various contexts as usual, in particular $\Gamma \Vdash \varphi$ means that $c, \vec{\mathsf{a}} \Vdash \varphi$ whenever $c, \vec{\mathsf{a}} \Vdash \Gamma$.

9.4.8. THEOREM. *The conditions $\Gamma \Vdash \varphi$ and $\Gamma \models \varphi$ are equivalent.*

More about semantics can be found in [88, 106, 107, 108].

9.5. Exercises

9.5.1. EXERCISE. Find constructions for formulas (1), (3), (5), (7), (9) and (11) of Example 9.2.1, and do not find constructions for the other formulas.

9.5.2. EXERCISE. A first-order formula is in *prenex normal form* iff it begins with a sequence of quantifiers followed by an open formula. Consider a signature with no function symbols, and let φ be an intuitionistic theorem in a prenex normal form. Show that there exists an open formula φ' obtained from φ by removing quantifiers and by replacing some variables by constants, and such that $\vdash \varphi'$. *Hint:* Use the existence property (Corollary 9.3.2).

9.5.3. EXERCISE (V.P. Orevkov). Apply Exercise 9.5.2 to show that the prenex fragment of intuitionistic first-order logic over function-free signatures is decidable. (In fact, this remains true even with function symbols in the signature, but fails for logics with equality, see [29]).

9.5.4. EXERCISE. Prove that every first-order formula is classically equivalent to a formula in prenex normal form. Then prove that intuitionistic first-order logic does not have this property.

9.5.5. EXERCISE. Show that the existence property (Corollary 9.3.2) does not hold for classical logic. Why does the proof break down in this case?

9.5.6. EXERCISE. Prove Theorem 9.3.7.

9.5.7. EXERCISE. Let X be an arbitrary set with more than one element. Show that the semantics of classical logic where values of formulas are taken in the family $P(X)$ of all subsets of X is equivalent to the ordinary semantics. That is, the sets of tautologies are the same.

9.5.8. EXERCISE. Verify that the odd-numbered formulas of Example 9.2.1 are intuitionistically valid, while the even-numbered ones are not.

9.5.9. Exercise. Show that the following classical first-order tautologies are not valid intuitionistically:

- $\exists x(\varphi(x) \to \forall x\, \varphi(x))$;
- $\exists x(\varphi(0) \vee \varphi(1) \to \varphi(x))$;
- $\forall x\, \neg\neg\varphi(x). \leftrightarrow \neg\neg\forall x\, \varphi(x)$;
- $\exists x\, \neg\neg\varphi(x). \leftrightarrow \neg\neg\exists x\, \varphi(x)$.

9.5.10. Exercise. A Kripke model $\mathcal{C} = \langle C, \leq, \{\mathcal{A}_c : c \in C\}\rangle$ has *constant domains* iff all the $\mathcal{A}_c$ are the same. Prove that the formula $\forall x(\psi \vee \varphi(x)) \leftrightarrow \psi \vee \forall x\, \varphi(x)$, where $x \notin FV(\psi)$ (formula (6) of Example 9.2.1) is valid in all models with constant domains.

9.5.11. Exercise. Prove that the formula $\neg\neg\forall x(\varphi \vee \neg\varphi)$ (formula (10) of Example 9.2.1) is valid in all Kripke models with finite sets of states.

CHAPTER 10

Dependent types

Dependent types can probably be considered as old as the whole idea of *propositions-as-types*. Explicitly, dependent types were perhaps first used in various systems aimed at constructing and verifying formal proofs. One of the first was the project AUTOMATH of de Bruijn, see [28]. Another such system that gained much attention is the Edinburgh Logical Framework (LF) of Harper, Honsell and Plotkin [51]. The expression "logical frameworks" is now used as a generic name for various similar calculi, see [59]. Last but not least, one should mention here Martin-Löf's type theory [77]. For more references, see [108, 51, 8]. Our presentation of dependent types follows essentially that of [8].

From a programmer's point of view, a dependent type is one that *depends* on an object value. For instance, one may need to introduce a type $\mathit{string}(n)$ of all binary strings of length n.[1] This type depends on a choice of $n : \mathit{int}$. The operator *string* makes a type from an integer, and corresponds, under the Curry-Howard isomorphism, to a predicate over *int*. Such a predicate is called a *type constructor*, or simply *constructor*. Of course, we have to classify constructors according to their domains, and this leads to the notion of a *kind*: we say that our constructor *string* is of kind $\mathit{int} \Rightarrow *$, where $*$ is the kind of all types. Of course, there is no reason to disallow binary predicates, and so on, and thus the family of kinds should include $\tau_1 \Rightarrow \cdots \tau_n \Rightarrow *$.

A definition of an object of type $\mathit{string}(n)$ may happen to be uniform in n, i.e., we may have a generic procedure *Onlyzeros* that turns any $n : \mathit{int}$ into a string of zeros of length n. The type of such a procedure should be written as $(\forall x{:}\mathit{int})\mathit{string}(x)$.

In general, a type of the form $(\forall x{:}\tau)\sigma$ is a type of a function applicable to objects of type τ and returning an object of type $\sigma[x := \mathbf{a}]$, for each argument $\mathbf{a} : \tau$. It is not difficult to see that this idea is more general than the idea of a function type. Indeed, if x is not free in σ, then $(\forall x{:}\tau)\sigma$ behaves

[1] Example from [74].

exactly as $\tau \to \sigma$. Thus, in presence of dependent types there is no need to introduce $\to$ separately.

The set-theoretic counterpart of a dependent type is the *product*. Recall that if $\{A_t\}_{t\in T}$ is an indexed family of sets (formally a function that assigns the set A_t to any $t \in T$)[2] then the product of this family is the set:

$$\prod_{t\in T} A_t = \{f \in (\bigcup_{t\in T} A_t)^T : f(t) \in A_t, \text{ for all } t \in T\}.$$

For $f \in \prod_{t\in T} A_t$, the value of $f(t)$ is in a set A_t, perhaps different for each argument, rather than in a single co-domain A. If all A_t are equal to a fixed set A, we obtain the equality

$$\prod_{t\in T} A_t = A^T,$$

corresponding to our previous observation about $\to$ versus $\forall$.

The logical counterpart of this should now be as follows: the implication is a special case of universal quantification. And that is correct, because we have already agreed to identify objects of type τ with proofs of τ. We only have to agree that this way of thinking applies to individual objects as well, so that, for instance, an integer can be seen as a *proof* of *int*.

10.1. System λP

We will now define the Church-style system λP of dependent types. We begin with a calculus without existential quantification, as in [8]. Unfortunately, even without existential quantifiers, the language of λP is a broad extension of the language of simply-typed lambda-calculus. We have three sorts of expressions: object expressions (ranged over by M, N, etc.), constructors (ranged over by τ, φ, etc.) and kinds (ranged over by $\boldsymbol{\kappa}, \boldsymbol{\kappa}'$, etc.). There are object and constructor variables (ranged over by x, y, ... and α, β, ... , respectively), and one kind constant $*$. A type is treated as a special case of a constructor, so we do not need extra syntax for types.

Contexts can no longer be arbitrary sets of assumptions. This is because in order to declare a variable of type e.g., αx, one has to know *before* that the application is legal, i.e., that the type of x fits the kind of α. Thus, contexts in λP are defined as sequences of assumptions. In addition, not every sequence of declarations can be regarded as a valid context, and being valid or not depends on derivability of certain judgements.

For similar reasons, being a constructor or a kind, also depends on derivable judgements. Well-formed types, kinds and contexts are thus formally defined by the rules of our system.

[2]That is why we do *not* use here the set notation $\{A_t : t \in T\}$.

Unfortunately, we cannot stick to $\Rightarrow$ as the only way to build kinds, and we have to introduce a more general product operator also at the level of kinds. To understand why, see Example 10.2.3(2).

10.1.1. DEFINITION.

1. *Raw expressions* (raw contexts Γ, raw kinds $\boldsymbol{\kappa}$, raw constructors ϕ and raw lambda-terms M) are defined by the following grammar:

   $\Gamma ::= \{\} \mid \Gamma, (x : \phi) \mid \Gamma, (\alpha : \boldsymbol{\kappa})$;
   $\boldsymbol{\kappa} ::= * \mid (\Pi x : \phi)\boldsymbol{\kappa}$;
   $\phi ::= \alpha \mid (\forall x{:}\phi)\phi \mid (\phi M)$;
   $M ::= x \mid (MM) \mid (\lambda x{:}\phi.M)$.

2. Beta reduction on raw terms is defined as follows:
   - $(\lambda x{:}\tau.M)N \to_\beta M[N := x]$;
   - If $\tau \to_\beta \tau'$ then $\lambda x{:}\tau.M \to_\beta \lambda x{:}\tau'.M$;
   - If $M \to_\beta M'$ then $\lambda x{:}\tau.M \to_\beta \lambda x{:}\tau.M'$ and $NM \to_\beta NM'$ and $MN \to_\beta M'N$;
   - If $\tau \to_\beta \tau'$ then $(\Pi x : \tau)\boldsymbol{\kappa} \to_\beta (\Pi x : \tau')\boldsymbol{\kappa}$;
   - If $\boldsymbol{\kappa} \to_\beta \boldsymbol{\kappa}'$ then $(\Pi x : \tau)\boldsymbol{\kappa} \to_\beta (\Pi x : \tau)\boldsymbol{\kappa}'$;
   - If $\tau \to_\beta \tau'$ then $(\forall x{:}\tau)\sigma \to_\beta (\forall x{:}\tau')\sigma$;
   - If $\sigma \to_\beta \sigma'$ then $(\forall x{:}\tau)\sigma \to_\beta (\forall x{:}\tau)\sigma'$;
   - If $\varphi \to_\beta \varphi'$ then $\varphi M \to_\beta \varphi' M$;
   - If $M \to_\beta M'$ then $\varphi M \to_\beta \varphi M'$.

3. If $(x : \tau)$ or $(\alpha : \boldsymbol{\kappa})$ is in Γ then we write $\Gamma(x) = \tau$ or $\Gamma(\alpha) = \boldsymbol{\kappa}$, respectively. We also write $\mathrm{Dom}(\Gamma)$ for the set of all constructors and object variables declared in Γ.

4. We skip the obvious definition of free variables. Of course, there are three binding operators now: lambda-abstraction, quantification and the product of kinds. And we also omit the definition of substitution.

5. We use arrows as abbreviations: if x is not free in $\boldsymbol{\kappa}$ then we write $\tau \Rightarrow \boldsymbol{\kappa}$ instead of $(\Pi x : \tau)\boldsymbol{\kappa}$. And if x is not free in σ then $(\forall x{:}\tau)\sigma$ is abbreviated by our good old implication $\tau \to \sigma$.

10.1.2. REMARK. In order to spare the reader some additional noise, we choose a non-standard presentation of λP, namely we do not allow lambda abstractions in constructors. Thus, every constructor must be of the form $(\forall x{:}\tau_1) \cdots (\forall x{:}\tau_n)\alpha M_1 \ldots M_n$. This restriction is not essential as long as our primary interest is in *types*. Indeed, a dependent type in normal form always obeys this pattern, and a term substitution may never create a constructor redex.

We will have three different sorts of judgements in our system:

- kind formation judgements of the form "$\Gamma \vdash \boldsymbol{\kappa} : \Box$";
- kinding judgements of the form "$\Gamma \vdash \varphi : \boldsymbol{\kappa}$";
- typing judgements of the form "$\Gamma \vdash M : \tau$".

The meaning of "$\boldsymbol{\kappa} : \Box$" is just that $\boldsymbol{\kappa}$ is a well-formed kind, and $\Box$ itself is not a part of the language.

10.2. Rules of λP

Kind formation rules:

$$\vdash * : \Box \qquad\qquad \frac{\Gamma, x : \tau \vdash \boldsymbol{\kappa} : \Box}{\Gamma \vdash (\Pi x{:}\tau)\boldsymbol{\kappa} : \Box}$$

Kinding rules:

$$\frac{\Gamma \vdash \boldsymbol{\kappa} : \Box}{\Gamma, \alpha : \boldsymbol{\kappa} \vdash \alpha : \boldsymbol{\kappa}}\ (\alpha \notin \mathrm{Dom}(\Gamma))$$

$$\frac{\Gamma \vdash \varphi : (\Pi x{:}\tau)\boldsymbol{\kappa} \quad \Gamma \vdash M : \tau}{\Gamma \vdash \varphi M : \boldsymbol{\kappa}[x := M]} \qquad\qquad \frac{\Gamma, x : \tau \vdash \sigma : *}{\Gamma \vdash (\forall x{:}\tau)\sigma : *}$$

Typing rules:

$$\frac{\Gamma \vdash \tau : *}{\Gamma, x : \tau \vdash x : \tau}\ (x \notin \mathrm{Dom}(\Gamma))$$

$$\frac{\Gamma \vdash N : (\forall x{:}\tau)\sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash NM : \sigma[x := M]} \qquad\qquad \frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\tau.M : (\forall x{:}\tau)\sigma}$$

Note that there is no restriction "$x \notin \mathrm{FV}(\Gamma)$" attached to the $\forall$-introduction rule. This restriction is unnecessary because otherwise $\Gamma, x : \tau$ would not be a valid context, in which case the premise could not be derived.

Weakening rules:

There are six weakening rules, but all obey the same pattern: an additional assumption does not hurt, as long as it is well-formed. We need explicit wekening (rather than relaxed axioms) because of the sequential structure of contexts we must respect.

$$\frac{\Gamma \vdash \tau : * \quad \Gamma \vdash \boldsymbol{\kappa} : \Box}{\Gamma, x : \tau \vdash \boldsymbol{\kappa} : \Box}\ (x \notin \mathrm{Dom}(\Gamma)) \quad \frac{\Gamma \vdash \boldsymbol{\kappa} : \Box \quad \Gamma \vdash \boldsymbol{\kappa'} : \Box}{\Gamma, \alpha : \boldsymbol{\kappa} \vdash \boldsymbol{\kappa'} : \Box}\ (\alpha \notin \mathrm{Dom}(\Gamma))$$

$$\frac{\Gamma \vdash \tau : * \quad \Gamma \vdash \varphi : \boldsymbol{\kappa}}{\Gamma, x : \tau \vdash \varphi : \boldsymbol{\kappa}}\ (x \notin \mathrm{Dom}(\Gamma)) \qquad \frac{\Gamma \vdash \boldsymbol{\kappa} : \Box \quad \Gamma \vdash \varphi : \boldsymbol{\kappa'}}{\Gamma, \alpha : \boldsymbol{\kappa} \vdash \varphi : \boldsymbol{\kappa'}}\ (\alpha \notin \mathrm{Dom}(\Gamma))$$

$$\frac{\Gamma \vdash \tau : * \quad \Gamma \vdash M : \sigma}{\Gamma, x : \tau \vdash M : \sigma}\ (x \notin \mathrm{Dom}(\Gamma)) \qquad \frac{\Gamma \vdash \boldsymbol{\kappa} : \Box \quad \Gamma \vdash M : \sigma}{\Gamma, \alpha : \boldsymbol{\kappa} \vdash M : \sigma}\ (\alpha \notin \mathrm{Dom}(\Gamma))$$

Conversion rules

These rules are necessary because of the terms occurring in types which do not have to be in normal forms.

$$\frac{\Gamma \vdash \varphi : \boldsymbol{\kappa} \quad \boldsymbol{\kappa} =_\beta \boldsymbol{\kappa'}}{\Gamma \vdash \varphi : \boldsymbol{\kappa'}} \qquad \frac{\Gamma \vdash M : \sigma \quad \sigma =_\beta \sigma'}{\Gamma \vdash M : \sigma'}$$

10.2.1. Remark. Note that type-checking in λP is not a trivial task, even if our language is Church-style. This is because the conversion rules. A verification whether two types or kinds are equal may be as difficult as checking equality of simply-typed lambda terms, and this decision problem is non-elementary (Theorem 6.7.1).

10.2.2. Definition.
If $\Gamma \vdash \tau : *$ then we say that τ is a *type* in the context Γ.

10.2.3. Example.

1. The lambda term $\lambda y{:}(\forall x{:}0.\alpha x \to \beta x).\lambda z{:}(\forall x{:}0.\alpha x).\lambda x{:}0.yx(zx)$ has type $(\forall x{:}0.\alpha x \to \beta x) \to (\forall x{:}0.\alpha x) \to \forall x{:}0.\beta x$ in the context consisting of declarations $\alpha : 0 \Rightarrow *$ and $\beta : 0 \Rightarrow *$.
2. Here is an example of a kind that cannot be expressed only with $\Rightarrow$:

$$\alpha : 0 \Rightarrow * \vdash (\Pi y{:}0)(\alpha y \Rightarrow *) : \Box$$

   If we now declare a variable φ to be of this kind, we can consider types of the form φyx, where x is of type αy.

10.3. Properties of λP

System λP has the Church-Rosser and strong normalization properties. These facts follow from properties of simply typed terms, thanks to an embedding $M \mapsto M^\bullet$, described in [51]. Certain variants of this embedding (esp. the "forgetting map" $M \mapsto \overline{M}$, which erases all dependencies) were also discovered by V. Breazu-Tannen and Ch. Paulin, but apparently R.O. Gandy was the first to use this approach (see [108, p. 565]).

10.3.1. Definition. We define the dependency-erasure map on constructors:

$\overline{\alpha} = \alpha$;

$\overline{(\forall x : \tau)\sigma} = \overline{\tau} \to \overline{\sigma}$;

$\overline{\varphi M} = \overline{\varphi}$.

We write $\overline{\Gamma}$ for the contexts obtained by applying the above operation to all right-hand sides of declarations in Γ of the form $(x : \tau)$, and removing other declarations.

10.3.2. Definition. The translations $M \mapsto M^\bullet$ and $\tau \mapsto \tau^\bullet$ are defined for both types and terms of λP. Both terms and types are translated into terms. Below, 0 stands for a fixed type variable. Abusing the formalism a little, we assume that we can use as many fresh term variables as we want.

$(\alpha M_1 \dots M_n)^\bullet = x_\alpha M_1^\bullet \dots M_n^\bullet$, where x_α is a fresh variable;

$((\forall x{:}\tau)\sigma)^\bullet = x_\tau \tau^\bullet (\lambda x{:}\overline{\tau}.\sigma^\bullet)$, where x_τ is a fresh variable;

$x^\bullet = x$;

$(MN)^\bullet = M^\bullet N^\bullet$;

$(\lambda x{:}\tau.M)^\bullet = (\lambda y{:}0 \lambda x{:}\overline{\tau}.M^\bullet)\tau^\bullet$, where y is a fresh variable.

10.3.3. Lemma.

1. *If* $\Gamma \vdash \tau : *$ *then* $\overline{\Gamma'} \vdash \tau^\bullet : 0$, *for some extension* Γ' *of* Γ.

2. *If* $\Gamma \vdash M : \tau$ *then* $\overline{\Gamma'} \vdash M^\bullet : \overline{\tau}$, *for some extension* Γ' *of* Γ.

3. *If* $\Gamma \vdash M : \tau$ *and* $M \to_\beta M_1$ *then* $M^\bullet \twoheadrightarrow_\beta M_1^\bullet$ *in at least one step.*

Proof. Exercise 10.7.2. □

10.3.4. Corollary (Strong normalization).
The system λP *has the strong normalization property.*

Proof. By part (3) of Lemma 10.3.3, an infinite reduction starting from M would be translated to an infinite reduction in the simply-typed lambda calculus. □

10.3.5. Corollary. *The system* λP *has the Church-Rosser property.*

Proof. Exercise 10.7.5. □

10.4. Dependent types à la Curry

10.4.1. Definition. We define a type-erasure mapping $|\cdot|$ from terms of λP to pure lambda terms, as usual:

- $|x| = x$;
- $|\lambda x{:}\tau.M| = \lambda x.|M|$;
- $|MN| = |M||N|$.

For a a pure lambda term N, we write $\Gamma \vdash_P N : \tau$ iff $N = |M|$, for some Church-style term M with $\Gamma \vdash M : \tau$. We say that a pure lambda term N is *typable* iff $\Gamma \vdash_P N : \tau$ holds for some Γ, τ.

An alternative to the above definition is to define a type assignment system for pure lambda-terms, corresponding to λP. Note that in this case we must deal with different notion of a type, since types must depend on pure lambda-terms rather than on Church-style terms. It follows that the type-erasure mapping must be extended to types and kinds. Fortunately, the notion of a typable term is the same with this approach as with our simple definition, see [5].

10.4.2. Proposition. *The Curry-style variant of* λP *has the subject-reduction property, that is if* $\Gamma \vdash_P N : \tau$ *and* $N \twoheadrightarrow_\beta N'$ *then also* $\Gamma \vdash_P N' : \tau$.

Proof. Boring. See [5]. □

Let us come back to the dependency-erasure map. We extend it to terms as follows:

10.4.3. Definition.
$\overline{x} = x$;
$\overline{MN} = \overline{M}\,\overline{N}$;
$\overline{\lambda x{:}\tau.M} = \lambda x{:}\overline{\tau}.\overline{M}$;

We have the following result:

10.4.4. Lemma. *If* $\Gamma \vdash M : \tau$ *then* $\overline{\Gamma} \vdash \overline{M} : \overline{\tau}$ *(in* $\lambda\!\to$*).*

Proof. Exercise 10.7.3. □

10.4.5. Proposition. *A term is typable in* λP *iff it is simply-typable. In particular, the type reconstruction problem for* λP *is decidable in polynomial time.*

PROOF. Suppose that $\Gamma \vdash M : \tau$ in λP. By Lemma 10.4.4, we have $\overline{\Gamma} \vdash \overline{M} : \overline{\tau}$, in the simply typed lambda calculus. But it is easy to see that $|\overline{M}| = |M|$.

We have just shown that all pure terms typable in λP are typable in simple types. The converse is obvious, and thus type reconstruction in λP is the same as in simple types. □

Here comes the surprise: it is not at all that easy with type checking!

10.4.6. THEOREM (G. Dowek [31]). *Type checking in the Curry-style version of* λP *is undecidable.*

NO PROOF. We regret that we do not have enough time to present this nice proof. But we recommend reading the original paper [31]. □

10.5. Existential quantification

This section is a diggression. The system λP, as most other typed lambda-calculi, is normally studied with $\forall$ and $\to$ as the only basic connectives. Of course, an extension of this system with $\vee$, $\wedge$ and $\bot$ can be defined in much the same way as for the simply-typed lambda calculus. It remains to see what is the lambda calculus counterpart of existential quantification. The intuition suggests that $(\exists x{:}\tau)\,\varphi(x)$ should be understood as a *disjoint union* or *coproduct* of types $\varphi(\mathbf{a})$, for all objects $\mathbf{a}$ of type τ. That is, objects of type $(\exists x{:}\tau)\,\varphi(x)$ are pairs consisting of an object $\mathbf{a}$ of type τ and a proof M of $\varphi(\mathbf{a})$. This may be syntactically written as: **pack** $M, \mathbf{a}$ **to** $(\exists x{:}\tau)\,\varphi(x)$.

An elimination operator for type $(\exists x{:}\tau)\,\varphi(x)$ takes such a pair and uses it whenever an object of type $\varphi(\mathbf{x})$ can be used with an unspecified $\mathbf{x}$. This leads to the following deduction rules:

$$(\exists\mathrm{I})\frac{\Gamma \vdash M : \varphi[x := N] \quad \Gamma \vdash N : \tau}{\Gamma \vdash \mathbf{pack}\ M, N\ \mathbf{to}\ (\exists x{:}\tau)\,\varphi : (\exists x{:}\tau)\,\varphi}$$

$$(\exists\mathrm{E})\frac{\Gamma \vdash M : (\exists x{:}\tau)\,\varphi \quad \Gamma, x : \tau, z : \varphi \vdash N : \psi}{\Gamma \vdash \mathbf{let}\ z : \varphi\ \mathbf{be}\ M : (\exists x{:}\tau)\,\varphi\ \mathbf{in}\ N : \psi}\ (x \notin FV(\Gamma, \psi))$$

Note that although ψ in $(\exists\mathrm{E})$ cannot contain x free, this restriction does not extend to the proof N. The variable x may thus be free in N, but it should be considered bound in the **let** expression.

The reduction rule for sum types is as follows:

$$\mathbf{let}\ z : \varphi\ \mathbf{be}\ (\mathbf{pack}\ M, N\ \mathbf{to}\ (\exists x{:}\tau)\,\varphi)\ \mathbf{in}\ Q \longrightarrow_\beta Q[x := N][z := M]$$

Everything is fine with this rule, as long as we do not get seduced by the temptation of making a Curry-style calculus with existential quantifiers. Look, how beautiful rules we may have:

$$(\exists\mathrm{I})\frac{\Gamma \vdash M : \varphi[x := N] \quad \Gamma \vdash N : \tau}{\Gamma \vdash M : (\exists x{:}\tau)\,\varphi}$$

$$(\exists\mathrm{E})\frac{\Gamma \vdash M : (\exists x{:}\tau)\,\varphi \quad \Gamma, z : \varphi \vdash N : \psi}{\Gamma \vdash N[z := M] : \psi}\ (x \notin FV(\Gamma, \psi))$$

These rules are based on the idea of *existential polymorphism*: a term of an existential type is like an object with some data being abstracted or encapsulated or "private", and not available for external manipulations. It seems very appealing that this sort of abstraction might be done with only implicit typing discipline.

There are however some annoying little difficulties when one attempts to prove the subject reduction property for a type inference system $\vdash_\exists$ with rules as above. Some of these difficulties can be eliminated by improving the rules, but some cannot. The following example is based on an idea used in [6] for union types (which cause similar problems).

10.5.1. EXAMPLE. Consider a context Γ with the following declarations:

$$\begin{array}{l} 0 : *, \quad \alpha : *, \quad \beta : *, \quad \sigma : 0 \Rightarrow *, \quad x : 0, \\ X : \sigma x \to \sigma x \to \alpha, \quad Y : \beta \to (\exists x{:}0)\sigma x, \quad Z : \beta. \end{array}$$

Then $\Gamma \vdash_\exists X(\mathbf{I}YZ)(\mathbf{I}YZ) : \alpha$, but $\Gamma \not\vdash_\exists X(\mathbf{I}YZ)(YZ) : \alpha$.

10.6. Correspondence with first-order logic

The system λP of dependent types is much stronger than is needed to provide a term assignment for first-order intuitionistic logic. In fact, first-order logic corresponds to a fairly weak fragment of λP. This fragment is obtained by restricting the syntax[3] so that each context must satisfy the following:

- There is only one type variable 0 (which should be regarded as a type constant), representing the type of individuals;
- All kinds are of the form $0 \Rightarrow \cdots \Rightarrow 0 \Rightarrow *$;
- There is a finite number of distinguished constructor variables, representing relation symbols in the signature (they must be of appropriate kinds, depending on arity);
- Function symbols in the signature are represented by distinguished object variables of types $0 \to \cdots \to 0 \to 0$, depending on arity;
- Constant symbols are represented by distinguished object variables of type 0;

[3]Of course one has to either add $\exists$ to λP, as in the previous section, or to consider only universal quantification in formulas.

- Other declarations in the context may only be of the form $(x : 0)$, and correspond to individual variables.

Clearly, algebraic terms are represented now by lambda terms of type 0 in normal form. Formulas are represented by types. Because of strong normalization, we can restrict attention to formulas containing only terms in normal form. It follows that an inhabited formula is a theorem and conversely. The reader is invited to formally define the syntax of the first-order fragment (Exercise 10.7.9).

In summary, we have the following pairs of corresponding items:

formulas	$\sim$	types
proofs	$\sim$	terms
domain of individuals	$\sim$	type constant 0
algebraic terms	$\sim$	terms of type 0
relations	$\sim$	constructors of kind $0 \Rightarrow 0 \Rightarrow \cdots \Rightarrow 0 \Rightarrow *$
atomic formula $r(t_1, \dots, t_n)$	$\sim$	dependent type $rt_1 \dots t_n : *$
universal formula	$\sim$	product type
proof by generalization	$\sim$	abstraction $\lambda x : 0.M^{\varphi}$
proof by *modus ponens*	$\sim$	application $M^{\forall x:0\,\varphi}N^0$

10.6.1. Example. Consider the Hilbert-style axioms of Section 9.3 Each of them corresponds to a type in λP, extended by existential quantification as in Section 10.5. These types are inhabited as follows. (We write types as upper indices to improve readability.)

- $\lambda Y^{(\forall x:0)\varphi(x)}.Yt : (\forall x:0)\varphi(x). \to \varphi(t)$;
- $\lambda X^{\varphi(t)}.\mathbf{pack}\ X, t\ \mathbf{to}\ (\exists x:0)\varphi(x) : \varphi(t) \to (\exists x:0)\varphi(x)$;
- $\lambda Y^{\psi}\lambda x^0.Y : \psi \to (\forall x : 0)\,\psi$, where $x \notin FV(\psi)$;
- $\lambda Y^{(\exists x:0)\psi}.\mathbf{let}\ z{:}\psi\ \mathbf{be}\ Y{:}(\exists x:0)\psi\ \mathbf{in}\ z : (\exists x:0)\psi. \to \psi$, where $x \notin FV(\psi)$;
- $\lambda X^{(\forall x:0)(\varphi\to\psi)}\lambda Y^{(\exists x:0)\varphi}.\mathbf{let}\ z^{\varphi}\ \mathbf{be}\ Y{:}(\exists x:0)\varphi\ \mathbf{in}\ \mathbf{pack}\ Xxz, x\ \mathbf{to}\ (\exists x:0)\varphi : (\forall x:0)(\varphi \to \psi). \to ((\exists x:0)\varphi. \to (\exists x:0)\psi)$;
- $\lambda X^{(\forall x:0)(\varphi\to\psi)}\lambda Y^{(\forall x:0)\varphi}\lambda z^0.Xz(Yz) : (\forall x:0)(\varphi \to \psi). \to ((\forall x:0)\varphi. \to (\forall x:0)\psi)$.

Out of the above six examples three do not use existential quantification. If we apply the dependency-erasing translation $M \mapsto \overline{M}$ to the third and sixth of the above terms, we obtain the typed combinators

- $\mathbf{K} : \overline{\psi} \to 0 \to \overline{\psi}$ and

- $\mathbf{S} : (0 \to \overline{\varphi} \to \overline{\psi}) \to (0 \to \overline{\varphi}) \to 0 \to \overline{\psi}$.

This may be a suprise on the first look that we obtain something so much familiar. But it is fully justified by the fact that universal quantification is a generalization of implication.

The first example is a little less spectacular, as it erases to $\lambda Y^{0 \to \overline{\varphi}}.Yt : (0 \to \overline{\varphi}) \to \overline{\varphi}$ (parameterized by $t : 0$).

The above embedding of first-order logic into λP can easily be generalized to various extensions. For instance, many-sorted logic is handled just by allowing more than one atomic type. Here are some examples of other features that are not present in first-order logic:

- There are sorts (types) for many domains for individuals, and also for functions on the individuals;
- There is a function abstraction mechanism (we deal with lambda-terms rather than algebraic terms);
- Quantification over functions is permitted; a quantifier ranges over arbitrary expressions of a given sort;
- Proofs are terms, so that properties of proofs can be expressed by formulas.

The presence of quantification over objects of all finite types means that λP can express many higher-order languages (see [51]). We would however prefer not to use the expression "higher-order logic" here, as there is no quantification over propositions. A more adequate statement about λP is that it provides a many-sorted first-order representation of higher-order logic.

Another issue is expressibility of first-order *theories* in λP. You can find more about this in [8, p. 202].

As we have already observed, first-order intuitionistic logic is undecidable. Thus type inhabitation in the corresponding fragment of λP must be also undecidable. But λP can be shown conservative over this fragment, and it follows that type inhabitation for the whole system is also undecidable. Another direct and simple proof can be found in [13].

10.6.2. THEOREM. *Type inhabitation in* λP *is undecidable.* □

10.7. Exercises

10.7.1. EXERCISE. Let *string*(n) be the type of binary strings of length n. Every such string w determines a record type with integer and boolean fields corresponding to the digits in w. For example, for 01101 we take

$int \wedge bool \wedge bool \wedge int \wedge bool$. Define an appropriate context Γ declaring variables $string$ and $record$ so that $record(n)(w)$ is a well-formed type, for $n : int$ and $w : string(n)$.

10.7.2. Exercise. Prove Lemma 10.3.3. (First extend the translation to kinds and define Γ'. A substitution lemma will also be necessary.)

10.7.3. Exercise. Prove Lemma 10.4.4. Also prove that $\Gamma \vdash M : \tau$ and $M \to_\beta M'$ implies $\overline{M} \to_\beta \overline{M}'$ or $M = M'$.

10.7.4. Exercise. Consider the following attempt to prove strong normalization for λP:

Assume first that a term M is such that no type occurring in M contains a lambda term. In this case an infinite reduction starting from M would be translated, by Lemma 10.4.4, to an infinite reduction in the simply-typed lambda calculus. The general case follows from the following induction step: If all subterms of M (including all terms occurring in types in M) are strongly normalizing then M is strongly normalizing. To prove the induction step, we observe that an infinite reduction starting from M translates to a sequence of pure terms N_i, with $N_i = N_j$, for all $i, j > i_0$. Thus our infinite reduction must, from some point on, consist exclusively of reduction steps performed exclusively within types. But all terms occurring in types are obtained from subterms of the original term M and thus must strongly normalize by the induction hypothesis.

Find the bug in this proof. Can you fix it?[4]

10.7.5. Exercise. Prove the Church-Rosser property for λP (Corollary 10.3.5). Warning: do first Exercise 10.7.6. *Hint:* Apply Newman's Lemma.

10.7.6. Exercise. Show an example of λP terms M and M' of the same type such that $\overline{M} = \overline{M}'$, but $M \neq_\beta M'$.

10.7.7. Exercise. The proof technique that fails for Church-style λP (Exercise 10.7.4) works for the Curry-style terms. Prove strong normalization for Curry-style λP with help of the dependency-erasing translation $M \mapsto \overline{M}$.

10.7.8. Exercise. Verify the correctness of Example 10.5.1.

10.7.9. Exercise. Define formally the fragment of λP corresponding to first-order logic over a fixed signature.

10.7.10. Exercise. Consider the odd-numbered formulas of Example 9.2.1, as types in an appropriate extension of λP. Write lambda-terms inhabiting these types. (If necessary use operators related to conjunction, disjunction, and the existential quantifier.)

[4] We do not know how.

10.7.11. EXERCISE. We know that types of **K** and **S** are sufficient to axiomatize the arrow-only fragment of propositional intuitionistic logic. These types correspond to the third and sixth axiom of Section 9.3 (see Example 10.6.1). One can thus conjecture that the first axiom $\forall x\, \varphi(x). \to \varphi(t)$, which becomes $(0 \to \overline{\varphi}) \to \overline{\varphi}$ after erasing dependencies, can be eliminated from the axiom system. Show that this conjecture is wrong: our formula cannot be derived from the other two axiom schemes.

CHAPTER 11

First-order arithmetic and Gödel's T

Arithmetic is the core of almost all of mathematics. And expressing and proving properties of integers always was one of the primary goals of mathematical logic. In this chapter, we will trace the Curry-Howard correspondence back to the 40's and 50's, to discover it in some fundamental works of Kleene and Gödel. Both these works aimed at proving consistency of Peano Arithmetic. For this purpose they give some fundamental insights on the constructive contents of arithmetical proofs.

11.1. The language of arithmetic

The signature (cf. Definition 6.6) of first-order arithmetic consists of two binary function symbols $+$ and $\cdot$, two constant symbols 0 and 1 and the symbol $=$ for equality.[1] The *standard model* of arithmetic is the set of integers $\mathbb{N}$ with the ordinary understanding of these symbols, i.e., the structure:

$$\mathcal{N} = \langle \mathbb{N}, +, \cdot, 0, 1, = \rangle.$$

Note that all elements of $\mathbb{N}$ can be given names in the language of arithmetic. Let $\underline{n}$ denote the term $1 + 1 + \cdots + 1$, with exactly n copies of 1 (assuming that $\underline{0}$ is 0).

By $\mathrm{Th}(\mathcal{A})$ we denote the set of all first-order sentences that are classically true in $\mathcal{A}$ (i.e., the set of all sentences φ such that $\mathcal{A} \models \varphi$, in the sense of Section 9.4). The following classical result shows the limitation of first-order logic:

11.1.1. THEOREM. *There exists a nonstandard model of arithmetic, i.e., a structure* $\mathcal{M} = \langle \mathbb{M}, \oplus, \otimes, \mathbf{0}, \mathbf{1}, = \rangle$, *such that* $\mathrm{Th}(\mathcal{M}) = \mathrm{Th}(\mathcal{N})$, *but* $\mathcal{M}$ *and* $\mathcal{N}$ *are not isomorphic.*

[1] Another typical choice is to take a unary function symbol s for the successor function, instead of 1.

The above fact is a consequence of *compactness theorem*, see [71].

However, the definitional strength of first-order formulas over the standard model is quite nontrivial. Let us say that a k-ary relation r over $\mathbb{N}$ is *arithmetical* iff there exists a formula $\varphi(\vec{x})$ with k free variables $\vec{x}$, such that, for every $\vec{n} \in \mathbb{N}^k$:

$$r(\vec{n}) \text{ holds} \quad \text{iff} \quad \mathcal{N} \models \varphi(\underline{\vec{n}}).$$

A function is *arithmetical* iff it is arithmetical as a relation. We have the following theorem of Gödel:

11.1.2. THEOREM. *All partial recursive functions (in particular all recursive functions) are arithmetical.*

In fact, partial recursive functions and relations are just the very beginning of the "arithmetical hierarchy". The above theorem implies in paricular that $\mathrm{Th}(\mathcal{N})$ must be undecidable. Otherwise, membership in every r.e. set would be also decidable.

11.2. Peano Arithmetic

Before Gödel, people thought that it could be possible to axiomatize $\mathrm{Th}(\mathcal{N})$, i.e., to give a simple set of axioms A such that all sentences of $\mathrm{Th}(\mathcal{N})$ would be consequences of A. Peano Arithmetic, abbreviated PA, is such an attempt. The axioms of PA are the following formulas:

- $\forall x\,(x = x)$;
- $\forall x \forall y\,(x = y \to y = x)$;
- $\forall x \forall y\,(\varphi(x) \to x = y \to \varphi(y))$;
- $\forall x \neg(x + 1 = 0)$;
- $\forall x \forall y\,(x + 1 = y + 1 \to x = y)$;
- $\forall x\,(x + 0 = x)$;
- $\forall x \forall y\,(x + (y + 1) = (x + y) + 1)$;
- $\forall x\,(x \cdot 0 = 0)$;
- $\forall x \forall y\,(x \cdot (y + 1) = (x \cdot y) + x)$;
- $\forall x\,(\varphi(x) \to \varphi(x + 1)). \to \varphi(0) \to \forall x\, \varphi(x))$.

The third and the last items are actually axiom schemes, not single axioms. Although the set of axioms is thus infinite, it is still recursive, and thus the set of theorems (derivable sentences) of PA is r.e. The last axiom scheme is called the *induction scheme*.

A theory T (a set of sentences) is *complete* iff for all sentences ψ of the first-order language of T, either $T \vdash \psi$ or $T \vdash \neg\psi$. Gödel's famous incompleteness theorem asserts that PA is not a complete theory. This statement is equivalent to $\mathrm{PA} \neq \mathrm{Th}(\mathcal{N})$, because every $\mathrm{Th}(\mathcal{A})$ is complete.

The importance of Gödel's theorem is that it holds also for all extensions of PA, as long as they are effectively axiomatizable. (A consequence of this is of course that $\mathrm{Th}(\mathcal{N})$ is not r.e.)

11.2.1. THEOREM (Gödel incompleteness). *There is a sentence Z such that neither* $\mathrm{PA} \vdash Z$ *nor* $\mathrm{PA} \vdash \neg Z$.

PROOF. The proof of the theorem is so beautiful that we cannot resist the temptation to sketch here the main idea, which is to express the "liar paradox"[2] in the language of arithmetic. This cannot be done in full, as it would imply inconsistency of arithmetic, but a weaker property will do. Gödel's sentence Z expresses the property "*Z has no proof in* PA". More formally, we have:

$$\mathcal{N} \models Z \quad \text{iff} \quad \mathrm{PA} \nvdash Z.$$

Now if $\mathrm{PA} \vdash Z$ then $\mathcal{N} \models Z$, because $\mathcal{N}$ is a model of PA, and thus $\mathrm{PA} \nvdash Z$. On the other hand, if $\mathrm{PA} \vdash \neg Z$ then $\mathcal{N} \models \neg Z$, but also $\mathcal{N} \models Z$, by the property of Z. Thus Z can be neither proved nor disproved within PA.

The construction of Z is based on the idea of Gödel numbering. Each expression in the language gets a number, and we can write formulas expressing properties of expressions by referring to their numbers. In particular, one can write a formula $T(x, y)$, such that:

$$\mathcal{N} \models T(\underline{n}, \underline{m}) \quad \text{iff} \quad \mathrm{PA} \vdash \varphi_n(\underline{m})$$

whenever n is a number of a formula $\varphi_n(x)$ with one free variable x. The formula $\neg T(x, x)$ must also have a number, say $\neg T(x, x) = \varphi_k(x)$. Thus

$$\mathcal{N} \models \varphi_k(\underline{n}) \quad \text{iff} \quad \mathrm{PA} \nvdash \varphi_n(\underline{n}).$$

The formula Z that says "I have no proof!" can now be defined as $\varphi_k(\underline{k})$:

$$\mathcal{N} \models \varphi_k(\underline{k}) \quad \text{iff} \quad \mathrm{PA} \nvdash \varphi_k(\underline{k}).$$

□

[2] The sentence: *"This sentence is false"* cannot be true and cannot be false.

It was a popular opinion among mathematicians that Gödel's theorem is of little practical importance. Indeed, the formula Z is based on an artifficial diagonalization, and everything one finds in number theory textbooks could be formalized in PA. It is commonly assumed that the first "natural" mathematical problem independent from PA, was shown by Paris and Harrington, see [11]. This problem concerns finite combinatorics and may indeed be considered natural. But it is not a purely arithmetical problem, i.e., it has to be coded into arithmetic. In addition, it was actually invented for the purpose of being independent, rather than suggested by actual mathematical research. We will see later a strong normalization theorem (and these definitely belong to the mathematical practice) independent from PA. This theorem was obtained already by Gödel, and thus is much older than the Paris and Harrington example.

With the Gödel numbers technique, one can express consistency of PA. Indeed, let $T(x)$ be a formula such that:

$$\mathcal{N} \models T(\underline{n}) \quad \text{iff} \quad \mathrm{PA} \vdash \varphi_n,$$

whenever n is a number of a sentence φ_n. Let k be the number of the sentence "$0 = 1$" and let **Con** be the formula "$\neg T(\underline{k})$". Then **Con** expresses consistency of PA:

$$\mathcal{N} \models \mathbf{Con} \quad \text{iff} \quad \text{PA is consistent.}$$

The following theorem was also obtained by Gödel, by a refinement of techniques used for the proof of Theorem 11.2.1.

11.2.2. Theorem (Non-provability of consistency). *If PA is consistent then* PA $\nvdash$ **Con**.

The conclusion is that to prove consistency of arithmetic, one must necessarily use tools from outside the arithmetic.

11.3. Representable and provably recursive functions

We now consider two properties of functions that are stronger than being arithmetical. We not only want our functions to be definable over the standard model, but we want to prove in PA (or some other theory) that the appropriate formula actually defines a function.

In the definition below, the symbol $\exists!$ should be read as "there exists exactly one". Formally, $\exists! x\, \varphi(x)$ is an abbreviation for "$\exists x\, \varphi(x). \wedge \forall y\, (\varphi(y) \to x = y)$".

11.3.1. Definition. We say that a k-ary total function f over $\mathbb{N}$ is *representable* in PA iff there exists a formula $\varphi(\vec{x}, y)$, with $k+1$ free variables $\vec{x}, y$, such that:

1) $f(\vec{n}) = m$ implies $\mathrm{PA} \vdash \varphi(\underline{\vec{n}}, \underline{m})$, for all $\vec{n}, m$;

2) $\mathrm{PA} \vdash \exists! y\, \varphi(\underline{\vec{n}}, y)$, for all $\vec{n} \in \mathbb{N}^k$.

A function is *strongly representable* in PA, if (1) holds and

3) $\mathrm{PA} \vdash \forall \vec{x}\, \exists! y\, \varphi(\vec{x}, y)$.

Each representable function is in fact strongly representable (Exercise 11.8.3) but proving that (2) implies (3) is a brutal application of *tertium non datur*. Of course, each representable function is arithmetical. The converse is not true, but we have the following stronger version of Theorem 11.1.2:

11.3.2. THEOREM (Gödel). *A function is representable in PA if and only if it is recursive.*

The above theorem implies that the totality of every recursive function can actually be proven in PA. However, the excluded middle trick used in Exercise 11.8.3 suggests that such proofs are not necessarily constructive. Proofs required by part (2) of Definition 11.3.1 are constructive, but non-uniform. What we want, is a constructive and uniform proof of termination for all arguments, such that we are able to actually compute the value m of $f(\vec{n})$ from this proof. We should understand however that for this reason we should be concerned with particular algorithms rather than extensionally understood functions. This calls for a finer notion of provable totality.

Recall that, by Kleene's *normal form theorem*, every partial recursive function f can be written as

$$f(\vec{n}) = \pi_2(\mu y. t_f(\vec{n}, y) = 0),$$

where π_2 is a projection (second inverse to the pairing function) and t_f is primitive recursive. The function t_f describes a particular algorithm computing f. Termination of this particular algorithm can be expressed by a formula of the form

$$\forall \vec{x}\, \exists y\, (t_f(\vec{x}, y) = 0).$$

Fortunately, primitive recursive functions do not create any of the above mentioned difficulties. That is, proofs of totality for primitive recursive functions are completely effective. In addition, every primitive recursive function can be uniquely defined by means of equational axioms. Thus, we can actually extend the language of PA by names and defining equations for "as many primitive recursive functions as we wish"[3] without any unwanted side-effects (i.e., this extension of PA is conservative.) It follows that assuming the above formula to be literally a formula of PA is as harmless as it is convenient.

[3]A quotation from [26].

11.3.3. Definition. A recursive function f is said to be *provably total* (or *provably recursive*) in PA iff

$$\mathrm{PA} \vdash \forall \vec{x}\, \exists y\, (t_f(\vec{x}, y) = 0).$$

It is customary to talk about provably recursive *functions*, but what we actually deal with is the notion of a provably recursive *algorithm*. A *function* should be regarded provably total if one of its algorithms is provably total.

The class of functions provably total in PA is very large and includes most of commonly considered functions, and much more, up to unbelievable complexity. But there are recursive functions that are not provably total in PA.

11.4. Heyting Arithmetic

The search for a constructive meaning of classical proof of totality of a recursive function, leads of course to the notion of intuitionistic arithmetic. By *Heyting Arithmetic* (HA), we mean a formal theory based on the following axioms and rules:

- All axioms and rules of first-order intuitionistic logic;
- All axioms of Peano Arithmetic;
- Defining equations for all primitive recursive functions.

This means that HA is a theory in the language of arithmetic, extended by new function symbols for all primitive recursive functions. This extension is not essential, because of conservativity, but is technically very useful.

Here are some interesting properties of HA. (More can be found in [107] and [26].)

11.4.1. Theorem.

1. $\mathrm{HA} \vdash \forall x\, \forall y\, (x = y \vee \neg(x = y))$.
2. *If* $\mathrm{HA} \vdash \exists x\, \varphi(x)$ *for a closed formula* $\exists x\, \varphi(x)$ *then* $\mathrm{HA} \vdash \varphi(\underline{k})$, *for some* $k \in \mathbb{N}$.
3. $\mathrm{HA} \vdash (\varphi \vee \psi) \leftrightarrow \exists x\, ((x = 0 \to \varphi) \wedge (\neg(x = 0) \to \psi))$, *for all* φ, ψ.
4. *If* $\mathrm{HA} \vdash \varphi \vee \psi$, *for closed* φ *and* ψ, *then either* $\mathrm{HA} \vdash \varphi$ *or* $\mathrm{HA} \vdash \psi$.
5. *If* $\mathrm{HA} \vdash \forall x\, (\varphi(x) \vee \neg\varphi(x))$ *and* $\mathrm{HA} \vdash \neg\neg\exists x\, \varphi(x)$ *then* $\mathrm{HA} \vdash \exists x\, \varphi(x)$. *(Markov's Principle)*

Proof.

1) Exercise 11.8.6.

2) See [26] for a semantical proof using Kripke models.

3) Follows from part 2.

4) Follows from parts 2 and 3.

5) From HA $\vdash \neg\neg\exists x\, \varphi(x)$ it follows that PA $\vdash \exists x\, \varphi(x)$, and thus $\mathcal{N} \models \varphi(\underline{k})$, for some k. But we have HA $\vdash \varphi(\underline{k}) \vee \neg\varphi(\underline{k})$ and thus, by (4), either $\varphi(\underline{k})$ or $\neg\varphi(\underline{k})$ is a theorem of HA. In each case we conclude that HA $\vdash \exists x\, \varphi(x)$. (But note that this proof is classical.)

□

11.4.2. Theorem (Kreisel (1958)). *A recursive function is provably total in Peano Arithmetic iff it is provably total in Heyting Arithmetic.*

Proof. The right-to-left part is immediate. We prove the other part. Without loss of generality, we consider the case of a unary function. Let PA $\vdash \forall x\, \exists y\, (t_f(x,y) = 0)$. Thus also PA $\vdash \exists y\, (t_f(x,y) = 0)$. One can show that the Kolmogorov translation (see Chapter 8 and 9) works for arithmetic (see e.g. [107]), so that we obtain HA $\vdash \neg\neg\exists y\, (t_f(x,y) = 0)$. From Theorem 11.4.1(1), we have

$$\text{HA} \vdash \forall x \forall y\, (t_f(x,y) = 0 \vee \neg t_f(x,y) = 0).$$

We apply Markov's Principle (Theorem 11.4.1(5)) to obtain the desired result. □

There is also a direct syntactic proof due to Friedman, which does not require the whole power of Markov's Principle and carries over to second-order arithmetic. See Exercise 11.8.8.

The proof of Kreisel's theorem works as well for any formula of the form $\forall x\, \exists y\, R(x,y)$, where R is a primitive recursive predicate. For instance, R may be just $\bot$, in which case the quantifiers are redundant. We conclude with the following result.

11.4.3. Corollary. HA *is consistent if and only if* PA *is consistent.* □

Kreisel's theorem has the following consequence: classical termination proofs can be made constructive.

11.4.4. Example. Consider a formula of the form $\forall x \exists y\, P(x,y) = 0$ with primitive recursive P. It can be seen as a specification for an input-output relation of a program. A classical or intuitionistic proof of our formula asserts that such a program (a recursive function) exists. A program computing this function can actually be extracted from a constructive proof. For this, let us assume that the signature of arithmetic (and perhaps a bit more) has been added to λP. Then a formula like $\forall n \exists m (n = \underline{2} \cdot m \vee n = \underline{2} \cdot m + 1)$ is inhabited by a proof, i.e., a lambda term. This lambda term M, applied to any specific $\underline{n}$, will evaluate to a normal form **pack** $\mathrm{in}_i(N), \underline{m}$ **to**... for a specific value of m. Thus M is actually a program for dividing numbers by 2.

The little missing point in the above example is the "bit more" to be added to the lambda calculus. We have specific axioms in arithmetic, most notably the induction scheme. And this has to be accounted for by extending the lambda calculus by a primitive recursion operator.

11.5. Kleene's realizability interpretation

The BHK interpretation mentioned in previous chapters relies on the informal notion of a "construction." Kleene [60] proposed a way to make this precise.

The idea is that a construction of a formula is a *number* encoding the constructions of the subformulas of the formula. For instance, a construction of a conjunction $\varphi_1 \wedge \varphi_2$ is a number n encoding a pair of numbers n_1 and n_2, where n_1 and n_2 are constructions of φ_1 and φ_2, respectively.

The main problematic parts of the BHK-interpretation is in the case of implication and universal quantifier. For instance, a construction of an implication $\varphi \to \psi$ is an effective procedure that maps any construction of φ into a construction of ψ. Having settled on $\mathbb{N}$ as the domain of constructions it is now natural to require that a construction of an implication be a number encoding a recursive function that maps any construction of the antecedent to a construction of the succedent.

Below we carry this out in detail, following [61], showing that any formula provable in intuitionistic arithmetic has a construction in this sense.

11.5.1. Definition. Let $e \in \mathbb{N}$ and φ be a closed formula of arithmetic. Then the circumstances under which e *realizes* φ are defined as follows.

1. e realizes A, where A is an atomic formula, if $e = 0$ and A is true;
2. e realizes $\varphi_1 \wedge \varphi_2$ if $e = 2^a \cdot 3^b$ where a realizes φ_1 and b realizes φ_2;
3. e realizes $\varphi_1 \vee \varphi_2$ if $e = 2^0 \cdot 3^a$ and a realizes φ_1, or $e = 2^1 \cdot 3^a$ and a realizes φ_2;
4. e realizes $\varphi_1 \to \varphi_2$ if e is the Gödel number of a partial recursive function f of one argument such that, whenever a realizes φ_1, then $f(a)$ realizes φ_2;

5. e realizes $\exists x\varphi(x)$, where $\varphi(x)$ is a formula containing only x free, if $e = 2^n \cdot 3^a$ where a realizes $\varphi(\underline{n})$.

6. e realizes $\forall x\varphi(x)$, where $\varphi(x)$ is a formula containing only x free, if e is the Gödel number of a general recursive function f of one argument such that $f(n)$ realizes $\varphi(\underline{n})$, for every n.

A closed formula φ is *realizable* if there exists a number e which realizes φ. A formula $\varphi(x_1, \dots, x_k)$ containing only the distinct variables $x_1, \dots, x_k$ $(k \geq 0)$ free is *realizable* if there exists a general recursive function f of k variables such that $f(n_1, \dots, n_k)$ realizes $\varphi(\underline{n_1}, \dots, \underline{n_k})$, for every $n_1, \dots, n_k$.

We shall prove below that every formula provable in HA is realizable. Before proceeding with the details it is convenient to introduce some notation for partial recursive functions.

Let us recall again Kleene's normal form theorem, which we used in Section 11.3. The predicate t_f is in fact uniform in f, or more precisely in the Gödel number of f. That is, for every partial recursive function $f(x_1, \dots, x_n)$ of n variables, there is a number e such that

$$f(x_1, \dots, x_n) = \pi_2(\mu y.\, T_n(e, x_1, \dots, x_n, y)),$$

where T_n is a certain primitive recursive predicate. Informally speaking, T_n states that e is an encoding of f, and y is an encoding of a computation of the function encoded by e (i.e., f) on input $x_1, \dots, x_n$. This encoding is a pair, and the second component of this pair is the output. The projection π_2 extracts this output from y. The number e is called a *Gödel number* of f. We abbreviate $\pi_2(\mu y.\, T_n(e, x_1, \dots, x_n, y))$ by $\Phi_n(e, x_1, \dots, x_n)$.

Also recall that by Kleene's S_n^m *theorem,* there is for every $m, n \geq 0$ an $m+1$-ary primitive recursive function $S_n^m(z, y_1, \dots, y_m)$ such that, if e is a Gödel number of the $m+n$-ary function $f(y_1, \dots, y_m, x_1, \dots, x_n)$, then for each m-tuple $(k_1, \dots, k_m)$ of numbers, $S_n^m(e, k_1, \dots, k_m)$ is a Gödel number of the n-ary function $f(k_1, \dots, k_m, x_1, \dots, x_n)$.

When $f(y_1, \dots, y_n, x_1, \dots, x_m)$ is an $n{+}m$-ary partial recursive function with Gödel number e, we denote by

$$\Lambda x_1 \dots x_m . f(y_1, \dots, y_n, x_1, \dots, x_m)$$

the function $S_n^m(e, y_1, \dots, y_m)$.

Thus, $\Lambda x_1 \dots x_n . f(x_1, \dots, x_n)$ denotes a Gödel number of the function f and $\Lambda x_1 \dots x_n . f(y_1, \dots, y_m, x_1, \dots, x_n)$ denotes a primitve recursive function $f(y_1, \dots, y_m)$ whose value for each m-tuple $(k_1, \dots, k_m)$ of numbers is a Gödel number of the n-ary function $f(k_1, \dots, k_m, x_1, \dots, x_n)$.

We also write $\{z\}(x_1, \dots, x_n)$ for $\Phi_n(z, x_1, \dots, x_n)$.

We then have for any n-tuple $k_1, \dots, k_n$ of numbers that

$$\{\Lambda x_1 \dots x_n . f(y_1, \dots, y_m, x_1, \dots, x_n)\}(k_1, \dots, k_n) \\ \approx f(k_1, \dots, k_m, x_1, \dots, x_n)$$

where $\approx$ means that the two functions have the same domain and have the same results on same arguments inside the domain.

11.5.2. THEOREM. *If $\vdash_{HA} \varphi$ then φ is realizable.*

PROOF. We use a Hilbert-type formulation of HA: this amounts to the Hilbert-type formulation of intuitionistic predicate calculus from Chapter 9, together with the axioms for arithmetic in the first section above.

The proof is by induction on the derivation of $\vdash_{HA} \varphi$. (This requires a generalized induction hypothesis that makes sense for formulas with free variables.) We skip most of the proof, showing only two example cases. The reader may find the details in §82 of Kleene's classic [61].

1. Let the proof be an instance of the axiom $A \to B \to A$ for some A, B. Then $\varphi = A \to B \to A$. We define

   $$e = \Lambda a . \Lambda b . a$$

   (does this look familiar?) To show that e realizes φ, let a realize A. We must show that $\{\Lambda a . \Lambda b . a\}(a)$ i.e.$\Lambda b . a$ realizes $B \to A$. For this end let b realize B, and we must show that $\{\Lambda b . a\}(b)$, i.e., a realizes A, but this holds by assumption.

2. Let the proof be an instance of $(A \to B \to C) \to (A \to B) \to A \to C$ for some A, B, C. Then:

   $$e = \Lambda p . \Lambda q . \Lambda a . \{\{p\}(a)\}(\{q\}(a))$$

□

11.5.3. COROLLARY. HA *is consistent.*

PROOF. There is no number realizing $\perp$. □

11.5.4. COROLLARY. PA *is consistent.*

PROOF. Immediate from Corollaries 11.4.3 and 11.5.3. □

The proof that all intuitionistically provable arithmetical statements have realizers works by mapping proofs into realizers. These realizers are numbers coding recursive functions.

Another similar approach would be to identify the proofs with λ-terms in an appropriate extension of λP. This would avoid the passing back and forth between functions and encoding of functions. And we would have a nice example of a Curry-Howard correspondence. If we define such an extension, then consistency of arithmetic should be inferred from normalization: there is no normal form of type $\bot$.

In fact, it occurs that we do not need dependent types at all to perform a similar construction. It was actually done by Gödel at the simply-typed (propositional) level, with help of his System **T**.

11.6. Gödel's System T

We have seen that in $\lambda\!\to$, the simply typed λ-calculus, very few functions can be defined. For instance, among the numeric functions, only the extended polynomials can be defined.

In this section we consider Gödel's system **T**, which arises from $\lambda\!\to$ by addition of primitive types for numbers and booleans and by addition of primitive recursion and conditionals for computing with these new types. the exposition follows [46] to a large extent.

It will be seen that λT is far more expressible than $\lambda\!\to$. The system was conceived and used by Gödel to prove the consistency of arithmetic.

11.6.1. Definition. Gödel's system **T**, denoted also by λT, is defined as follows.

1. λT has the same set of types as simply typed λ-calculus $\lambda\!\to$, with the following additions:
$$\tau ::= \ldots \mid \mathbf{int} \mid \mathbf{bool}$$
2. λT has the same set of terms as simply typed λ-calculus $\lambda\!\to$ à la Curry, with the following additions:
$$M ::= \ldots \mid \mathbf{z} \mid \mathbf{s}(M) \mid \mathbf{r}(M,N,L) \mid \mathbf{t} \mid \mathbf{f} \mid \mathbf{d}(M,N,L)$$
3. λT has the same set of typing rules as simply typed λ-calculus $\lambda\!\to$ à la Curry, with the following additions:

$$\frac{}{\Gamma \vdash \mathbf{z} : \mathbf{int}} \qquad\qquad \frac{\Gamma \vdash M : \mathbf{int}}{\Gamma \vdash \mathbf{s}(M) : \mathbf{int}}$$

$$\frac{\Gamma \vdash M : \tau \;\&\; \Gamma \vdash N : \tau \to \mathbf{int} \to \tau \;\&\; \Gamma \vdash L : \mathbf{int}}{\Gamma \vdash \mathbf{r}(M,N,L) : \tau}$$

$$\frac{}{\Gamma \vdash \mathbf{t} : \mathbf{bool}} \qquad\qquad \frac{}{\Gamma \vdash \mathbf{f} : \mathbf{bool}}$$

$$\frac{\Gamma \vdash M : \tau \;\&\; \Gamma \vdash N : \tau \;\&\; \Gamma \vdash L : \mathbf{bool}}{\Gamma \vdash \mathbf{d}(M,N,L) : \tau}$$

4. λT has the same set of reduction rules as simply typed λ-calculus $\lambda\!\rightarrow$ à la Curry, with the following additions:

$$\begin{array}{lcl} \mathbf{r}(M,N,\mathbf{z}) & \to & M \\ \mathbf{r}(M,N,\mathbf{s}(L)) & \to & N\,(\mathbf{r}(M,N,L))L \\ & & \\ \mathbf{d}(M,N,\mathbf{t}) & \to & M \\ \mathbf{d}(M,N,\mathbf{f}) & \to & N \end{array}$$

By $\to_T$ we denote the union of $\to_\beta$ and the above reductions.

As mentioned above, **int** and **bool** denote types for integers and booleans, respectively. The term formation operators **z** and **s** denote zero and successor, respectively, as one might imagine.

The two first inference rules for **int** can be seen as *introduction* rules, whereas the third rule for **int** is an *elimination* rule. Analogously, **z** and **s** are *constructors* of type **int** and **r** is a *destructor* of type **int**. Similar remarks apply to the inference rules and term formation operators for **bool**.

11.6.2. REMARK. As always one can study a Church variant and one can study a variant in which pairs are included.

The following two theorems show that the main properties of simply typed λ-calculus are preserved. In particular, the extra expressibility does not come at the expense of loosing strong normalization.

11.6.3. THEOREM. *The relation $\to_T$ is Church-Rosser.*

PROOF. By the Tait–Martin-Löf technique. □

11.6.4. THEOREM. *The relation $\to_T$ is strongly normalizing.*

PROOF. By the method of Tait. □

Recall that Tait's method (which applies to λT with very few adjustments) is based on a construction that assigns a set of terms to every type. If we attempt to formalize Tait's proof, we necessarily must use expressions of the form: *"For all sets A of terms, satisfying ... we have ... "*. We can talk about terms using their numbers, but one cannot enumerate all possible sets of terms, and expressions as the above cannot be coded into first-order arithmetic. One needs to quantify over sets of numbers.

Specifically, Lemma 4.4.3(iii) asserts that *for all types σ the set $[\![\sigma]\!]$ is saturated*. If we could define $[\![\sigma]\!]$ directly then we could write a formula $\xi_\sigma(x)$ expressing in arithmetic that x is a member of $[\![\sigma]\!]$. Then, the statement of our lemma could be expressed in the first-order language. But the definition of $[\![\sigma]\!]$ is by induction, and there is no single formula expressing the property of being a member of $[\![\sigma]\!]$. The definition of $[\![\sigma]\!]$ can only be seen as a set of postulates about a set (unary relation) variable X. Thus, Lemma 4.4.3(iii) can only be formalized as follows: *for all types σ and all sets X, if X satisfies the postulates on $[\![\sigma]\!]$ then X is saturated.* And this involves the quantifier "$\forall X$".

Our first normalization proofs were for simply typed λ-calculus. We proved both weak normalization by the "simple" Turing-Prawitz technique and strong normalization by the "nontrivial" technique due to Tait. There are also "simple" proofs of the strong normalization theorem for $\lambda\rightarrow$. The difference between *simple* and *nontrivial* can be given more precise meaning: the "simple" techniques can be formalized in arithmetic, whereas the "nontrivial" can not.

In the case of λT the situation is different. The Tait proof can be adapted to prove strong normalization for λT, but the simple proofs mentioned above do not work. In fact, any strong normalization proof for System **T** must be "nontrivial". See Corollary 11.7.5.

However, if we restrict attention to finitely many types, the situation is different. We no longer need to quantify over all sets, because we only need finitely many of them, and these can be explicitely defined by formulas. Thus, the whole proof carries over in first-order arithmetic. This situation occurs in particular when we deal with a single function f, definable by a term F, see Proposition 11.6.10.

In λT, one can compute with booleans.

11.6.5. Example. Define

$$\begin{array}{lcl}\mathbf{not}(M) & = & \mathbf{d}(\mathbf{f},\mathbf{t},M)\\ \mathbf{or}(M,N) & = & \mathbf{d}(\mathbf{t},N,M)\\ \mathbf{and}(M,N) & = & \mathbf{d}(N,\mathbf{f},M)\end{array}$$

Then

$$\begin{array}{lcl}\mathbf{not}(\mathbf{t}) & \rightarrow_T & \mathbf{f}\\ \mathbf{not}(\mathbf{f}) & \rightarrow_T & \mathbf{t}\end{array}$$

and we similarly have the expected reductions for **or** and **and**.

We also have some more reductions. For instance,

$$\mathbf{or}(\mathbf{t},N)\rightarrow_T \mathbf{t}$$

However, we do not have

$$\mathbf{or}(N,\mathbf{t})\rightarrow_T \mathbf{t}$$

Indeed, one can show that there is no typable term $G(x,y)$ such that both $G(\mathbf{t},N)\rightarrow_T \mathbf{t}$ and $G(N,\mathbf{t})\rightarrow_T \mathbf{t}$.

Above we have introduced **int** as a representation of integers. The following characterization of normal forms gives the precise interpretation of that.

11.6.6. Proposition. *Let M be a closed normal form and suppose $\vdash M : \tau$.*

1. *if $\tau = \mathbf{int}$ then $M = \underline{m}$, for some $m \in \mathbb{N}$;*
2. *if $\tau = \mathbf{bool}$ then $M = \mathbf{f}$ or $M = \mathbf{t}$;*
3. *if $\tau = \tau_1 \to \tau_2$ then $M = \lambda x.N$.*

In λT one can also compute with integers.

11.6.7. Example. Define for any number $n \in \mathbb{N}$:

$$\underline{n} = \mathbf{s}^n(\mathbf{z})$$

Then we can define

$$\mathbf{plus}(M, N) \quad = \quad \mathbf{r}(M, \lambda x.\lambda y.\mathbf{s}(x), N)$$

Indeed,

$$\begin{array}{rcl} \mathbf{plus}(\underline{m}, \mathbf{z}) & = & \mathbf{r}(\underline{m}, \lambda x.\lambda y.\mathbf{s}(x), \mathbf{z}) \\ & \to_T & \underline{m} \end{array}$$

and

$$\begin{array}{rcl} \mathbf{plus}(\underline{m}, \underline{n+1}) & = & \mathbf{plus}(\underline{m}, \mathbf{s}(\underline{n})) \\ & = & \mathbf{r}(\underline{m}, \lambda x.\lambda y.\mathbf{s}(x), \mathbf{s}(\underline{n})) \\ & \to_T & (\lambda x.\lambda y.\mathbf{s}(x))\ (\mathbf{r}(\underline{m}, \lambda x.\lambda y.\mathbf{s}(x), \underline{n}))\underline{n} \\ & \twoheadrightarrow_T & \mathbf{s}(\mathbf{r}(\underline{m}, \lambda x.\lambda y.\mathbf{s}(x), \underline{n})) \\ & \twoheadrightarrow_T & \mathbf{s}(\underline{m+n}) \\ & = & \underline{m+n+1} \end{array}$$

11.6.8. Definition. A function $f : \mathbb{N}^k \to \mathbb{N}$ is *definable* in λT by a term F if[4]

1. $\vdash F : \mathbf{int}^k \to \mathbf{int}$;
2. $F(\underline{m_1}, \ldots, \underline{m_k}) =_T \underline{f(m_1, \ldots, m_k)}$.

It is an easy exercise to show that multiplication, exponential, predecessor, etc.. are definable.

As long as τ in the typing rules for the new constructs is restriced to base types, i.e., **bool** and **int**, the functions that can be defined in this way are primitive recursive. However, as the type τ increases, more and more functions become definable. In fact, one can show that Ackermann's function is definable in λT (Exercise 11.8.11).

The system λT is also called the system of *primitive recursive functionals of finite type*, because it makes a system of notation for higher-order functions (i.e., functions on functions etc.) defined over the set of integers by means of primitive recursion. (The "al" in "functionals" reflects exactly the point that we deal with higher-order objects.)

[4]$\mathbf{int}^k$ means $\mathbf{int} \to \ldots \to \mathbf{int}$ (k arrows).

11.6.9. Remark. A construction related to the *recursor* $\mathbf{r}$ is the *iterator* $\mathbf{i}$. It has form $\mathbf{i}(M, N, L)$ with the typing rule

$$\frac{\Gamma \vdash M : \tau \ \& \ \Gamma \vdash N : \tau \to \tau \ \& \ \Gamma \vdash L : \mathbf{int}}{\Gamma \vdash \mathbf{i}(M, N, L) : \tau}$$

and reduction rules

$$\begin{array}{lcl} \mathbf{i}(M, N, \mathbf{z}) & \to & M \\ \mathbf{i}(M, N, \mathbf{s}(L)) & \to & N\,(\mathbf{i}(M, N, L)) \end{array}$$

The predecessor function satisfying $\mathbf{p}(\mathbf{s}(x)) \to_T x$ can be defined by the recursor but not by the iterator. However, also the iterator can define the predecessor provided one only requires that $\mathbf{p}(\underline{m+1}) \to_T \underline{m}$. In fact, one can define the recursor from the iterator and pairing, provided one only requires reductions of this form.

The question arises exactly which functions can be defined in λT. It is not difficult to see that every primitive recursive function is definable. It is also not difficult to see that not every recursive function can be definable. Indeed, suppose otherwise. All terms of type $\mathbf{int} \to \mathbf{int}$ can be effectively enumerated, and each of these terms defines a total function (because of strong normalization). Thus, we can enumerate all total recursive functions of one argument: $g_0, g_1, \ldots$. But the function $h(x) = g_x(x)$ is recursive and cannot occur in the sequence. We can have even a tighter upper bound.

11.6.10. Proposition. *All functions definable in* λT *are provably total in* PA.

Proof. If a function f is definable by a term F then this term describes an algorithm to compute f. Thus, a predicate t_f can be effectively computed from F and conversely. Proving the formula $\forall x\, \exists y\, (t_f(x, y) = 0)$ thus reduces to proving that all applications of the form $F\underline{n}$ are strongly normalizable. One can do it (with Tait's technique) so that only finitely many types must be considered. All this argument can be coded into arithmetic. □

We will see later (Theorem 11.7.7) that the class of functions definable in λT coincides with the class of provably total functions of PA.

11.7. Gödel's *Dialectica* interpretation

Gödel introduced System $\mathbf{T}$ as a vehicle to prove consistency of PA. More precisely, he translates each formula of arithmetic into a statement about the primitive recursive functionals of finite type by the so-called *Dialectica interpretation.* The original Gödel's paper is [48]. We will only sketch the main ideas here. More details can be found e.g. in [55, Ch.18].

The basis of the method is a translation of a formula φ in the language of arithmetic into a term φ_D of type **bool**. Suppose first that φ is an atomic formula. Since all primitive recursive functions and predicates are definable in λT, we can define φ_D so that

$$HA \vdash \varphi \quad \text{iff} \quad \varphi_D =_T \mathbf{t}.$$

Here we do not require any additional information about the proof of φ to be preserved by φ_D. (This is because primitive recursive statements are treated as "observables". Compare this to 0 realizing all atomic formulas in Kleene's approach.) For complex formulas, we want more. An ideal situation would be as follows: φ_D has one free variable x, and

$$HA \vdash \varphi \quad \text{iff} \quad \varphi_D[x := M] =_T \mathbf{t}, \quad \text{for some } M.$$

The term M would be the realizer, i.e., it would represent the proof. (Note the distinction: φ_D is a syntactic translation, the computational contents is in M.)

Life is not *that* easy, and we have to settle for something a little less transparent and more complicated. But the essential idea remains the same. Below we work in Church-style λT, extended with product types for simplicity. (This is a syntactic sugar that allows us to identify sequences of types (variables etc.) with single types (variables, etc.) if we find it convenient.)

11.7.1. Definition.

1. For a term M : **bool**, with $\mathrm{FV}(M) = \{z^\sigma\}$, we write $\mathbf{T} \models M$ iff $M[z := Z] =_T \mathbf{t}$, for all closed $Z : \sigma$.
2. We define a λ-*formula* as an expression of the form "$\exists x^\sigma \forall y^\tau D(x, y)$", where x, y are variables, τ and σ are arbitrary types, and $D(x, y)$ has type **bool**.
3. We write $\mathbf{T} \models \exists x^\sigma \forall y^\tau D$ iff there exists a term $X : \sigma$ with $y \notin \mathrm{FV}(X)$, such that $\mathbf{T} \models D[x := X]$.[5]

The variables x and y in the above definition, part (2) may be of product types, thus actually may represent *sequences* of variables.

11.7.2. Definition. For each formula φ in the language of arithmetic, we define a λ-formula $\varphi^D = \exists x \forall y\, \varphi_D$ by induction with respect to φ.

- If φ is an atom (a primitive recursive relation) then $\varphi^D = \exists x^{\mathbf{int}} \forall y^{\mathbf{int}}\, \varphi_D$, where φ_D is a term which defines this relation in λT, and x and y are fresh variables.

[5]The term X should be seen as parameterized by the free variables of D, except y.

Note that $\perp$ is a primitive recursive relation. Thus, as $\perp_D$ we can take e.g. the term $\mathbf{f}$. Also note that the quantifiers $\exists x^{\mathbf{int}}\forall y^{\mathbf{int}}$ are redundant and introduced here just for uniformity.

Let now assume that $\varphi^D = \exists x^\sigma \forall y^\tau\, \varphi_D(x,y)$ and $\psi^D = \exists u^\rho \forall v^\mu\, \psi_D(u,v)$. Then:

- $(\varphi \wedge \psi)^D = \exists xu\forall yv(\varphi_D(x,y) \wedge \psi_D(u,v))$;
- $(\varphi \vee \psi)^D = \exists z^{\mathbf{int}}xu\forall yv((z = 0 \to \varphi_D) \wedge (z \neq 0 \to \psi_D))$;
- $(\varphi \to \psi)^D = \exists {u_1}^{\sigma\to\rho}{y_1}^{\sigma\to\mu\to\tau}\forall xv(\varphi_D(x, y_1xv) \to \psi_D(u_1x, v))$;
- $(\neg\varphi)^D = \exists z^{\sigma\to\tau}\forall x(\neg\varphi_D(x, zx))$;

And now take $\varphi^D = \exists x^\sigma \forall y^\tau\, \varphi_D(x, y, z^{\mathbf{int}})$. Then

- $(\exists z\, \varphi)^D = \exists zx\forall y\, \varphi_D(x,y,z)$;
- $(\forall z\, \varphi)^D = \exists {x_1}^{\mathbf{int}\to\sigma}\forall yz\varphi_D(x_1z, y, z)$

11.7.3. Theorem (Gödel). *If* HA $\vdash \varphi$ *then* $\mathbf{T} \models \varphi^D$.

Proof. The proof is by induction with respect to the proof of φ. We omit this proof, which can be found in [55, Ch.18]. □

The proof of the above theorem is actually building a realizer X for a given proof in HA. The computational contents of the proof in HA is preserved by the realizer as we will see below. Here are some consequences of the interpretation.

11.7.4. Corollary. HA *is consistent.*

Proof. Suppose HA $\vdash \perp$. Then $\mathbf{T} \models \perp_D$, i.e., we have $\mathbf{f} =_T \mathbf{t}$. □

11.7.5. Corollary. *The strong normalization theorem for System* $\mathbf{T}$ *is a statement independent from* PA.[6]

Proof. Otherwise, all the proof of SN would be formalizable in PA. This contradicts Theorem 11.2.2. □

The above result can be explained as follows. Gödel's consistency proof makes use of the normalization theorem for λT, and every other part than this latter result can be proved in PA itself, provided we go through the effort of translating terms to numbers by some Gödel-numbering. By Gödel's theorem about the unprovability of consistency of arithmetic within arithmetic

[6]SN is expressible in the language of arithmetic with help of König's Lemma: *"For each (Gödel number of a) term M there is n such that all reduction paths from M (coded by a Gödel number) consist of at most n steps".*

it follows that the normalization therem cannot be proved in PA—unless PA is inconsistent.

Thus, in proving the normalization for λT we must be using methods which essentially transcend proof techniques that are formalizable in PA, i.e., induction over natural numbers.

In normalization proof for simply typed λ-calculus we used induction on lexicographically order triples, and similarly in the proof of Gentzen's Hauptsatz (the cut-elimination theorem). Such inductions can be reduced to nested inductions and are therefore still formalizable in PA.

One can also view induction on lexicographically ordered tuples as an ordinal induction—induction up to ω^n—which happens to be formalizable in PA. Gentzen discovered that by moving to larger ordinals, one could prove a cut-elimination theorem for arithmetic. More specifically he considers induction up to ϵ_0. This is the first ordinal that cannot be reached by addition, multiplication, and exponentiation.

Also, one can show that the functions definable in λT are the functions which are definable by transfinite recursion up to the ordinals which are strictly smaller than ϵ_0, see [96]. This shows that the expressive power of λT is enormous.

11.7.6. COROLLARY. *All functions provably total in first-order arithmetic are definable in* **T**.

PROOF. Without loss of generality we can consider a unary function. Assume PA $\vdash \forall x\, \exists y\, (t_f(x,y) = 0)$, where t_f is as in Definition 11.3.3. Recall that $t_f(x,y) = 0$ is primitive, and thus treated as an atomic formula. Thus, the translation of the above formula (after removing redundant quantifiers) has the form

$$(\forall x\, \exists y\, (t_f(x,y) = 0))^D = \exists y_1^{\mathbf{int}\to\mathbf{int}} \forall x^\omega D(y_1 x, x),$$

where D represents the relation $t_f(x,y) = 0$.

Since HA $\models (\forall x\, \exists y\, (t_f(x,y) = 0))^D$, we have a realizing term Y_1, such that $D(Y_1 \underline{n}, \underline{n})$ reduces to $\mathbf{t}$. Let $Y_1 \underline{n} =_T \underline{m}$. Then we have $t_f(n,m) = 0$, and thus $f(n) = \pi_2(m)$. Clearly, the projection is primitive recursive and represented by some term Π_2. The conclusion is that the term $\lambda x.\Pi_2(Y_1 x)$ represents f in System **T**. □

Together with Proposition 11.6.10, this gives:

11.7.7. THEOREM. *The functions definable in* λT *are exactly those that are provably total in PA.*

11.7.8. REMARK. Let us now come back to the idea mentioned in Example 11.4.4 of a variant of λP corresponding to first-order arithmetic. We shall argue that System **T** can be seen as a propositional counterpart of such a calculus. Indeed, let us consider again the induction scheme

$$(\forall x{:}\mathbf{int})(\varphi(x) \to \varphi(x+1)). \to \varphi(0) \to (\forall x{:}\mathbf{int})\varphi(x).$$

In order to incorporate arithmetic into λP one has to make sure that every such type is inhabited. One way to do it is to introduce new constants R_φ of the above types. A proof by induction is now represented by a term $R_\varphi M^{(\forall x:\mathbf{int})(\varphi(x)\to\varphi(x+1))} N^\varphi 0$ of type $(\forall x{:}\mathbf{int})\varphi(x)$.

The next question is what should be a reduction rule associated with R_φ. Let $n : \mathbf{int}$ be a specific integer. The term $R_\varphi MNn$ represents a proof of $\varphi(n)$ obtained as follows: first prove the general statement $(\forall x{:}\mathbf{int})\varphi(x)$ by induction and then apply it to n. Such a proof can be seen as containing a redundancy: an introduction of universal quantifier by R_φ followed by elimination of that quantifier. We can avoid redundancy by applying M to N exactly n times instead. This justifies the reduction rules:

- $R_\varphi MN0 \ \to_\beta \ N$;
- $R_\varphi MN(n+1) \ \to_\beta \ Mn(R_\varphi MNn)$.

Now observe what happens to our constant under the dependency erasing translation $M \mapsto \overline{M}$. The type of R_φ becomes:

$$(\mathbf{int} \to \varphi \to \varphi) \to \varphi \to \mathbf{int} \to \varphi(x),$$

which is, up to permutation, the same as the type of the recursor **r**. Also, the reduction rules above differ from those of Definition 11.6.1(4) just in the order of arguments.

11.8. Exercises

11.8.1. Exercise. Show that the following theorems are derivable in PA:

- $\forall x\,(\neg(x=0) \to \exists y(y = x+1))$;
- $\forall x\forall y\forall z\,(x=y \to y=z \to x=z)$;
- $\forall x\forall y\forall z\,((x+y)+z = x+(y+z))$;
- $\forall x\forall y\,(x+y=y+x)$;
- $\underline{2}+\underline{2}=\underline{4}$;
- $\forall x\exists y\,(x=\underline{2}\cdot y \lor x = \underline{2}\cdot y+1)$;
- other common arithmetical properties.

11.8.2. Exercise. Check whether the axiom scheme $\forall x \forall y\, (\varphi(x) \to x = y \to \varphi(y))$ of PA can be replaced by $\forall x \forall y\, (x = y \to x + 1 = y + 1)$ and $\forall x \forall y \forall z\, (x = y \to y = z \to x = z)$.

11.8.3. Exercise. Show that every function representable in PA is strongly representable (and conversely). *Hint:* Consider the formula:

$$(\exists! z\, \varphi(\vec{x}, z).\wedge \varphi(\vec{x}, y)) \vee (\neg \exists! z\, \varphi(\vec{x}, z).\wedge y = 0).$$

11.8.4. Exercise. Show that the condition (2) in Definition 11.3.1 can be replaced by:

2') PA $\vdash \exists y\, \varphi(\underline{\vec{n}}, y)$, for all $\vec{n} \in \mathbb{N}^k$;

2") $\mathcal{N} \models \varphi(\underline{\vec{n}}, \underline{m})$ implies $f(n) = m$, for all $\vec{n}$ and m.

Hint: Consider the formula: $\varphi(\vec{x}, y) \wedge \forall z\, (\varphi(\vec{x}, z) \to y \leq z)$.

11.8.5. Exercise. Consider again the formulas of Exercise 11.8.1. Are they provable in HA?

11.8.6. Exercise. Prove part 1 of Theorem 11.4.1.

11.8.7. Exercise. Let ϱ be a fixed formula. For any formula φ, let φ^{ϱ} be obtained from φ by replacing every atomic subformula ν by $\nu \vee \varphi$. Show that if HA $\vdash \varphi$ then also HA $\vdash \varphi^{\varrho}$.

11.8.8. Exercise (H. Friedman). Prove Theorem 11.4.2, using Exercise 11.8.7 as a lemma. *Hint:* Take ϱ to be $\exists y(t_f(\vec{x}, y) = 0)$, and apply Exercise 11.8.7 to the formula $\neg\neg\exists y(t_f(\vec{x}, y) = 0)$.

11.8.9. Exercise. Show that multiplication, exponentiation, subtraction, and all your favourite integer functions are definable in λT.

11.8.10. Exercise. Here are our favourite integer functions: let f_0 be the successor function, and define $f_{k+1}(x) := f_k^x(x)$ (apply f_k to x exactly x times). Show that all functions f_k are definable i n λT.

11.8.11. Exercise. The *Ackermann function* f_ω is defined by $f_\omega(x) = f_x(x)$. Prove that the Ackermann function is not primitive recursive. *Hint:* Show that every primitive recursive function is majorized by one of the f_k's.

11.8.12. Exercise. Show that the Ackermann function f_ω (Exercise 11.8.11) is definable in λT.

11.8.13. Exercise. Show that all functions $f_{\omega+k+1}(x) = f^x_{\omega+k}(x)$ are definable in λT, as well as the function $f_{\omega \cdot 2}(x) = f_{\omega+x}(x)$.

11.8.14. Exercise. Go ahead, define even faster growing functions, all definable in λT. Will this ever stop?

11.8.15. Exercise. Show that all functions definable in $\lambda\!\!\rightarrow$ are definable in λT. Too easy? Do not use Schwichtenberg theorem.

11.8.16. Exercise. Show that Booleans make syntactic sugar in λT, that is, the class of integer functions definable in λT without Booleans is the same.

CHAPTER 12

Second-order logic and polymorphism

We often say that individuals are objects of *order zero*. Functions and relations on individuals are of *order one*. Further, operations on objects of order n will be themselves classified as being of order $n+1$. This terminology is often used in the metalanguage, referring to problems, systems etc. For instance, the unification problem discussed in Chapter 6, is often called "first-order unification". Note that the unified expressions (unknowns) are of order zero, its the unification itself (an operation on terms) that is first-order. One can also consider *second-order unification*, with function unknowns and so on.

We talk of "first-order logic", because we have first-order predicates, and because quantification can be seen as an operator acting on individuals. So what "second-order logic" should be? Typically, one adds to the language variables ranging over predicates, sets or functions, and quantify over such variables. Thus, second-order logic is usually an extension of first-order logic. However, in presence of second-order quantification, the first-order features become less important than one could expect, and many properties of second-order logic can be studied in a simplified settings: propositional second-order logic. This logic is obtained by adding second-order features directly to propositional calculus. That is, quantifiers are now binding propositional variables.

12.1. Propositional second-order formulas

We extend the language of propositional logic by second-order quantifiers, i.e., quantifiers over propositions. As before, we assume an infinite set PV of propositional variables and we define the *second-order propositional formulas* by induction, represented by the following grammar:

$$2\Phi ::= \bot \mid p \mid (2\Phi \to 2\Phi) \mid (2\Phi \vee 2\Phi) \mid (2\Phi \wedge 2\Phi) \mid \forall p\, 2\Phi \mid \exists p\, 2\Phi,$$

where p ranges over PV. The quantifiers are meant to bind propositional variables within their scope, so that e.g., $\mathrm{FV}(\forall p\,\varphi) = \mathrm{FV}(\varphi) - \{p\}$. (We skip the full definition of FV, leaving this pleasure to the reader, as well as another one: to define the operation of substitution $\varphi[p := \psi]$.) We identify alpha-convertible formulas. Notational conventions are similar to those used for propositional and first-order logic.

The intended meaning of "$\forall p\,\varphi(p)$" is that $\varphi(p)$ holds for all possible meanings of p. The meaning of "$\exists p\,\varphi(p)$" is that $\varphi(p)$ holds for some meaning of p. Classically, there are just two possible such meanings: the two truth values. Thus, the statement $\forall p\,\varphi(p)$ is classically equivalent to $\varphi(\top) \wedge \varphi(\bot)$, and $\exists p\,\varphi(p)$ is equivalent to $\varphi(\top) \vee \varphi(\bot)$. Therefore, every property expressible with quantifiers can be also expressed without.[1] In fact, every function over the two-element Boolean algebra $\{\bot, \top\}$ can be defined with help of ordinary propositional connectives (this property is called *functional completeness*) and thus no extension at all of the propositional language can increase its expressive power.

12.1.1. Warning. The above is no longer true when we add second-order quantification to first-order classical logic, and when the quantified predicates may depend on individual terms. These are no longer just truth-values, but rather truth-valued functions on individuals.

In the intuitionistic logic, there is no finite set of truth-values, and the propositional quantifiers should be regarded as ranging over some infinite space of predicates. (In fact, there is nothing like functional completeness: Kreisel [64] and Goad [47] show predicates non-expressible in propositional intuitionistic logic but definable with help of quantifiers or infinite operations.)

The intuitive meaning of quantified expressions is best explained by means of the Brouwer-Heyting-Kolmogorov interpretation. Note that we deal only with propositions expressible in our language. Indeed, to handle the predicates in a constructive way, we must be able to refer to their proofs.

- *A construction of $\forall p\,\varphi(p)$ is a method (function) transforming every construction of any proposition* $\mathbf{P}$ *into a proof of* $\varphi(\mathbf{P})$.
- *A construction of $\exists p\,\varphi(p)$ consists of a proposition* $\mathbf{P}$*, together with a construction of* $\mathbf{P}$*, and a construction of* $\varphi(\mathbf{P})$.

Syntactically, predicates $\mathbf{P}$ must be themselves represented by formulas. The class of formulas quantifiers range over can be taken to be the full set 2Φ, or a proper subset of 2Φ. We choose the first option (called *full comprehension*) so that the following schemes are valid (for every φ):

[1]But at a certain cost: compare the PSPACE-completeness of satisfiability of quantified Boolean formulas to the NP-completeness of ordinary propositional formulas. It follows that removing quantifiers may cause an exponential increase of the size of a formula.

- $\forall p\, \varphi. \to \varphi[p := \psi]$;
- $\exists p(p \leftrightarrow \varphi)$.

One has to be aware that the full comprehension postulate has the following side-effect, called *impredicativity* of second-order logic. The meaning of a formula $\forall p\, \varphi$ is determined by the meanings of all formulas $\varphi[p := \psi]$, including the cases when ψ is either equal or more complex than $\forall p\, \varphi$ itself. There is no well-founded hierarchy with respect to the semantics, in particular many proof methods based on induction must fail.

On the other hand, the assumption that quantifiers range over *definable* propositions only is a sharp restriction compared to the ordinary understanding of (full) second-order classical logic, as in Warning 12.1.1. However, from a constructive point of view, a *proof* and not a *model* is the ultimate criterium, and thus the syntactic approach should be given priority over the semantic way of thinking. See also Remark 12.2.7.

12.1.2. DEFINITION. Natural deduction.

The natural deduction system for second-order intuitionistic propositional logic consists of the ordinary rules for propositional connectives plus the following rules for quantifiers:

$$(\forall\mathrm{I})\frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall p\, \varphi}\ (p \notin FV(\Gamma)) \qquad\qquad (\forall\mathrm{E})\frac{\Gamma \vdash \forall p\, \varphi}{\Gamma \vdash \varphi[p := \vartheta]}$$

$$(\exists\mathrm{I})\frac{\Gamma \vdash \varphi[p := \vartheta]}{\Gamma \vdash \exists p\, \varphi} \qquad\qquad (\exists\mathrm{E})\frac{\Gamma \vdash \exists p\, \varphi \quad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}\ (p \notin FV(\Gamma, \psi))$$

In the above, the notation $FV(\Gamma)$ is the union of all $\mathrm{FV}(\varrho)$, for $\varrho \in \Gamma$.

One can also define other proof systems, most notably sequent calculus for second-order propositional intuitionistic logic. Cut-elimination proofs for this calculus were obtained independently by several authors; three of them (by Girard, Martin-Löf and Prawitz) are published in the book [34].

12.2. Semantics

We begin, as usual, with the algebraic approach, based on Heyting algebras, although, historically, second-order Kripke semantics was considered first.

12.2.1. DEFINITION. Let $v : V \to \mathcal{H}$ be a valuation of propositional variables in a complete Heyting algebra $\mathcal{H}$. We extend v to arbitrary second-order formulas as follows:

- $v(\varphi \vee \psi) = v(\varphi) \cup v(\psi)$;

- $v(\varphi \wedge \psi) = v(\varphi) \cap v(\psi)$;
- $v(\varphi \to \psi) = v(\varphi) \Rightarrow v(\psi)$;
- $v(\bot) = 0$;
- $v(\forall p\varphi) = \inf\{v_p^a(\varphi) : a \in \mathcal{H}\}$;
- $v(\exists p\varphi) = \sup\{v_p^a(\varphi) : a \in \mathcal{H}\}$.

where v_p^ψ is a valuation defined by $v_p^a(p) = a$, and $v_p^a(q) = v(q)$, for $q \neq p$.

We use the symbol $\models$ in the obvious way, except that we deal now exclusively with complete algebras. For instance, we write $\models \varphi$ (and we say that φ is a tautology) iff φ has the value 1 under all valuations in all complete Heyting algebras.

12.2.2. Theorem (Heyting completeness). *The conditions* $\Gamma \models \varphi$ *and* $\Gamma \vdash \varphi$ *are equivalent.*

Proof. Omitted.[2] See the paper [40] for details. □

The paper of Geuvers [40] is suggested for reading, but algebraic semantics for various second-order and higher-order intuitionistic logics was known before Geuvers, cf. the work of Albert Dragalin [32].

Kripke semantics for second-order propositional formulas was considered by several authors. There are various sorts of models and different variants of the logics under consideration. One should begin with a reference to Prawitz [86], who first proved a completeness theorem for a class of *Beth models*, structures similar in spirit to Kripke models. Then, Gabbay [36, 37] showed completeness for a slight extension of our second-order logic for a restricted class of Kripke models. This result was adjusted by Sobolev [100] so that the Gabbay's axiom (see Remark 12.2.5) was no longer necessary. We recommend the paper [99] of Skvorov for a survey of these results. Our definition below follows the latter paper (up to syntactic sugar).

12.2.3. Definition.

1. A second-order *Kripke model* is a tuple of the form $\mathcal{C} = \langle C, \leq, \{D_c : c \in C\}\rangle$, where C is a non-empty set, $\leq$ is a partial order in C, and the D_c's are families of upward-closed[3] subsets of C, satisfying
$$\text{if } c \leq c' \text{ then } D_c \subseteq D_{c'}.$$
The intuition is that D_c is the family of predicates meaningful at state c.

[2]A difficulty in this proof is that the Lindenbaum algebra of second-order formulas is not complete and has to be *embedded* into a complete one in such a way that the existing joins and meets are preserved.

[3]If $c' \in x \in D_c$ and $c' \leq c''$ then also $c'' \in x$.

2. A *valuation* in $\mathcal{C}$ assigns upward-closed subsets of C to propositional variables. Such a valuation v is *admissible* for a state c iff $v(p) \in D_c$, for all propositional variables p. Clearly, a valuation admissible for c is also admissible for all $c' \geq c$. We write v_p^x for a valuation satisfying $v_p^x(p) = x$, and $v_p^x(q) = v(q)$, for $q \neq p$.

   The forcing relation $c, v \Vdash \varphi$ is defined (when v is admissible for c) as follows:

   - $c, v \Vdash p$ iff $c \in v(p)$;
   - $c, v \Vdash \varphi \vee \psi$ iff $c, v \Vdash \varphi$ or $c, v \Vdash \psi$;
   - $c, v \Vdash \varphi \wedge \psi$ iff $c, v \Vdash \varphi$ and $c, v \Vdash \psi$;
   - $c, v \Vdash \varphi \to \psi$ iff $c', v \Vdash \psi$, for all $c' \geq c$ with $c', v \Vdash \varphi$;
   - $c, v \Vdash \bot$ never happens;
   - $c, v \Vdash \exists p\, \varphi$ iff $c, v_p^x \Vdash \varphi$, for some $x \in D_c$;
   - $c, v \Vdash \forall p\, \varphi$ iff $c', v_p^x \Vdash \varphi$, for all $c' \geq c$, and all $x \in D_{c'}$.

3. A Kripke model is *complete* iff for every formula φ, every c and v, the set $v(\varphi) = \{c' : c', v \Vdash \varphi\}$ is in D_c, whenever v is admissible for c. (If we understand the meaning of propositional variables free in φ then we should understand the formula too.)

4. We write $\Gamma \Vdash \varphi$ iff for every complete Kripke model $\mathcal{C}$, every $c \in C$ and every valuation v admissible for c, such that c, v forces all formulas in Γ, we also have $c, v \Vdash \varphi$.

The completeness theorem for Kripke models in the form below should probably be attributed to Sobolev [100].

12.2.4. Theorem (Kripke completeness). *The conditions* $\Gamma \Vdash \varphi$ *and* $\Gamma \vdash \varphi$ *are equivalent.*

12.2.5. Remark. The additional axiom scheme used by Gabbbay is:

$$\forall p(\psi \vee \varphi(p)). \to \psi \vee \forall p\, \varphi(p), \qquad \text{where } p \notin \mathrm{FV}(\psi).$$

This is a classical second-order tautology, but not an intuitionistic tautology. The class of models corresponding to propositional second-order intuitionistic logic extended with Gabbay's axiom (called also Grzegorczyk schema) is obtained by postulating that all D_c are equal to each other (models with constant domains).

12.2.6. Remark. Note that the postulate of completeness of Kripke models reflects the idea of impredicativity. Indeed, it guarantees that the range of a quantified variable includes every definable predicate. In fact, if we do not require completeness, the following tautology schemes (expressing full comprehension) would be no longer valid:

- $\forall p\, \varphi \to \varphi[p := \psi]$;
- $\exists p\, (p \leftrightarrow \varphi)$.

Observe that completeness cannot be replaced by a modified definition of forcing,

$$c, v \Vdash \forall p\, \varphi \quad \text{iff} \quad c', v_p^{v(\psi)} \Vdash \varphi, \qquad \text{for all } c' \geq c, \text{ and all formulas } \psi,$$

because such a definition would be circular. (Take $\psi = \forall p \varphi$.)

12.2.7. Remark. It is tempting to consider Kripke models with all D_c equal to the family of all upward-closed subsets of C (*principal* Kripke models). Unfortunately, the class of all formulas valid in principal models is not recursively enumerable, and thus non-axiomatizable in a finitary way. This result is due to Skvorov [99] and independently to Kremer [65].

Of course the above mentioned results of Skvorov and Kremer imply that the set of second-order sentences true in all principal models cannot be decidable or even recursively enumerable. The set of provable sentences of our second-order intuitionistic logic is of course recursively enumerable. But it is undecidable.

12.2.8. Theorem.
It is undecidable whether a given formula $\varphi \in 2\Phi$ has a proof.

The first proof of undecidability was given by Gabbay [36, 37]. But this proof applies to the logic extended by Gabbay's axiom (see Remark 12.2.5), and it does not extend automatically to the pure intuitionistc case. The proof is using completeness theorem, for Kripke models with constant domains. The paper [100] of Sobolev filled this gap, and allowed to infer Theorem 12.2.8 by essentially the same method. In the meantime, M.H. Löb [67] has published another proof of Theorem 12.2.8. The main result of [67] which implies undecidability, is an effective translation from first-order classical logic to second-order intuitionistic logic. Unfortunately, Löb's paper is quite incomprehensible. It has been later slightly simplified by Arts [3] and Arts and Dekkers [4], but the resulting presentations are still quite complicated. A simpler, syntactic proof of Theorem 12.2.8 can be found in [112].

12.3. Polymorphic lambda-calculus (System F)

The polymorphic lambda-calculus $\lambda 2$, often referred to as "System **F**" is an excellent example of a Curry-Howard correspondence and provides a surprising evidence for the relationships between logic and computer science. This system was actually invented twice: by the logician Jean-Yves Girard [44] and by the computer scientist John Reynolds [90]. The first one's goal was to design a proof notation needed for his work on second-order logic, the other's idea was to build a type system for a polymorphic programming language. The results (after dissolving the syntactic sugar) were essentially the same.

For a treatment of System **F** extending the scope of the present notes, one can consult the following books: [45, 66, 74].

12.3.1. Definition.

1. (Second-order) *types* are defined as follows:

   - Type variables are types;
   - If σ and τ are types then $(\sigma \to \tau)$ is a type;
   - If σ is a type and α is a type variable, then $\forall\alpha\,\sigma$ is a type.

   Thus, types coincide with second-order propositional formulas over $\to$ and $\forall$ only.

2. Well-typed lambda-terms (Church style) are defined by the type inference rules below. Every term is either a variable, an ordinary application or abstraction, or it is

   - a *polymorphic abstraction*, written $\Lambda\alpha.M$, where M is a term and α is a type variable, or
   - a *type application*, written $(M\tau)$, where M is a term and τ is a type.

The intuitive meaning of $\Lambda\alpha.M$ is that the term M (which may refer to a free type variable α) is taken as a polymorphic procedure with a type parameter α. Type application correspond to a call to such a generic procedure with an actual type parameter. This is an *explicit* form of polymorphism (type as parameter) as opposed to *implicit* polymorphism of ML.

12.3.2. Definition (Type inference rules). A *context* is again a finite set of declarations $(x : \tau)$, for different variables (i.e., finite partial function from variables to types). The axiom and the rules for $\to$ are as usual:

$$\Gamma, x{:}\tau \vdash x : \tau$$

$$\frac{\Gamma \vdash N : \tau \to \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash NM : \sigma} \qquad\qquad \frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\tau.M : \tau \to \sigma}$$

and we also have rules for $\forall$ corresponding to natural deduction rules ($\forall$I) and ($\forall$E)

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash (\Lambda\alpha.M) : \forall\alpha\,\sigma}\ (\alpha \notin FV(\Gamma)) \qquad\qquad \frac{\Gamma \vdash M : \forall\alpha\,\sigma}{\Gamma \vdash M\tau : \sigma[\alpha := \tau]}$$

Let us recall that a Church-style term makes sense for us only within a context that assigns types to free variables. We sometimes stress this fact by placing (informally) an upper index, as for instance in $x^{\forall\alpha(\sigma\to\rho)}\tau y^{\sigma[\alpha:=\tau]}$. In fact, finding a proper decoration of free variables in a given term expression (to obtain a proper Church-style term) is, surprisingly, an undecidable problem, see [95].

12.3.3. Convention. As in previous chapters, we sometimes write upper indices to mark types of certain (sub)terms. Also, we sometimes write e.g. $\lambda x^\tau . M$, rather than $\lambda x{:}\tau . M$, to improve readability.

As should be now clear from the rules, the universal type $\forall\alpha\,\sigma(\alpha)$ corresponds to a product construction of the form $\prod_{\tau\in\star}\sigma(\tau)$. This answers very well to the idea that a proof of $\forall\alpha\,\sigma(\alpha)$ is a function translating proofs of τ into proofs of $\sigma(\tau)$.

From the programmer's point of view, $\forall\alpha\,\sigma$ is a type of a polymorphic procedure. Note that the restriction $\alpha \notin FV(\Gamma)$ in the $\forall$ introduction rule (called also generalization rule) corresponds to that a type parameter must be a *local* identifier.

12.3.4. Example. Here are some well-typed Church-style terms:

- $\vdash \Lambda\alpha.(\lambda x^{\forall\alpha(\alpha\to\alpha)}.x(\beta\to\beta)(x\beta) : \forall\beta(\forall\alpha(\alpha\to\alpha).\to\beta\to\beta)$;
- $\vdash \Lambda\alpha.\lambda f^{\alpha\to\alpha}\lambda x^\alpha.f(fx) : \forall\alpha((\alpha\to\alpha)\to(\alpha\to\alpha))$;
- $\vdash \lambda f^{\forall\alpha(\alpha\to\beta\to\alpha)}\Lambda\alpha\lambda x^\alpha.f(\beta{\to}\alpha)(f\alpha x) : \forall\alpha(\alpha\to\beta\to\alpha).\to\forall\alpha(\alpha\to\beta\to\beta\to\alpha)$.

12.3.5. Theorem (Curry-Howard isomorphism). *We have $\Gamma\vdash M:\tau$ in the polymorphic lambda calculus if and only if $|\Gamma|\vdash\tau$ has a proof in the $\{\forall,\to\}$-fragment of the second-order intuitionistic propositional logic.*

12.3.6. Corollary. *The inhabitation problem for the polymorphic lambda calculus* ("Given type τ, is there a closed term of type τ?") *is undecidable.*

Proof. Immediate from the above and Theorem 12.2.8. $\square$

12.3.7. Warning. We skip the detailed definition of free variables and substitution. The reader should be able to write this definition herself, provided she remembers about the following:

- There are free object variables in terms as well as free type variables. The latter occur in the lambda abstractions "$\lambda x{:}\tau$" and in type applications. Thus we have to consider substitutions of the form $M[x := N]$ and of the form $M[\alpha := \tau]$.

- There are two binding operators in terms: the big and the small lambda. Substitutions must account for both. And note that the term N in $M[x := N]$ may contain free type variables that are bound in M. Thus, both sorts of renaming may be necessary.

- The definition of alpha conversion must account for the two sorts of bindings.

- Binding an object variable x^ρ does not mean binding type variables in ρ.

- The effect of substitution $x[\alpha := \tau]$ is of course x. But if we work in a context containing a declaration $(x : \sigma)$, then one should better understand it this way: $x^\sigma[\alpha := \tau] = x^{\sigma[\alpha:=\tau]}$. This is because what we want is $\Gamma[\alpha := \tau] \vdash M[\alpha := \tau] : \sigma[\alpha := \tau]$ whenever $\Gamma \vdash M : \sigma$.

12.3.8. Definition (Beta reduction).
There are two sorts of beta reduction rules:

- Object reduction: $(\lambda x{:}\tau.M)N \longrightarrow_\beta M[x := N]$;

- Type reduction: $(\Lambda\alpha.M)\tau \longrightarrow_\beta M[\alpha := \tau]$.

This notion of reduction has the expected properties, in particular it is Church-Rosser and preserves types.

12.4. Expressive power

In classical logic, the connectives $\neg$, $\vee$ and $\wedge$ can be defined by means of $\bot$ and $\to$. The quantifier $\exists$ is also expressible via the De Morgan law, so that $\bot$, $\to$ and $\forall$ make a sufficient set of operators. This is not the case in intuitionistic logic, neither propositional nor first-order. However, in second-order propositional logic, this opportunity appears again. And we can even get more: using $\to$ and $\forall$ one can express the other connectives and also the constant $\bot$. We have postponed these definitions until now, in order to accompany them with term notation.

12.4.1. Definition (Absurdity). We define

$$\bot := \forall\alpha\,\alpha.$$

We have the following term assignment to rule (E$\bot$):

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash M\tau : \tau}$$

It is easy to see that there is no closed term in normal form that can be assigned type $\forall\alpha\,\alpha$. It will follow from strong normalization that $\bot$ is an empty type.

12.4.2. Definition (Conjunction (product)).

Let α be not free in τ nor σ. Then

$$\tau \wedge \sigma := \forall\alpha((\tau \to \sigma \to \alpha) \to \alpha).$$

(Read this definition as: $\tau \wedge \sigma$ holds iff everything holds that can be derived from $\{\tau, \sigma\}$.)

Lambda-terms related to conjunction are pairs and projections. We define them as follows:

- $\langle P, Q\rangle := \Lambda\alpha\lambda z^{\tau\to\sigma\to\alpha}.\, zPQ$;
- $\pi_1(M^{\tau\wedge\sigma}) := M\tau(\lambda x^\tau\lambda y^\sigma.\, x)$;
- $\pi_2(M^{\tau\wedge\sigma}) := M\sigma(\lambda x^\tau\lambda y^\sigma.\, y)$.

It is left to the reader to check that the term assignment to the $\wedge$-related rules of natural deduction, described in Section 4.2:

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

is correct, as well as the beta-reduction $\pi_i(\langle P_1, P_2\rangle) \to_\beta P_i$ is implemented (but with $\to_\beta$ replaced by $\twoheadrightarrow_\beta$). Note however that eta-conversion is *not* implemented: if $y^{\tau\wedge\sigma}$ is a variable then $\langle\pi_1(y), \pi_2(y)\rangle$ is a normal form.

12.4.3. Definition (Disjunction (variant)).

We define the disjunction of τ and σ as their weakest common consequence. That is, $\tau \vee \sigma$ holds iff all common consequences of τ and σ hold. Formally, for $\alpha \notin \mathrm{FV}(\sigma) \cup \mathrm{FV}(\tau)$, we take:

$$\tau \vee \sigma := \forall\alpha((\tau \to \alpha) \to (\sigma \to \alpha) \to \alpha).$$

We define injections and case eliminator this way:

- $\mathrm{in}_1(M^\tau) := \Lambda\alpha\lambda u^{\tau\to\alpha}\lambda v^{\sigma\to\alpha}.\, uM$;
- $\mathrm{in}_2(M^\sigma) := \Lambda\alpha\lambda u^{\tau\to\alpha}\lambda v^{\sigma\to\alpha}.\, vM$;
- $\mathrm{case}(L^{\tau\vee\sigma};\, x^\tau.M^\rho;\, y^\sigma.N^\rho) := L\rho(\lambda x^\tau.M)(\lambda y^\sigma.N)$.

The reader is invited to check the correctness of rules:

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

$$\frac{\Gamma \vdash L : \psi \vee \varphi \quad \Gamma, x : \psi \vdash M : \rho \quad \Gamma, y : \varphi \vdash N : \rho}{\Gamma \vdash \mathrm{case}(L; x.M; y.N) : \rho}$$

as well as the correctness of beta reduction (Exercise 12.7.6).

Before we discuss the existential quantifier, let us observe that various data types can be implemented in System **F**. For instance, type **Bool** can be interpreted as $\forall\alpha(\alpha \to \alpha \to \alpha)$, with **true** $= \Lambda\alpha\lambda x^\alpha y^\alpha.\, x$ and **false** $= \Lambda\alpha\lambda x^\alpha y^\alpha.\, y$. Integers are represented by the type

$$\omega := \forall\alpha((\alpha \to \alpha) \to \alpha \to \alpha),$$

with the polymorphic Church numerals

$$c_n := \Lambda\alpha\lambda f^{\alpha\to\alpha} x^\alpha.\, f(\cdots f(x)\cdots)$$

representing numbers. We can now generalize the notion of a *definable* integer function in the obvious way. Clearly, all functions definable in simple types can also be defined in System **F** by simply adding some Λ's at the beginning. For instance, we define the successor function as:

$$\mathrm{succ} := \lambda n^\omega.\Lambda\alpha.\lambda f^{\alpha\to\alpha} x^\alpha.\, f(n\alpha f x).$$

But one can do much more, for instance the function $n \mapsto n^n$ can be defined as follows:

$$\mathrm{Exp} := \lambda n^\omega.\, n(\alpha \to \alpha)(n\alpha).$$

Note that this trick uses polymorphism in an essential way. We can generalize it to represent primitive recursion. Indeed, System **T** (as an equational theory) can be embedded into System **F**.

12.4.4. PROPOSITION. *For a given type σ, define $\mathbf{r}_\sigma : \sigma \to (\sigma \to \omega \to \sigma) \to \omega \to \sigma$ as the following term:*

$$\lambda y^\sigma \lambda f^{\sigma\to\omega\to\sigma} \lambda n^\omega.\, \pi_1(n(\sigma\wedge\omega)(\lambda v^{\sigma\wedge\omega}.\, \langle f(\pi_1(v))(\pi_2(v)), \mathrm{succ}(\pi_2(v))\rangle)\langle y, c_0\rangle).$$

Then $\mathbf{r}_\sigma MN(c_0) =_\beta M$ and $\mathbf{r}_\sigma MN(\mathrm{succ}(n)) =_\beta N(\mathbf{r}_\sigma MNn)n$,

PROOF. Exercise 12.7.8. □

The reader is invited to define representation of various other data types in Exercise 12.7.7.

Let us now consider the existential quantifier. We need a term assignment the introduction and elimination rules. One possibility is as follows:

$$(\exists\mathrm{I})\frac{\Gamma \vdash M : \sigma[\alpha := \tau]}{\Gamma \vdash \mathbf{pack}\ M, \tau\ \mathbf{to}\ \exists\alpha.\,\sigma : \exists\alpha.\,\sigma}$$

$$(\exists\mathrm{E})\frac{\Gamma \vdash M : \exists\alpha.\,\sigma \quad \Gamma, x : \sigma \vdash N : \rho}{\Gamma \vdash \mathbf{abstype}\ \alpha\ \mathbf{with}\ x : \sigma\ \mathbf{is}\ M\ \mathbf{in}\ N : \rho}\ (\alpha \notin FV(\Gamma, \rho))$$

As in the first-order case, existential quantification corresponds to data abstraction. An existential type of the form $\exists\alpha\,\sigma$ can be seen as a partial type specification, where type α is "private" and not accessible for the user. For instance, one can consider a type of push-down stores (with the *push* and *pop* operations) defined as

$$\omega\textit{-pds} := \exists\alpha(\alpha \wedge (\omega \to \alpha \to \alpha) \wedge (\alpha \to \alpha \wedge \omega)).$$

A user can operate on such a pds without knowing the actual type used to implement it. A generic pds type may now be defined this way:

$$\textit{generic-pds} := \forall\beta\exists\alpha(\alpha \wedge (\beta \to \alpha \to \alpha) \wedge (\alpha \to \alpha \wedge \beta)).$$

The beta reduction rule for existential type constructors is as follows:

$$\mathbf{abstype}\ \alpha\ \mathbf{with}\ x : \sigma\ \mathbf{is}\ \mathbf{pack}\ M, \tau\ \mathbf{to}\ \exists\alpha.\,\sigma\ \mathbf{in}\ N \longrightarrow_\beta N[\alpha := \tau][x := M].$$

This corresponds to using an abstract type in a context where an actual implementation may be hidden from the user. More on existential types can be found in Mitchell's book [74].

Existential quantification can be represented in System **F** as follows:

12.4.5. Definition. Assuming $\beta \notin \mathrm{FV}(\sigma)$, we define

$$\exists\alpha\,\sigma := \forall\beta(\forall\alpha(\sigma \to \beta).\to \beta).$$

The packing and unpacking terms are as follows:

- $\mathbf{pack}\ M, \tau\ \mathbf{to}\ \exists\alpha.\,\sigma = \Lambda\beta.\lambda x^{\forall\alpha(\sigma\to\beta)}.\, x\tau M;$
- $\mathbf{abstype}\ \alpha\ \mathbf{with}\ x : \sigma\ \mathbf{is}\ M\ \mathbf{in}\ N^\rho = M\rho(\lambda\alpha.\lambda x^\sigma.N).$

Compare the above definition to Definition 12.4.3. We again have the weakest common consequence of all $\sigma(\alpha)$, for arbitrary type α. This supports the understanding of existential quantifier as infinite disjunction. But note also that there is a similarity to De Morgan's law here: take $\bot$ instead of β and we obtain $\neg\forall\alpha\neg\sigma$. We leave to the reader the verification that beta reduction is correctly implemented.

12.5. Curry-style polymorphism

The Curry-style variant of System **F** is defined by the following type assignment rules for pure lambda terms. These rules correspond exactly to these in Definition 12.3.2. (The notion of a type and a context remains the same.)

$$\Gamma, x{:}\tau \vdash x : \tau$$

$$\frac{\Gamma \vdash N : \tau \to \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash NM : \sigma} \qquad \frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall \alpha\, \sigma}\ (\alpha \notin FV(\Gamma)) \qquad \frac{\Gamma \vdash M : \forall \alpha\, \sigma}{\Gamma \vdash M : \sigma[\alpha := \tau]}$$

Rules for abstraction and application are the same as for the simply-typed Curry-style lambda-calculus. Rules for introducing and eliminating the universal quantifier (called respectively *generalization* and *instantiation* rules) reflect the idea of *implicit* polymorphism: to have the universal type $\forall\alpha\,\tau$ means to have all possible instances of this type (all types $\tau[\alpha := \sigma]$).

12.5.1. Definition. The erasure map $|\cdot|$ from terms of Church-style System **F** to pure lambda terms, is defined by the following clauses:

$$\begin{array}{rcl} |x| & = & x \\ |MN| & = & |M||N| \\ |\lambda x{:}\sigma.\, M| & = & \lambda x.\, |M| \\ |\Lambda\alpha.\, M| & = & |M| \\ |M\tau| & = & |M| \end{array}$$

12.5.2. Proposition. *For a pure lambda term M, we have $\Gamma \vdash M : \tau$ if and only if there is a Church-style term M_0 with $|M_0| = M$, such that $\Gamma \vdash M_0 : \tau$.*

Proof. Easy.

A Church-style term M_0 can be seen as a type derivation for a Curry-style term $|M_0|$. The following theorem follows from the fact that every beta reduction performed in a typable pure lambda term corresponds to a reduction in the corrsponding Church-style term.

12.5.3. Theorem (Subject reduction). *Let $\Gamma \vdash M : \tau$, for a pure lambda term M. Then $M \to_\beta M'$ implies $\Gamma \vdash M' : \tau$.*

Proof. Omitted. □

The above result is not as obvious as it can perhaps appear at the first look. To see this, consider the following example:

12.5.4. Example. We have the following correct type assignment:

$$x : \alpha \to \forall\beta\,(\beta \to \beta) \vdash \lambda y.\, xy : \alpha \to \beta \to \beta,$$

and the eta-reduction $\lambda y.\, xy \to_\eta x$. However,

$$x : \alpha \to \forall\beta\,(\beta \to \beta) \not\vdash x : \alpha \to \beta \to \beta.$$

Observe that the Church-style term corresponding to $\lambda y.\, xy$ in our example is $\lambda y{:}\alpha.\, xy\beta$ and is *not* an η-redex. Thus, the reason why Curry-style System **F** is not closed under η-reductions is that there are Curry-style η-redexes that do not correspond to any Church-style redex.

12.5.5. Remark. Closure under η-reductions can be obtained (see Mitchell's paper [73]) by adding to the system a certain subtyping relation $\leq$ together with a *subsumption rule* of the form

$$\frac{\Gamma \vdash M : \tau, \quad \tau \leq \sigma}{\Gamma \vdash M : \sigma.}$$

12.5.6. Remark. Adding existential quantification to Curry-style version of System **F** results with the same problem as that described in Section 9.10. See Exercise 12.7.10.

With polymorphism, one can assign types to many pure lambda terms which are untypable in simple types. A prominent example is $\lambda x.\, xx$, and another one is $c_2\mathbf{K}$. As we will see below, only strongly normalizable terms can be typable, because of strong normalization. But there are strongly normalizable terms, untypable in **F**. The first such example[4] was given by Simona Ronchi Della Rocca and Paola Giannini in the paper [43], and it is the following term:

$$(\lambda zy.\, y(z\mathbf{I})(z\mathbf{K}))(\lambda x.\, xx).$$

The essential thing here is that we cannot find *one* type for $(\lambda x.\, xx)$ that could be applied to both **I** and **K**. Another example is:

$$c_2c_2\mathbf{K}.$$

Compare the latter to the typable term $c_2(c_2\mathbf{K})$.

It was long an open question whether the type reconstruction and type checking problem for System **F** was decidable. Both were shown undecidable by Joe Wells.

12.5.7. Theorem (Wells [117]). *Type reconstruction and type checking in the second-order λ-calculus are recursively equivalent and undecidable.*

Proof. Too long. □

[4]apparently based on an idea of Furio Honsell.

12.6. Strong normalization of second-order typed λ-calculus

We end the chapter by extending the proof of strong normalization of simply typed λ-calculus from Chapter 4 to second-order typed λ-calculus à la Curry.

As mentioned earlier, the standard method of proving strong normalization of typed λ-calculi was invented by Tait [104] for simply typed λ-calculus and generalized to second-order typed λ-calculus by Girard [44].

Our presentation follows again [8].

12.6.1. Definition.

(i) The set of type variables is denoted U and the set of second-order types is denoted by Π_2.

(ii) A *valuation* in $\mathbb{S}$ is a map

$$\xi : U \to \mathbb{S}.$$

(iii) For a valuation ξ, define the valuation $\xi\{\alpha := X\}$ by: defined by

$$\xi\{\alpha := X\}(\beta) = \begin{cases} X & \text{if } \alpha = \beta \\ \xi(\beta) & \text{otherwise} \end{cases}$$

(iv) For each valuation ξ in $\mathbb{S}$ and each $\sigma \in \Pi_2$ the set $[\![\sigma]\!]_\xi$ is defined by:

$$\begin{array}{lcl} [\![\alpha]\!]_\xi & = & \xi(\alpha) \\ [\![\sigma \to \tau]\!]_\xi & = & [\![\sigma]\!]_\xi \to [\![\tau]\!]_\xi \\ [\![\forall\alpha.\sigma]\!]_\xi & = & \bigcap_{X\in\mathbb{S}} [\![\sigma]\!]_{\xi\{\alpha:=X\}} \end{array}$$

12.6.2. Lemma. *For each valuation ξ in $\mathbb{S}$ and $\sigma \in \Pi_2$, we have $[\![\sigma]\!]_\xi \in \mathbb{S}$.*

Proof. Similar to the corresponding proof for $\lambda\!\to$. □

12.6.3. Definition.

(i) Let ρ be a substitution (i.e., a map from term variables to Λ), and ξ be a valuation in $\mathbb{S}$. Then

$$\rho, \xi \models M : \sigma \Leftrightarrow [\![M]\!]_\rho \in [\![\sigma]\!]_\xi$$

(ii) Let ρ be a substitution and ξ be a valuation in $\mathbb{S}$. Then

$$\rho, \xi \models \Gamma \Leftrightarrow \rho, \xi \models x : \sigma, \text{ for all } x : \sigma \text{ in } \Gamma$$

(iii) Finally,

$$\Gamma \models M : \sigma \Leftrightarrow \forall\rho\forall\xi[\rho, \xi \models \Gamma \Rightarrow \rho, \xi \models M : \sigma]$$

12.6.4. Proposition (Soundness).

$$\Gamma \vdash M : \sigma \Rightarrow \Gamma \models M : \sigma$$

PROOF. Similar to the proof for the corresponding property of $\lambda{\to}$, i.e., by induction on the derivation of $\Gamma \vdash M : \sigma$. There are two new cases corresponding to the $\forall$-rules.

If the derivation ends in

$$\frac{\Gamma \vdash M : \forall \alpha . \sigma}{\Gamma \vdash M : \sigma\{\alpha := \tau\}}$$

then by the induction hypothesis

$$\Gamma \models M : \forall \alpha . \sigma$$

Now suppose $\rho, \xi \models \Gamma$, and we are to show that $\rho, \xi \models M : \sigma\{\alpha := \tau\}$. We have

$$[\![M]\!]_\rho \in [\![\forall \alpha . \sigma]\!]_\xi = \bigcap_{X \in \mathbb{S}} [\![\sigma]\!]_{\xi\{\alpha := X\}}$$

Hence

$$[\![M]\!]_\rho \in [\![\sigma]\!]_{\xi\{\alpha := [\![\tau]\!]_\xi\}} = [\![\sigma\{\alpha := \tau\}]\!]_\xi .$$

If the derivation ends in

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall \alpha . \sigma}$$

where α is not free in Γ, then by the induction hypothesis,

$$\Gamma \models M : \sigma .$$

Suppose $\rho, \xi \models \Gamma$, we are to show that $\rho, \xi \models M : \forall \alpha . \sigma$. Since α is not free in Γ, we have for all $X \in \mathbb{S}$ that $\rho, \xi\{\alpha := X\} \models \Gamma$. Therefore,

$$[\![M]\!]_\rho \in [\![\sigma]\!]_{\xi\{\alpha := X\}}$$

for all $X \in \mathbb{S}$. Hence also

$$[\![M]\!]_\rho \in [\![\forall \alpha . \sigma]\!]_\xi$$

as required. □

12.6.5. THEOREM (Strong normalization). *If* $\Gamma \vdash M : \sigma$ *then* M *is strongly normalizing.*

PROOF. Similar to the corresponding proof for $\lambda{\to}$. □

The above proof differs in a very substantial way from Tait's strong normalization proof for simple types or for System **T**. The difference is that a formalization of this argument necessarily requires quantification over *sets of sets* of λ-terms.

Indeed, let us compare the statement of Lemma 12.6.2 and Lemma 4.4.3 part (iii). As we have noticed in Chapter 11, the latter requires quantification over sets X satisfying the definition of $[\![\sigma]\!]$. But then we could define a fixed set $[\![\sigma]\!]$ for each σ by induction. Because of impredicativity, such a definition of $[\![\sigma]\!]$ would now become circular and we must consider various *candidates* $[\![\sigma]\!]_\xi$ for the position of $[\![\sigma]\!]$. Each ξ gives a family of such candidates (sets), and the quantification over ξ in the statement of Lemma 12.6.2 is equivalent to quantifications over these families.

In other words, our proof cannot be formalized even in the second-order language of arithmetic. In fact, we can repeat Gödel's argument and obtain a consistency proof for second-order Peano Arithmetic in this way. Thus, strong normalization for System **F** makes an example of a statement independent from second-order Peano Arithmetic.

But as long as we are only interested in strong normalization of a set of terms involving only finitely many types, we can proceed within second-order arithmetic. Indeed, in this case, the functions ξ are of finite domains and can be handled as tuples of sets. Thus, if (an algorithm of) an integer function is definable in **F**, then its totality can be shown within second-order arithmetic. On the other hand, the *Dialectica* interpretation, generalized by Girard for System **F**, allows one to derive a definition in **F** for every function provably total in second-order arithmetic. We obtain the following result:

12.6.6. THEOREM (Girard). *The class of functions definable in* **F** *coincides with the class of provably recursive functions of second-order Peano Arithmetic.*

12.7. Exercises

12.7.1. EXERCISE. Prove that Gabbay's axiom holds in all Kripke models with constant domains, but not in all Kripke models.

12.7.2. EXERCISE. Can you simplify the definition of forcing of universal formulas in models with constant domains? *Hint:* Yes, you can.

12.7.3. EXERCISE. Show that in complete Kripke models, the semantics of the defined connectives (see Section 12.4) coincides with the semantics of their definitions. Does this hold in non-complete models?

12.7.4. EXERCISE. Show that the type assignment rules for pairs and projections are correct under Definition 12.4.2 and that $\pi_i(\langle P_1, P_2\rangle) \twoheadrightarrow_\beta P_i$.

12.7.5. EXERCISE. Show that the type assignment rules for injections and case elimination are correct under Definition 12.4.3 and that beta-reduction is properly implemented, i.e.,

$$\mathrm{case}(\mathrm{in}_i(N^{\tau_i});\, x_1^{\tau_1}.M_1^{\rho};\, x_2^{\tau_2}.M_2^{\rho}) \twoheadrightarrow_\beta M_i[x_i := N].$$

12.7.6. Exercise. Consider the eta reduction rule for disjunction, and the commuting conversions of Definition 7.23. Are they correct with respect to Definition 12.4.3?

12.7.7. Exercise. Define an interpretetion in System **F** for the types of:

- words over a fixed finite alphabet;
- finite binary trees;
- lists over a type τ,

with the appropriate basic operations on these types.

12.7.8. Exercise. Prove Proposition 12.4.4. Can you strenghten it by replacing $=_\beta$ by $\twoheadrightarrow_\beta$?

12.7.9. Exercise. Show that the definitions of **abstype** and **pack** in System **F** satisfy the beta reduction rule for existential types.

12.7.10. Exercise. Consider a Curry-style calculus with existential types, analogous to the Curry-style calculus with first-order existential quantification, as in Section 10.5. Show that it does not have the subject reduction property.

12.7.11. Exercise. Verify the correctness of Example 12.5.4.

12.7.12. Exercise. Show that all terms in normal form are typable in System **F**.

12.7.13. Exercise. Show that $(\lambda x.\, xx)(\lambda z.\, zyz)$ is typable in System **F**.

12.7.14. Exercise. Show an example of a pure lambda-term M, typable in System **F**, and such that $M \to_\eta M'$, for some untypable M'. **Warning:** An example known to the authors is the term

$$\lambda a.\, [\lambda yz.\, a((\lambda v.\, v(\lambda xy.\, yay))y)(zy)(z(\lambda x.\, xxx))]YZ,$$

where Y is $\lambda x.\, \mathbf{K}(xx)(xaa)$ and Z is $\lambda u.\, u(\lambda xy.\, a)$. A verification takes 5 handwritten pages.

12.7.15. Exercise. Prove that $(\lambda x.\, xx)(\lambda x.\, xx)$ is untypable in System **F**, without using the fact that typable terms can be normalized.

12.7.16. Exercise. Prove that the terms $c_2c_2\mathbf{K}$ and $(\lambda zy.\, y(z\mathbf{I})(z\mathbf{K}))(\lambda x.\, xx)$ are untypable in System **F**.

12.7.17. Exercise. Show that the polymorphic type assignment has no principal type property. (For instance show that $\lambda x.\, xx$ has no principal type.)

12.7.18. Exercise. Assume strong normalization for Curry-style System **F**. Derive strong normalization for Church-style System **F**.

CHAPTER 13

The λ-cube and pure type systems

In this chapter we introduce Barendregt's λ-*cube* which can be seen as a generic typed λ-calculus with eight instances. Among these are $\lambda\!\to$, $\lambda 2$, and λP seen earlier. We also present the generalization to *pure type systems* which can be seen as a generic typed λ-calculus with infinitely many typed λ-calculi as instances—among these are the eight systems of the λ-cube.

We first present some motivation for the study of the λ-cube and then proceed to the actual definitions. A number of examples are given to illustrate various aspects of the λ-cube. We then argue that the new presentations of $\lambda\!\to$ etc. are equivalent to the previous presentations, although we do not settle this in all detail.

After this, we introduce pure type systems, give a number of examples, and develop the most rudimentary theory of pure type systems.

The presentation follows [8].

13.1. Introduction

The previous chapters of the notes have presented several related typed λ-calculi, e.g, the simply typed λ-calculus $\lambda\!\to$, the system λP of dependent types, and the second-order typed λ-calculus $\lambda 2$.[1]

The simplest of the systems is clearly simply typed λ-calculus; in this system one can have a term $\lambda x{:}\sigma\,.\,x$ of type $\sigma \to \sigma$, in short

$$\vdash \lambda x{:}\sigma\,.\,x : \sigma \to \sigma.$$

Given a term M, we may form the new term $\lambda x : \sigma\,.\,M$ which expects a term as argument; in other words, the term $\lambda x{:}\sigma\,.\,M$ *depends on a term.* Thus, in $\lambda\!\to$, terms may depend on terms.

[1]In this chapter we are exclusively concerned with typed λ-calculi à la Church.

In $\lambda 2$ one can have a term[2] $\Lambda\alpha{:}*.\ \lambda x{:}\alpha\ .\ x$ of type $\forall\alpha.\alpha \to \alpha$, in short

$$\vdash \Lambda\alpha{:}*.\ \lambda x{:}\alpha\ .\ x : \forall\alpha.\alpha \to \alpha.$$

Given a term M, we may form the new term $\Lambda\alpha{:}*.\ M$ which expects a type σ as argument; in other words, the term $\Lambda\alpha{:}*.\ M$ *depends on a type.* Thus, in $\lambda 2$ terms may depend on types.

Finally, in the system λP, we can have an expression $\lambda x{:}\alpha\ .\ \alpha$ expecting a term of type α as argument and returning the type α as result. Such an expression is called a *type constructor* since it constructs a type when provided with suitable arguments. To describe the input and output of a term one uses types; a term *has* a certain type. Similarly, the input and output of constructors are described by *kinds.* For instance, if α is a type, then $\lambda x{:}\alpha\ .\ \alpha$ has kind $\alpha \to *$, in short

$$\alpha : * \vdash \lambda x{:}\alpha\ .\ \alpha : \alpha \Rightarrow *,$$

expressing the fact that the constructor expects a term of type α and constructs a member of $*$, i.e., a type. If we apply this expression to a term of type α, then we get a type:

$$\alpha : *, y : \alpha \vdash (\lambda x{:}\alpha\ .\ \alpha)\ y : *.$$

In conclusion, given a type α we may form the type constructor $\lambda x{:}\alpha\ .\ \alpha$ which expects a term of type α as argument; in other words, the type constructor $\lambda x : \alpha\ .\ \alpha$ *depends on a term.* Thus, in λP types may depend on terms.

There is one combination we have not mentioned: we have seen terms depending on terms, terms depending on types and types depending on terms—what about types depending on types? Can we have expressions like $\lambda\alpha : *\ .\ \alpha \to \alpha$? This would again be a type constructor, but where the argument is now a type rather than a term; such an expression would have kind $* \to *$. In fact, a system exists in which such expressions may be formed, known as $\lambda\underline{\omega}$.

The three systems $\lambda 2$, λP, and $\lambda\underline{\omega}$ each arise from $\lambda{\to}$ by adding another type of dependency than terms depending on terms; it is natural to study also the combination of these dependencies, and systems exist in which several dependencies are allowed, e.g., the system $\lambda\omega$ in which both terms depending on types and types depending on types are allowed, and this will also be done below.

Before proceeding with the details of such systems, it will be well worth our effort to reconsider the style or presentation from previous chapters. In all the λ-calculi presented in previous chapters, one or more forms of

[2]The annotation ": $*$" was not written in the preceding chapter. The informal meaning of $\sigma : *$ is that σ is a type.

abstraction are present, and different rules govern the typing of the various forms of abstraction. For instance, in $\lambda 2$ we have both

$$\frac{\Gamma, x : \sigma \ \vdash \ M : \tau}{\Gamma \ \vdash \ \lambda x{:}\sigma \,.\, M : \sigma \to \tau}$$

and

$$\frac{\Gamma \ \vdash \ M : \tau}{\Gamma \ \vdash \ \Lambda\alpha{:}\,* \,.\, M : \forall\alpha.\tau}$$

Similarly, we have several types of products, e.g. $\sigma \to \tau$ and $\forall\alpha.\sigma$.

It would be better if we could present the systems with a single notion of abstraction parametrized over the permitted dependencies. This is exactly what one does on the λ-cube.

This idea facilitates a more slick presentation of each of the systems in the λ-cube, and also makes the connection between the various systems more transparent. Moreover, properties about the various systems can be developed once and for all by providing proofs that are parametric in the permitted dependencies: if we can prove, e.g., subject reduction regardless of which dependencies are permitted, then we do not have to prove the property for all the systems individually.

13.2. Barendregt's λ-cube

We introduce Barendregt's λ-cube following [8].

13.2.1. Definition.

(i) The set $\mathcal{S}$ of *sorts* is defined by $\mathcal{S} = \{*, \square\}$. We use s, s', s_1, s_2, etc. to range over sorts.

(ii) For each $s \in \mathcal{S}$, let $\mathcal{V}_s$ denote a countably infinite set of *variables* such that $\mathcal{V}_s \cap \mathcal{V}_{s'} = \emptyset$ when $s \neq s'$, and let $\mathcal{V} = \bigcup_{s\in\mathcal{S}} \mathcal{V}_s$.

(iii) The set $\mathcal{E}$ of *expressions* is given by the abstract syntax:

$$\mathcal{E} = \mathcal{V} \mid \mathcal{S} \mid \mathcal{E}\mathcal{E} \mid \lambda\mathcal{V} : \mathcal{E}.\mathcal{E} \mid \Pi\mathcal{V} : \mathcal{E}.\mathcal{E}$$

As before, we assume familiarity with the *subexpression* relation $\subseteq$, with the set $\mathrm{FV}(M)$ of *free variables* of M, and with substitution $M\{x := N\}$ for $x \in \mathcal{V}$ and $M, N \in \mathcal{E}$. We write $A \to B$ for $\Pi d{:}\,A.B$ when $d \notin \mathrm{FV}(B)$. We use $=$ to denote syntactic identity modulo α-conversion and adopt the usual hygiene conventions.

(iv) The relation $\to_\beta$ on $\mathcal{E}$ is the compatible closure of the rule

$$(\lambda x{:}A \,.\, M)\ N \quad \beta \quad M\{x := N\}$$

Also, $\twoheadrightarrow_\beta$ and $=_\beta$ are the transitive, reflexive closure and the transitive, reflexive, symmetric closure of $\to_\beta$, respectively.

(v) The set $\mathcal{C}$ of *contexts* is the set of all sequences

$$x_1\!:\!A_1, \dots, x_n\!:\!A_n$$

where $x_1, \dots, x_n \in \mathcal{V}$, $A_1, \dots, A_n \in \mathcal{E}$, and $x_i \neq x_j$ when $i \neq j$. The empty sequence is $[]$, and the concatenation of Γ and Δ is Γ, Δ. We write $x : A \in \Gamma$ if $\Gamma = \Gamma_1, x : A, \Gamma_2$, for some Γ_1, Γ_2, and we write $\Gamma \subseteq \Delta$ if, for every $x : A \in \Gamma$, also $x : A \in \Delta$. For $\Gamma \in \mathcal{C}$, $\mathrm{dom}(\Gamma) = \{x \mid x\!:\!A \in \Gamma, \text{ for some } A\}$.

(vi) For any set $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ of *rules,* the relation $\vdash\ \subseteq \mathcal{C} \times \mathcal{E} \times \mathcal{E}$ is defined in Figure 13.1. If $\Gamma \vdash M : A$, then Γ is *legal* and M, A are *legal* (in Γ). We use the notation $\Gamma \vdash A : B : C$ meaning that $\Gamma \vdash A : B$ and $\Gamma \vdash B : C$.

$$
\begin{array}{lll}
\text{(axiom)} & [] \vdash * : \Box & \\[2ex]
\text{(start)} & \dfrac{\Gamma \vdash A : s}{\Gamma, x\!:\!A \vdash x : A} & \text{if } x \in \mathcal{V}_s \ \&\ x \notin \mathrm{dom}(\Gamma) \\[2ex]
\text{(weakening)} & \dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x\!:\!C \vdash A : B} & \text{if } x \in \mathcal{V}_s \ \&\ x \notin \mathrm{dom}(\Gamma) \\[2ex]
\text{(product)} & \dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x\!:\!A \vdash B : s_2}{\Gamma \vdash (\Pi x\!:A.\, B) : s_2} & \text{if } (s_1, s_2) \in \mathcal{R} \\[2ex]
\text{(application)} & \dfrac{\Gamma \vdash F : (\Pi x\!:A.\, B) \quad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B\{x := a\}} & \\[2ex]
\text{(abstraction)} & \dfrac{\Gamma, x\!:\!A \vdash b : B \quad \Gamma \vdash (\Pi x\!:A.\, B) : s}{\Gamma \vdash \lambda x\!:\!A\,.\, b : \Pi x\!:A.\, B} & \\[2ex]
\text{(conversion)} & \dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} & \text{if } B =_\beta B'
\end{array}
$$

Figure 13.1: Inference rules of the λ-cube

(vii) The λ-*cube* consists of the eight λ-calculi obtained by taking the different sets of rules $\{(*, *)\} \subseteq \mathcal{R} \subseteq \{(*, *), (\Box, *), (*, \Box), (\Box, \Box)\}$ specified in the table in Figure 13.2.

$\lambda\!\to$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
$\lambda\underline{\omega}$	$(*,*)$		$(\Box,\Box)$	
$\lambda\omega = \lambda\underline{\omega}2$	$(*,*)$	$(\Box,*)$	$(\Box,\Box)$	
λP	$(*,*)$			$(*,\Box)$
$\lambda P2$	$(*,*)$	$(\Box,*)$		$(*,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(\Box,\Box)$	$(*,\Box)$
$\lambda C = \lambda P\omega$	$(*,*)$	$(\Box,*)$	$(\Box,\Box)$	$(*,\Box)$

Figure 13.2: Rules of the λ-cube

13.2.2. Remark. The different rules in Figure 13.2 correspond to the dependencies mentioned earlier:

$(*,*)$:	terms depending on terms
$(*,\Box)$:	terms depending on types
$(\Box,*)$:	types depending on terms
$(\Box,\Box)$:	types depending on types

Informally speaking, terms may also depend on types in $\lambda\!\to$; for instance, $\lambda x:\alpha\ .\ x$ is a term depending on the type α. However, the crucial point is that we may not abstract over α in the term. In contract, this form of abstraction is permitted in $\lambda 2$.

The λ-cube is depicted diagrammatically in Figure 13.3.

$$\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda C \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda P2 \arrow[ur] & \\
 & \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda P\underline{\omega} \arrow[uu] \\
\lambda\!\to \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda P \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

Figure 13.3: The λ-cube

13.3. Example derivations

There are several subtle details in the previous definition. Some of these are illustrated in the following examples, taken from [8], which the reader is encouraged to work out in detail.

13.3.1. Example. In $\lambda\!\!\to$ one can derive the following.

1. $\alpha : * \vdash \Pi x{:}\alpha.\,\alpha : *$. Being a member of $*$ should be conceived as being a type, so if α is a type, then so is $\Pi x{:}\alpha.\,\alpha$.

   Here α is a variable from $\mathcal{V}_\Box$ (type variables) and x is another variable from $\mathcal{V}_*$ (term variables). Hence $x \notin \mathrm{FV}(\alpha)$, so $\Pi x{:}\,\alpha.\,\alpha = \alpha \to \alpha$, using the abbreviation from Definition 13.2.1(iii).

   This means that we may form expressions that map terms of type α to terms of type α (terms may depend on terms).

   Note that the type variable α must be declared in the context; the only typing in the empty context in this formulation of $\lambda\!\!\to$ is $\vdash * : \Box$!

2. $\alpha : * \vdash \lambda x : \alpha\ .\ x : \Pi x{:}\,\alpha.\,\alpha$. Using the abbreviation from Definition 13.2.1(iii), this means that we have $\alpha : * \vdash \lambda x{:}\alpha\ .\ x : \alpha \to \alpha$.

   Note that $\alpha : * \vdash \lambda x : \alpha\ .\ x : \Pi x{:}\,\alpha.\,\alpha$ does not follow from the mere fact that $\alpha : *, x : \alpha \vdash x : \alpha$ alone; we also have to show that $\alpha : * \vdash \Pi x{:}\alpha.\alpha : *$, which can be derived using the fact that $(*, *) \in \mathcal{R}$.

3. $\alpha : *, \beta : *, y : \beta \vdash \lambda x{:}\alpha\ .\ y : \Pi x{:}\alpha.\,\beta$.

4. $\alpha : *, y : \alpha \vdash (\lambda x{:}\alpha\ .\ x)\ y : \alpha$.

5. $\alpha : *, y : \alpha \vdash y : \alpha$. We have

$$(\lambda x{:}\alpha\ .\ x)\ y \quad \to_\beta \quad y.$$

   Note that typing is preserved under this particular reduction.

6. $\alpha : *, \beta : *, x : \alpha, y : \beta \vdash (\lambda z : \alpha\ .\ y)\ x : \beta$. Note how the weakening rule is required to establish this.

7. $\alpha : *, \beta : * \vdash \lambda x{:}\alpha\ .\ \lambda y{:}\beta\ .\ y : \alpha \to \beta \to \alpha : *$.

13.3.2. Example. In $\lambda 2$ one can derive the following.

1. $\alpha : * \vdash \lambda x{:}\alpha\ .\ x : \alpha \to \alpha$, just like in $\lambda\!\!\to$.

2. $\vdash \lambda\alpha{:}*\ .\ \lambda x{:}\alpha\ .\ x : \Pi\alpha{:}*.\ \alpha \to \alpha : *$.

   To understand the relationship with the previous formulation of $\lambda 2$, the reader should realize that $\Pi\alpha{:}\,*.\ \sigma$ is just a new way of writing $\forall\alpha.\sigma$ and that $\lambda\alpha{:}*\ .\ M$ is just a new way of writing $\Lambda\alpha{:}*.\ M$.

This means that we may form expressions that map any type α to a term of type $\alpha \to \alpha$ (terms may depend on types).

The reader may be worried about the absence of the side condition "where α is not free in the context" in the abstraction rule for second-order generalization. How can we be sure that we do not move from $\alpha : *, x : \alpha \vdash M : \sigma$ to $x : \alpha \vdash \lambda\alpha : * \,.\, M : \Pi\alpha{:}\,*.\,\sigma$, where we generalize over a variable which is free in the context? Interestingly, this is ensured by the fact that contexts are *sequences* and that we may discharge the *rightmost* assumption only.

This also shows why the weakening rule is necessary: without weakening, we could not derive something as elementary as $\alpha : *, x : \alpha \vdash \alpha : *$.

3. $\beta : * \vdash (\lambda\alpha{:}* \,.\, \lambda x{:}\alpha \,.\, x)\; \beta : \beta \to \beta$.

4. $\beta : *, y : \beta \vdash (\lambda\alpha{:}* \,.\, \lambda x{:}\alpha \,.\, x)\; \beta\; y : \beta$. We have

$$(\lambda\alpha{:}* \,.\, \lambda x{:}\alpha \,.\, x)\; \beta\; y \quad \to_\beta \quad (\lambda x{:}\beta \,.\, x)\; y \quad \to_\beta \quad y$$

and each of these terms has type β.

5. Let $\bot = \Pi\alpha{:}*.\alpha$. Then $\vdash (\lambda\beta{:}* \,.\, \lambda x{:}\bot \,.\, x\; \beta) : \Pi\beta{:}*.\bot \to \beta$. Via the Curry-Howard isomorphism this type is a proposition in second-order propositional logic which states that everything follows from absurdity; the term is the constructive proof of the proposition.

13.3.3. Example. Let $D = \lambda\beta : * \,.\, \beta \to \beta$. In $\lambda\underline{\omega}$ one can derive the following.

1. $\vdash * \to * : \square$. Being a member of $\square$ should be conceived as being a kind, so $* \to *$ is a kind.

   This means that we may form expressions that map types to types (types may depend on types).

2. $\vdash D : * \to *$.

3. $\alpha : * \vdash D\; \alpha : *$.

4. $\alpha : * \vdash D\; (D\; \alpha) : *$.

5. $\alpha : * \vdash \lambda x{:}(D\; \alpha) \,.\, x : (D\; \alpha) \to (D\; \alpha)$.

6. $\alpha : * \vdash \lambda x{:}(D\; \alpha) \,.\, x : D\; (D\; \alpha)$. Note how the conversion rule is used to derive this. We have

$$(D\; \alpha) \to (D\; \alpha) =_\beta D\; (D\; \alpha)$$

and the conversion rule allows us to exchange the first type with the second.

13.3.4. Example. In λP one can derive the following.

1. $\alpha : * \vdash \alpha \to * : \Box$.

   This means that we may form expressions that map a term of type α to a type (types may depend on terms).

   If we view α as a set and $*$ as the universe of propositions, then a member of $\alpha \to *$ is a map from α to propositions, i.e., a predicate on α.

2. $\alpha : *, p : \alpha \to *, x : \alpha \vdash p\ x : *$. Again, if we view α as a set, p as a predicate on α, and x as an element of α, then $p\ x$ is a proposition.

3. $\alpha : *, p : \alpha \to \alpha \to * \vdash \Pi x{:}\,\alpha.\, p\ x\ x : *$.

   If α is a set and p is a binary predicate on α, then $\Pi x{:}\,\alpha.\, p\ x\ x$ is a proposition.

4. $\alpha : *, p : \alpha \to *, q : \alpha \to * \vdash \Pi x{:}\,\alpha.\, p\ x \to q\ x : *$. This proposition states that every member of α which satisfies p also satisfies q.

5. $\alpha : *, p : \alpha \to * \vdash \lambda x{:}\alpha\ .\ \lambda z{:}p\ x\ .\ z : \Pi x{:}\,\alpha.\, p\ x \to p\ x$.

6. $\alpha : *, x_0 : \alpha, p : \alpha \to *, q : * \vdash$
   $$\lambda z{:}(\Pi x{:}\,\alpha.\, p\ x \to q)\ .\ \lambda y{:}(\Pi x{:}\,\alpha.\, p\ x)\ .\ (x\ a_0)\ (y\ a_0) :$$
   $$(\Pi x{:}\,\alpha.\, p\ x \to q) \to (\Pi x{:}\,\alpha.\, p\ x) \to q.$$
   This proposition states that in every non-empty set A,
   $$(\forall x \in A : P\ x \to Q) \to (\forall x \in A : P\ x) \to Q.$$

13.3.5. Example. In $\lambda\omega$, the following can be derived.

1. Let $\alpha\&\beta = \Pi\gamma{:}\,*.\ (\alpha \to \beta \to \gamma) \to \gamma$. Then $\alpha : *, \beta : * \vdash \alpha\&\beta : *$. This is can also be derived in $\lambda 2$; it is the second-order definition of conjunction.

2. Let $\mathrm{AND} = \lambda\alpha : *\ .\ \lambda\beta : *\ .\ \alpha\&\beta$. Then $\vdash \mathrm{AND} : * \to * \to *$ and $\vdash \mathrm{AND}\,\alpha\,\beta : *$. Thus, AND is a uniform definition of conjunction over all types—this cannot be done in $\lambda 2$.

3. Let $\pi_i = \lambda\alpha_1{:}*\ .\ \lambda\alpha_2{:}*\ .\ \lambda x_1{:}\alpha_1\ .\ \lambda x_2{:}\alpha_2\ .\ x_i$. Then
   $$\vdash \pi_i : \Pi\alpha_1{:}*.\ \Pi\alpha_2{:}*.\ \alpha_1 \to \alpha_2 \to \alpha_i.$$
   Also,
   $$\alpha : *, \beta : * \vdash \lambda x{:}\mathrm{AND}\,\alpha\,\beta\ .\ x\ \alpha\ (\pi_1\ \alpha\ \beta) : \mathrm{AND}\ \alpha\ \beta \to \alpha.$$

13.3.6. EXAMPLE. In $\lambda P2$ the following can be derived.

1. $\alpha : *, p : \alpha \to * \vdash \lambda x{:}\alpha \,.\, p\ x \to \bot : \alpha \to *$ Here the construction of $\bot$ requires $\lambda 2$ and the construction of expressions $\alpha \to *$ requires λP.

2. $\alpha : *, p : \alpha \to \alpha * \vdash (\Pi x{:}\alpha.\ \Pi y{:}\alpha.\ p\ x\ y \to p\ y\ x \to \bot) \to \Pi z{:}\alpha.\ p\ z\ z \to \bot : *$. This proposition states that an asymmetric binary relation is irreflexive.

13.3.7. EXAMPLE. In $\lambda P\underline{\omega}$ the following can be derived.

1. $\alpha : * \vdash \lambda p{:}\alpha \to \alpha \to * \,.\, \lambda x{:}\alpha \,.\, p\ x\ x : (\alpha \to \alpha \to *) \to (\alpha \to *)$.

   This constructor maps any predicate to its own "diagonalization."

   Here the presence of $*$ to the left and right of $\to$ requires $\lambda\underline{\omega}$, and the construction of $\alpha \to *$ requires λP.

2. $\vdash \lambda\alpha{:}* \,.\, \lambda p{:}\alpha \to \alpha \to * \,.\, \lambda x{:}\alpha \,.\, pxx : \Pi\alpha{:}*.(\alpha \to \alpha \to *) \to (\alpha \to *)$. This constructor does the same uniformly in α.

13.3.8. EXAMPLE. In λC the following can be derived.

1. $\lambda\alpha{:}* \,.\, \lambda p{:}\alpha \to \bot \,.\, \lambda x{:}\alpha \,.\, p\ x \to \bot : \Pi\alpha{:}*.\ (\alpha \to *) \to (\alpha \to *)$. This constructor maps a type α and a predicate p on α to the negation of p.

   Here the presence of $*$ on both sides of $\to$ requires $\lambda\underline{\omega}$, and $A \to *$ requires λP, and $\Pi\alpha{:}*.\ \ldots$ requires $\lambda 2$.

13.4. Classification and equivalence with previous formulations

The above presentation of $\lambda\!\to$ etc. has several advantages, as mentioned above. However, the presentation also has the disadvantage that in the case of some of the simple systems (e.g. $\lambda\!\to$) the presentation involves a certain amount of redundancy. For instance, in case of $\lambda\!\to$ the product $\Pi x{:}A.\ B$ mentioned in the product rule always has form $A \to B$. Also, the distinction between terms, types, and kinds, which is abandoned in the cube is, after all, useful for understanding the details of the various systems.

In this section we therefore try to recover some of what has been lost by showing that—to a certain extent—the traditional formulations of some of the systems in the λ-cube are equivalent with the formulations in terms of the λ-cube. More about this can be found in [12, 8, 41].

Below we show how terms belonging to systems in the λ-cube can be classified according to the notions of *object*, *constructors*, and *kinds*.

13.4.1. Definition. Define the sets $\mathcal{O}, \mathcal{T}, \mathcal{K}$ of *objects, constructors, and kinds* as follows:

$$\begin{array}{lcll}
\mathcal{O} & ::= & \mathcal{V}_* & \mid \lambda\mathcal{V}_*{:}\mathcal{T}\,.\,\mathcal{O} \mid \mathcal{O}\;\mathcal{O} \mid \lambda\mathcal{V}_\Box{:}\mathcal{K}\,.\,\mathcal{O} \mid \mathcal{O}\;\mathcal{T} \\
\mathcal{T} & ::= & \mathcal{V}_\Box & \mid \lambda\mathcal{V}_*{:}\mathcal{T}\,.\,\mathcal{T} \mid \mathcal{T}\;\mathcal{O} \mid \lambda\mathcal{V}_\Box{:}\mathcal{K}\,.\,\mathcal{T} \mid \mathcal{T}\;\mathcal{T} \mid \Pi\mathcal{V}_*{:}\mathcal{T}.\,\mathcal{T} \mid \Pi\mathcal{V}_\Box{:}\mathcal{K}.\,\mathcal{T} \\
\mathcal{K} & ::= & \Pi\mathcal{V}_*{:}\mathcal{T}.\,\mathcal{K} \mid \Pi\mathcal{V}_\Box{:}\mathcal{K}.\,\mathcal{K} \mid * &
\end{array}$$

Reading objects as terms, constructors as types, and kinds as—well—kinds, this crystalizes the four forms of dependencies between terms and types expressed by the four forms of abstractions, or the four forms of products.

The following selects among the above expressions those that are legal. In the definition and the following proposition, $\vdash$ refers to λC.

13.4.2. Definition. Define the sets $\mathcal{O}^+, \mathcal{T}^+, \mathcal{K}^+$ as follows:

$$\begin{array}{rcl}
\mathcal{O}^+ & = & \{O \in \mathcal{O} \mid \exists\Gamma, A\,.\,\Gamma \vdash O : A\} \\
\mathcal{T}^+ & = & \{T \in \mathcal{T} \mid \exists\Gamma, A\,.\,\Gamma \vdash T : A\} \\
\mathcal{K}^+ & = & \{K \in \mathcal{K} \mid \exists\Gamma\,.\,\Gamma \vdash K : \Box\}
\end{array}$$

13.4.3. Proposition (Classification of the λ-cube).

1. *The sets $\mathcal{O}^+, \mathcal{T}^+, \mathcal{K}^+$ and $\{\Box\}$ are pairwise disjoint and closed under reduction.*

2. *If $\Gamma \vdash A : B$ then exactly one of the following holds:*
   - $(A, B) \in \mathcal{O}^+ \times \mathcal{T}^+$, *or*
   - $(A, B) \in \mathcal{T}^+ \times \mathcal{K}^+$, *or*
   - $(A, B) \in \mathcal{K}^+ \times \{\Box\}$

One obtains similar classifications for particular systems within the λ-cube:

1. For $\lambda\!\to$,$\lambda 2$,$\lambda\omega$, and $\lambda\underline{\omega}$ omit the clauses $\lambda V^* : \mathcal{T}\ .\ \mathcal{T}$, $\mathcal{T}\ \mathcal{O}$, and $\Pi V^*{:}\mathcal{T}.\,\mathcal{K}$;

2. For $\lambda\!\to$,$\lambda 2$,$\lambda P2$, and λP omit the clauses $\lambda V^\Box : \mathcal{K}\ .\ \mathcal{T}$, $\mathcal{T}\ \mathcal{T}$, and $\Pi V^\Box{:}\mathcal{K}.\,\mathcal{K}$;

3. For $\lambda\!\to$,$\lambda\underline{\omega}$,$\lambda P\underline{\omega}$, and λP omit the clauses $\lambda V^\Box : \mathcal{K}\ .\ \mathcal{O}$, $\mathcal{O}\ \mathcal{T}$, and $\Pi V^\Box{:}\mathcal{K}.\,\mathcal{T}$.

One can use this to show, e.g., that in $\lambda\!\to$ products $\Pi x{:}\alpha.\,\beta$ in fact always have form $\alpha \to \beta$, i.e., $x \notin \mathrm{FV}(\alpha)$.

13.5. Pure type systems

In this subsection we introduce *pure type systems,* as presented by Barendregt, Geuvers, and Nederhof [8, 42, 41]. The main ideas in the step from the λ-cube to pure type systems are the following:

1. One takes a set $\mathcal{S}$ as the set of sorts rather than just $\{*, \Box\}$.
2. One takes a relation $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ as the set of axioms rather than the single axiom $(* : \Box)$.
3. One takes a relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ as the set of rules rather than the specific sets of rules from Figure 13.2.
4. The product rule is generalized so that products need not live in the same universe as their range. That is, $A \to B$ does not necessarily live in the same sort as B.[3]

13.5.1. Definition. A *pure type system* (*PTS*) is a triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ where

(i) $\mathcal{S}$ is a set of *sorts*;

(ii) $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of *axioms*;

(iii) $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ is a set of *rules.*

We write $(s, s') \in \mathcal{R}$ for $(s, s', s') \in \mathcal{R}$.

13.5.2. Definition. Let $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ be a PTS.

(i) For each $s \in \mathcal{S}$, let $\mathcal{V}_s$ denote a countably infinite set of *variables* such that $\mathcal{V}_s \cap \mathcal{V}_{s'} = \emptyset$ when $s \neq s'$, and let $\mathcal{V} = \bigcup_{s \in \mathcal{S}} \mathcal{V}_s$.

(ii) The set $\mathcal{E}$ of *expressions* is given by the abstract syntax:

$$\mathcal{E} = \mathcal{V} \mid \mathcal{S} \mid \mathcal{E}\mathcal{E} \mid \lambda\mathcal{V} : \mathcal{E}.\mathcal{E} \mid \Pi\mathcal{V} : \mathcal{E}.\mathcal{E}$$

We assume familiarity with the *subexpression* relation $\subseteq$, with the set $\mathrm{FV}(M)$ of *free variables* of M, and with substitution $M\{x := N\}$ for $x \in \mathcal{V}$ and $M, N \in \mathcal{E}$. We write $A \to B$ for $\Pi d{:}\, A.\, B$ when $d \notin \mathrm{FV}(B)$. We use $=$ to denote syntactic identity modulo α-conversion and adopt the usual hygiene conventions—see [7].

(iii) The relation $\to_\beta$ on $\mathcal{E}$ is smallest compatible relation satisfying

$$(\lambda x{:}A\, .\, M)\; N \qquad \to_\beta \qquad M\{x := N\}$$

Also, $\twoheadrightarrow_\beta$ and $=_\beta$ are the transitive, reflexive closure and the transitive, reflexive, symmetric closure of $\to_\beta$, respectively.

[3]Notice the difference in the side condition in the product rule in the λ-cube and the product rule in pure type systems (see below): in the latter case we also have to specify s_3—the sort in which the product is to live.

(iv) The set $\mathcal{C}$ of *contexts* is the set of all sequences

$$x_1\!:\!A_1, \dots, x_n\!:\!A_n$$

where $x_1, \dots, x_n \in \mathcal{V}$, $A_1, \dots, A_n \in \mathcal{E}$, and $x_i \neq x_j$ when $i \neq j$. The empty sequence is $[]$, and the concatenation of Γ and Δ is Γ, Δ. We write $x : A \in \Gamma$ if $\Gamma = \Gamma_1, x : A, \Gamma_2$, for some Γ_1, Γ_2, and we write $\Gamma \subseteq \Delta$ if, for every $x : A \in \Gamma$, also $x : A \in \Delta$. For $\Gamma \in \mathcal{C}$, $\mathrm{dom}(\Gamma) = \{x \mid x\!:\!A \in \Gamma, \text{ for some } A\}$.

(v) The relation $\vdash\, \subseteq \mathcal{C} \times \mathcal{E} \times \mathcal{E}$ is defined in Figure 13.4. If $\Gamma \vdash M : A$, then Γ is *legal* and M, A are *legal* (in Γ). We use the notation $\Gamma \vdash A : B : C$ meaning that $\Gamma \vdash A : B$ and $\Gamma \vdash B : C$.

(axiom)	$[] \vdash s_1 : s_2$	if $(s_1, s_2) \in \mathcal{A}$
(start)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x\!:\!A \vdash x : A}$	if $x \in \mathcal{V}_s$ & $x \notin \mathrm{dom}(\Gamma)$
(weakening)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x\!:\!C \vdash A : B}$	if $x \in \mathcal{V}_s$ & $x \notin \mathrm{dom}(\Gamma)$
(product)	$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x\!:\!A \vdash B : s_2}{\Gamma \vdash (\Pi x\!:\!A.\, B) : s_3}$	if $(s_1, s_2, s_3) \in \mathcal{R}$
(application)	$\dfrac{\Gamma \vdash F : (\Pi x\!:\!A.\, B) \quad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B\{x := a\}}$	
(abstraction)	$\dfrac{\Gamma, x\!:\!A \vdash b : B \quad \Gamma \vdash (\Pi x\!:\!A.\, B) : s}{\Gamma \vdash \lambda x\!:\!A\,.\, b : \Pi x\!:\!A.\, B}$	
(conversion)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}$	if $B =_\beta B'$

Figure 13.4: PURE TYPE SYSTEMS

13.5.3. CONVENTION. To save notation we often consider in the remainder a PTS λS and say, e.g., that $s \in \mathcal{S}$ or $M \in \mathcal{E}$ with the understanding that $\lambda S = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ and that $\mathcal{V}$, $\mathcal{E}$, $\mathcal{C}$, $\to_\beta$ and $\vdash$ are defined as in Definition 13.5.2.

13.6. Examples of pure type systems

13.6.1. EXAMPLE. The λ-*cube* consists of the eight PTSs λS, where

(i) $\mathcal{S} = \{*, \Box\}$;

(ii) $\mathcal{A} = \{(*, \Box)\}$;

(iii) $\{(*,*)\} \subseteq \mathcal{R} \subseteq \{(*,*), (\Box,*), (*,\Box), (\Box,\Box)\}$.

The following systems extend $\lambda\omega$ with sort $\triangle$, axiom $\Box : \triangle$, and some rules for the new sort. The system λHOL is defined by:

(i) $\mathcal{S} = \{*, \Box, \triangle\}$;

(ii) $\mathcal{A} = \{(*, \Box), (\Box, \triangle)\}$;

(iii) $\mathcal{R} = \{(*,*), (\Box,*), (\Box,\Box)\}$.

The system λU^- is defined by:

(i) $\mathcal{S} = \{*, \Box, \triangle\}$;

(ii) $\mathcal{A} = \{(*, \Box), (\Box, \triangle)\}$;

(iii) $\mathcal{R} = \{(*,*), (\Box,*), (\Box,\Box), (\triangle,\Box)\}$.

The system λU is defined by:

(i) $\mathcal{S} = \{*, \Box, \triangle\}$;

(ii) $\mathcal{A} = \{(*, \Box), (\Box, \triangle)\}$;

(iii) $\mathcal{R} = \{(*,*), (\Box,*), (\Box,\Box), (\triangle,*), (\triangle,\Box)\}$.

13.6.2. EXAMPLE. The system $\lambda *$ is defined by:

(i) $\mathcal{S} = \{*\}$;

(ii) $\mathcal{A} = \{(*,*)\}$;

(iii) $\mathcal{R} = \{(*,*)\}$.

We end this section with an example of a somewhat different pure type system.

13.6.3. EXAMPLE. The system λPRED is defined by:

(i) $\mathcal{S} = \{*^s, *^p, *^f, \Box^s, \Box^p\}$;

(ii) $\mathcal{A} = \{(*^s, \Box^s), (*^p, \Box^p)\}$;

(iii) $\mathcal{R} = \{(*^p, *^p), (*^s, *^p), (*^s, \Box^p), (*^s, *^s, *^f), (*^s, *^f, *^f)\}$.

This is another formulation of predicate logic as a PTS (other than λP). The significance of the sorts is as follows.

1. The sort $*^s$ is for sets.
2. The sort $*^p$ is for propositions.
3. The sort $*^f$ is for first-order functions from sets to sets.
4. The sorts $\Box^s$ and $\Box^p$ contain $*^s$ and $*^p$ respectively. There is no sort $\Box^f$. This means that we cannot have variables ranging over function spaces.

The significance of the rules is as follows.

1. The rule $(*^p, *^p)$ allows the formation of implication between propositions.
$$\varphi : *^p, \psi : *^p \vdash \varphi \to \psi = \Pi x{:}\varphi.\ \psi : *^p$$
2. The rule $(*^s, *^p)$ allows quantification over sets.
$$\alpha : *^s, \varphi : *^p \vdash \forall x \in \alpha . \varphi = \Pi x{:}\alpha.\ \varphi : *^p$$
3. The rule $(*^s, \Box^p)$ allows the formation of first-order predicates.
$$\alpha : *^s \vdash \alpha \to *^p = \Pi x{:}\alpha.\ *^p : \Box^p$$
so
$$\alpha : *^s, p : \alpha \to *^p, x : \alpha \vdash p\ x : *^p.$$
4. The rule $(*^s, *^s, *^f)$ allows the formation of function spaces between sets.
$$\alpha : *^s, \beta : *^s \vdash \alpha \to \beta : *^f.$$
5. The rule $(*^s, *^f, *^f)$ allows the formation of curried multi-argument function spaces between sets.
$$\alpha : *^s \vdash \alpha \to (\alpha \to \alpha) : *^f.$$

13.7. Properties of pure type systems

As mentioned in the introduction, one of the reasons for introducing the λ-cube is the desire to facilitate proofs which apply to a number of systems at the same time. For instance, it is better to prove the subject reduction property in a generic way which applies to all the systems in the λ-cube, regardless of the particular set $\mathcal{R}$ of rules, than it is to prove the property for each system individually.

This idea is even more compelling in the case of pure type systems: having shown that a property holds for all pure type systems we know not only that the property holds for all the systems of the λ-cube, but also for infinitely many other systems, a number of which have appeared independently in the literature.

In this section we develop the most rudimentary theory of pure type systems. Throughout the section, λS denotes an arbitrary pure type system.

13.7.1. LEMMA (Properties of substitution).

1. $A\{x := B\}\{y := C\} = A\{y := C\}\{x := B\{y := C\}\}$, *if* $y \notin \mathrm{FV}(B)$;
2. $B =_\beta C \;\Rightarrow\; A\{x := B\} =_\beta A\{x := C\}$;
3. $A =_\beta B \;\&\; C =_\beta D \;\Rightarrow\; A\{x := C\} =_\beta B\{x := D\}$.

PROOF. (1)-(2): By induction on A. (3): By induction on $A =_\beta B$, using (1)-(2). □

13.7.2. PROPOSITION (Church-Rosser). *The relation* $\to_\beta$ *on* $\mathcal{E}$ *is confluent.*

PROOF. By the technique of Tait and Martin-Löf—see e.g. [8]. □

13.7.3. LEMMA (Free variables). *If* $x_1{:}A_1, \ldots, x_n{:}A_n \vdash B : C$ *then:*

1. $x_1, \ldots, x_n$ *are distinct;*
2. $\mathrm{FV}(B) \cup \mathrm{FV}(C) \subseteq \{x_1, \ldots, x_n\}$;
3. $\mathrm{FV}(A_i) \subseteq \{x_1, \ldots, x_{i-1}\}$ *for* $1 \leq i \leq n$.

PROOF. By induction on the derivation of $x_1{:}A_1, \ldots, x_n{:}A_n \vdash B : C$. □

13.7.4. LEMMA (Start). *If* Γ *is legal then:*

1. $(s_1, s_2) \in \mathcal{A} \;\Rightarrow\; \Gamma \vdash s_1 : s_2$;
2. $x{:}A \in \Gamma \;\Rightarrow\; \Gamma \vdash x : A$.

PROOF. Since Γ is legal $\Gamma \vdash B : C$ for some B, C. Proceed by induction on the derivation of $\Gamma \vdash B : C$. □

13.7.5. LEMMA (Transitivity). *Let* Δ *be legal. If* $x_1{:}A_1 \ldots, x_n{:}A_n \vdash A : B$ *and* $\Delta \vdash x_i : A_i$ *for* $i = 1, \ldots, n$ *then* $\Delta \vdash A : B$.

PROOF. By induction on the derivation of $x_1 : A_1, \ldots, x_n : A_n \vdash A : B$ making use of the start lemma. □

13.7.6. LEMMA (Substitution). *If* $\Gamma, x{:}A, \Delta \vdash B : C$ *and* $\Gamma \vdash a : A$, *then*[4] $\Gamma, \Delta\{x := a\} \vdash A\{x := a\} : B\{x := a\}$.

[4]Substitution (and any other map) is extended from expressions to contexts in the usual way.

PROOF. By induction on the derivation of $\Gamma, x{:}A, \Delta \vdash B : C$ using the free variables lemma and properties of substitution. □

13.7.7. LEMMA (Thinning). *If* $\Gamma \vdash A : B$, Δ *is legal, and every* $(x{:}A)$ *in* Γ *is also in* Δ, *then* $\Delta \vdash A : B$.

PROOF. This follows from the start lemma and the transitivity lemma. □

13.7.8. LEMMA (Generation). *Suppose that* $\Gamma \vdash M : C$.

1. $M = s \Rightarrow \exists (s, s') \in \mathcal{A}.\ C =_\beta s'$
2. $M = x \Rightarrow \exists D \in \mathcal{E}.\ C =_\beta D\ \&\ x{:}D \in \Gamma$.
3. $M = \lambda x.Ab \Rightarrow \exists s {\in} \mathcal{S},\ B \in \mathcal{E}.\ C =_\beta \Pi x{:}A.\,B\ \&\ \Gamma, x{:}A \vdash b : B\ \&\ \Gamma \vdash \Pi x{:}A.\,B : s$.
4. $M = \Pi x{:}A.\,B \Rightarrow \exists (s_1, s_2, s_3) \in \mathcal{R}.\ C =_\beta s_3\ \&\ \Gamma \vdash A : s_1\ \&\ \Gamma, x : A \vdash B : s_2$.
5. $M = F\,a \Rightarrow \exists x {\in} V,\ A{,}B \in \mathcal{E}.\ C =_\beta B\{x := a\}\ \&\ \Gamma \vdash F : \Pi x{:}A.\,B\ \&\ \Gamma \vdash a : A$.

PROOF. By induction on the derivation of $\Gamma \vdash M : C$. □

13.7.9. LEMMA (Correctness of types). *If* $\Gamma \vdash A : B$ *then either* $B \in \mathcal{S}$ *or* $\exists s \in \mathcal{S}.\ \Gamma \vdash B : s$.

PROOF. By induction on $\Gamma \vdash A : B$, using the generation and substitution lemmas. □

13.7.10. THEOREM (Subject reduction). *If* $\Gamma \vdash A : B$ *and* $A \to_\beta A'$ *then* $\Gamma \vdash A' : B$.

PROOF. Prove by simultaneous induction on the derivation of $\Gamma \vdash A : B$:

1. if $\Gamma \vdash A : B$ and $A \to_\beta A'$ then $\Gamma \vdash A' : B$;
2. if $\Gamma \vdash A : B$ and $\Gamma \to_\beta \Gamma'$ then $\Gamma' \vdash A : B$.

The proof uses the substitution lemma. □

13.8. The Barendregt-Geuvers-Klop conjecture

The following terminology should be well-known by now.

13.8.1. Definition. Let λS be a PTS. A *β-reduction path* from an expression M_0 is a (possibly infinite) sequence $M_0 \to_\beta M_1 \to_\beta M_2 \to_\beta \ldots$ If the sequence is finite, it *ends* in the last expression M_n and has *length* n.

13.8.2. Definition. Let λS be a PTS, and M an expression.

(i) $M \in \infty_\beta \Leftrightarrow$ there is an infinite β-reduction path from M;

(ii) $M \in \mathrm{NF}_\beta \Leftrightarrow$ there is no β-reduction path of length 1 or more from M;

(iii) $M \in \mathrm{SN}_\beta \Leftrightarrow$ all β-reduction paths from M are finite;

(iv) $M \in \mathrm{WN}_\beta \Leftrightarrow$ there is a β-reduction from M ending in $N \in \mathrm{NF}_\beta$.

Elements of $\mathrm{NF}_\beta, \mathrm{SN}_\beta, \mathrm{WN}_\beta$ are *β-normal forms*, *β-strongly normalizing*, and *β-weakly normalizing*, respectively. We also write, e.g., $\infty_\beta(M)$ for $M \in \infty_\beta$.

13.8.3. Definition. λS is *weakly normalizing* if all legal expressions are weakly normalizing, and *strongly normalizing* if all legal expressions are strongly normalizing. In this case we write $\lambda S \models \mathrm{WN}_\beta$ and $\lambda S \models \mathrm{SN}_\beta$, respectively.

13.8.4. Example. All the systems of the λ-cube are strongly normalizing—see, e.g., [12, 8, 42, 41]. The system $\lambda *$ is the simplest PTS which is not strongly normalizing. The system λU is is a natural extension of $\lambda\omega$ which, surprisingly, is not strongly normalizing. This result shows that, apparently, the fact that $\lambda *$ fails to be strongly normalizing is not merely a consequence of the cyclicity in its axiom $(*, *)$.

We conclude the notes by mentioning an open problem in the field—for more on the problem see [101].

13.8.5. Conjecture (Barendregt, Geuvers, Klop). *For every PTS λS:*

$$\lambda S \models \mathrm{WN}_\beta \Rightarrow \lambda S \models \mathrm{SN}_\beta$$

CHAPTER 14

Solutions and hints to selected exercises

Some of the solutions below are based on actual homework done in LaTeX by Henning Niss (1.7.7) and Henning Makholm (6.8.20 and 6.8.21). In one of his homeworks, Henning Makholm has solved an apparently open problem (Exercise 6.8.3).

Exercise 1.7.7

14.0.6. LEMMA (Substitution Lemma). *If $x \neq y$ and $x \notin \mathrm{FV}(L)$ then*

$$M\{x := N\}\{y := L\} = M\{y := L\}\{x := N\{y := L\}\}$$

PROOF. By induction on M.

Case $M = x$. Since $x \neq y$:

$$\begin{array}{rcl} M\{x := N\}\{y := L\} & = & x\{x := N\}\{y := L\} \\ & = & N\{y := L\} \\ & = & x\{x := N\{y := L\}\} \\ & = & x\{y := L\}\{x := N\{y := L\}\} \\ & = & M\{y := L\}\{x := N\{y := L\}\} \end{array}$$

The remaining cases are shown in less detail.

Case $M = y$. Since $x \notin \mathrm{FV}(L)$:

$$\begin{array}{rcl} M\{x := N\}\{y := L\} & = & L \\ & & M\{y := L\}\{x := N\{y := L\}\} \end{array}$$

Case $M = z$ where $z \neq x$ and $z \neq y$. Then:

$$\begin{array}{rcl} M\{x := N\}\{y := L\} & = & z \\ & = & M\{y := L\}\{x := N\{y := L\}\} \end{array}$$

Case $M = \lambda z.P$. Without loss of generality we may assume $z \neq x$ and

$z \neq y$. Then by the induction hypothesis:

$$\begin{array}{rcl} M\{x := N\}\{y := L\} & = & \lambda z.P\{x := N\}\{y := L\} \\ & = & \lambda z.P\{y := L\}\{x := N\{y := L\}\} \\ & = & M\{y := L\}\{x := N\{y := L\}\} \end{array}$$

Case $M = P\ Q$. Similar to the preceding case. □

The following states that $\twoheadrightarrow_\beta$ is *compatible.*

14.0.7. LEMMA. *Assume that $P, P' \in \Lambda$ are such that $P \twoheadrightarrow_\beta P'$. Then, for all $x \in V$ and all $Q \in \Lambda$:*

(i) $\lambda x.P \twoheadrightarrow_\beta \lambda x.P'$;

(ii) $P\ Q \twoheadrightarrow_\beta P'\ Q$;

(iii) $Q\ P \twoheadrightarrow_\beta Q\ P'$.

PROOF. (i): By induction on the derivation of $P \twoheadrightarrow_\beta P'$.

Case $P \twoheadrightarrow_\beta P'$ because $P \rightarrow_\beta P$. Then $\lambda x.P \rightarrow_\beta \lambda x.P'$. Therefore also $\lambda x.P \twoheadrightarrow_\beta \lambda x.P'$.

Case $P \twoheadrightarrow_\beta P'$ because $P \twoheadrightarrow_\beta P''$ and $P'' \twoheadrightarrow_\beta P$. By the induction hypothesis, $\lambda x.P \twoheadrightarrow_\beta \lambda x.P''$ and $\lambda x.P'' \twoheadrightarrow_\beta \lambda x.P'$, so $\lambda x.P \twoheadrightarrow_\beta \lambda x.P'$.

Case $P \twoheadrightarrow_\beta P'$ because $P = P'$. Then $\lambda x.P = \lambda x.P'$. Therefore also $\lambda x.P \twoheadrightarrow_\beta \lambda x.P'$.

(ii)-(iii): induction on the definition of $P \twoheadrightarrow_\beta P'$ □

14.0.8. LEMMA. *For all $P, P', Q \in \Lambda$, if*

$$P \rightarrow_\beta P'$$

then also

$$P\{x := Q\} \rightarrow_\beta P'\{x := Q\}.$$

PROOF. By induction on the derivation of $P \rightarrow_\beta P'$. The interesting case is $P = (\lambda y.P_1)\ Q_1 \rightarrow_\beta P_1\{y := Q_1\} = P'$. We have

$$\begin{array}{rcl} ((\lambda y.P_1)\ Q_1)\{x := Q\} & = & (\lambda y.P_1\{x := Q\})\ (Q_1\{x := Q\}) \\ & \rightarrow_\beta & P_1\{x := Q\}\{y := Q_1\{x := Q\}\} \\ & = & P_1\{y := Q_1\}\{x := Q\}, \end{array}$$

where the last equality follows from the Substitution Lemma (since y is not free in Q). □

14.0.9. Lemma. *For all $P, Q, Q' \in \Lambda$, if*

$$Q \to_\beta Q'$$

then also

$$P\{x := Q\} \twoheadrightarrow_\beta P\{x := Q'\}.$$

Proof. By induction on the structure of P.

Case $P = x$, then $P\{x := Q\} = Q \to_\beta Q' = P\{x := Q'\}$.

Case $P = y$, then $P\{x := Q\} = y \twoheadrightarrow_\beta y = P\{x := Q'\}$.

Case $P = \lambda y.P'$, then

$$\begin{array}{rcl} (\lambda y.P')\{x := Q\} & = & \lambda y.P'\{x := Q\} \\ & \twoheadrightarrow_\beta & \lambda y.P'\{x := Q'\} \\ & = & (\lambda y.P')\{x := Q'\} \end{array}$$

where $\twoheadrightarrow_\beta$ follows from the induction hypothesis and compatibility of $\twoheadrightarrow_\beta$.

Case $P = P_1\ P_2$, then

$$\begin{array}{rcl} (P_1\ P_2)\{x := Q\} & = & (P_1\{x := Q\})\ (P_2\{x := Q\}) \\ & \twoheadrightarrow_\beta & (P_1\{x := Q'\})\ (P_2\{x := Q\}) \\ & \twoheadrightarrow_\beta & (P_1\{x := Q'\})\ (P_2\{x := Q'\}) \\ & = & (P_1\ P_2)\{x := Q'\} \end{array}$$

where both $\twoheadrightarrow_\beta$-steps follow from the induction hypothesis and compatibility of $\twoheadrightarrow_\beta$. □

Now the proposition in the exercise (1.72) can be proved.

14.0.10. Proposition (Weak Church-Rosser). *For all $M_1, M_2, M_3 \in \Lambda$, if $M_1 \to_\beta M_2$ and $M_1 \to_\beta M_3$, then there exists an $M_4 \in \Lambda$ such that $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$.*

Proof. By induction on the derivation of $M_1 \to_\beta M_2$.

Case $M_1 = (\lambda x.P)\ Q$ and $M_2 = P\{x := Q\}$. Then either (1) $M_3 = (\lambda x.P')\ Q$ for P' such that $P \to_\beta P'$, (2) $M_3 = (\lambda x.P)\ Q'$ for Q' such that $Q \to_\beta Q'$, or $M_3 = M_2$. In the last case we are done. In situation (1) we have $M_3 = (\lambda x.P')\ Q \to_\beta P'\{x := Q\}$ and $M_2 = P\{x := Q\} \to_\beta P'\{x := Q\}$ by Lemma 14.0.8, i.e., $M_4 = P'\{x := Q\}$. Situation (2) is similar using Lemma 14.0.9.

(1)

$$\begin{tikzcd} & (\lambda x.P)\ Q \arrow[dl] \arrow[dr] & \\ P\{x := Q\} \arrow[d] & & (\lambda x.P')\ Q \arrow[d] \\ P'\{x := Q\} & = & P'\{x := Q\} \end{tikzcd}$$

(2)

$$\begin{tikzcd} & (\lambda x.P)\ Q \arrow[dl] \arrow[dr] & \\ P\{x := Q\} \arrow[d] & & (\lambda x.P)\ Q' \arrow[d] \\ P\{x := Q'\} & = & P\{x := Q'\} \end{tikzcd}$$

Case $M_1 = \lambda x.P$ and $M_2 = \lambda x.P'$ because $P \;\to_\beta\; P'$. Then M_3 must be $\lambda x.Q$ for $Q \in \Lambda$ such that $P \;\to_\beta\; Q$. By the induction hypothesis there is a term M' such that $P' \twoheadrightarrow_\beta M'$ and $Q \twoheadrightarrow_\beta M'$. Then since $\twoheadrightarrow_\beta$ is compatible we have $M_2 = \lambda x.P' \twoheadrightarrow_\beta \lambda x.M'$ and $M_3 = \lambda x.Q \twoheadrightarrow_\beta \lambda x.M'$, and $M_4 = \lambda x.M'$ is the sought term.

$$\begin{tikzcd}
& \lambda x.P \arrow[dl] \arrow[dr] & \\
\lambda x.P' \arrow[dr] & & \lambda x.Q \arrow[dl] \\
& \lambda x.M' &
\end{tikzcd}$$

Case $M_1 = P\,Q$ and $M_2 = P'\,Q$ because $P \;\to_\beta\; P'$. Then

(1) $M_3 = P''\,Q$ for $P'' \in \Lambda$ such that $P \;\to_\beta\; P''$, or

(2) $M_3 = P\,Q'$ for $Q' \in \Lambda$ such that $Q \;\to_\beta\; Q'$.

In the former case by the induction hypothesis we obtain an M' such that $P' \twoheadrightarrow_\beta M'$ and $P'' \twoheadrightarrow_\beta M'$ and thus (again by compatibility of $\twoheadrightarrow_\beta$) that $M_2 = P'\,Q \twoheadrightarrow_\beta M'\,Q$ and $M_3 = P''\,Q \twoheadrightarrow_\beta M'\,Q$, i.e., $M_4 = M'\,Q$. In the latter case we have that $M_2 = P'\,Q \;\to_\beta\; P'\,Q'$ and $M_3 = P\,Q' \;\to_\beta\; P'\,Q'$ (also by compatibility) and consequently $M_4 = P\,Q'$ as desired.

$$\begin{tikzcd}
& (1) & & & (2) & \\
& P\,Q \arrow[dl] \arrow[dr] & & & P\,Q \arrow[dl] \arrow[dr] & \\
P'\,Q \arrow[dr] & & P''\,Q \arrow[dl] & P'\,Q \arrow[d] & & P\,Q' \arrow[d] \\
& M'\,Q & & P'\,Q' & = & P'\,Q'
\end{tikzcd}$$

This completes the proof. □

The proof does not extend directly to the full Church-Rosser property.

Let us say that a relation $\to$ on a set S is *weakly confluent* if, whenever $s_1 \to s_2$ and $s_1 \to s_3$, there is an s_4 such that $s_2 \to \ldots \to s_4$ and $s_3 \to \ldots \to s_4$. Let us call $\to$ *confluent* if, whenever $s_1 \to \ldots \to s_2$ and $s_1 \to \ldots \to s_3$, there is an s_4 such that $s_2 \to \ldots \to s_4$ and $s_3 \to \ldots \to s_4$.

There are relations that are weakly confluent and not confluent. The simplest example is when we have a four-element set $S = \{s_1, s_2, s_3, s_4\}$ $s_1 \leftarrow s_2$, and $s_2 \to s_3$, $s_2 \leftarrow s_3$, and $s_3 \to s_4$:

$$\begin{tikzcd}
s_1 & s_2 \arrow[l] \arrow[r, shift left=0] & s_3 \arrow[l, shift left=0] \arrow[r] & s_4
\end{tikzcd}$$

A relation $\to$ on a set S is *strongly normalizing* if there are no infinite reduction sequences

$$s_1 \to s_2 \to \ldots$$

of (not necessarily distinct) elements from S. Can you find a weakly confluent, strongly normalizing relation that is not confluent?

A diggression

How many details should a proof contain? Everybody should do a proof in the style of Lemma 14.0.6 (induction on $M \in \Lambda$) and a proof in the style of Lemma 14.0.7 (induction on the definition of $P \twoheadrightarrow_\beta P'$) in every detail at least once in their life.

However, having tried this, one can see that many details are completely mechanical. In choosing a level of detail in a proof, one should leave out details that can be reconstructed mechanically with little effort by the reader. In contrast, steps that require good ideas, even small good ideas, should usually not be left out.

Thus, a complete proof of Lemma 14.0.6 would be "Induction on M" and a complete proof of Lemma 14.0.7 would read "Induction on the definition of $P \twoheadrightarrow_\beta P'$." A complete proof of Lemma 14.0.8 might be "Induction on the definition of $P \to_\beta P'$," but since something interesting happens in the case where $P = (\lambda y.P_1)\ Q_1 \to_\beta P_1\{y := Q_1\} = P'$, one can also present that case and refer to the substitution lemma.

If several cases in a proof are similar, but non-trivial, one can do the first example in detail and omit the remaining ones.

Exercise 1.7.17

14.0.11. Proposition (Klop). *Let $\lambda x_1 x_2 \ldots x_n.M$ be an abbreviation for the λ-term $\lambda x_1.\lambda x_2.\ldots\lambda x_n.M$. Let*

$$\begin{array}{rcl} ? & = & \lambda abcdefghijklmnopqstuvwxyzr.r\ (thisisafixedpointcombinator) \\ \$ & = & ?????????????????????????? \end{array}$$

Then \$ is a fixed point combinator, *i.e., for any $F \in \Lambda$: $\$\ F =_\beta F\ (\$\ F)$.*

Proof. We have:

$$\begin{array}{rcl} \$\ F & = & ??????????????????????????\ F \\ & = & (\lambda abcdefghijklmnopqstuvwxyzr. \\ & & \qquad r\ (thisisafixedpointcombinator)) \\ & & \qquad\qquad ?????????????????????????\ F \\ & =_\beta & F\ (??????????????????????????F) \\ & = & F\ (\$\ F) \end{array}$$

as required. □

Exercise 2.7.2

We only show the closure under weakening. The proof is by induction with respect to the size of the proof of $\Gamma \vdash \varphi$. We proceed by cases depending on the last rule used in this proof. Recall that the notation Γ, ψ stands for $\Gamma \cup \{\psi\}$, whether or not $\psi \in \Gamma$. That's why e.g., Case 3 below works.

Case 1: The proof consists only of a single application of an axiom scheme, that is φ is an element of Γ. Then $\Gamma, \psi \vdash \varphi$ is also an axiom.

Case 2: The proof ends with an application of ($\wedge$I). That is, φ has the form $\varphi_1 \wedge \varphi_2$ and we have proven $\Gamma \vdash \varphi_1$ and $\Gamma \vdash \varphi_2$. The proofs of these judgements are contained in the proof of $\Gamma \vdash \varphi$, so we can apply the induction hypothesis to obtain $\Gamma, \psi \vdash \varphi_1$ and $\Gamma, \psi \vdash \varphi_2$. By an application of ($\wedge$I) we can derive $\Gamma, \psi \vdash \varphi_1 \wedge \varphi_2$.

Case 3: The proof ends with an application of ($\vee$E). That is, we have $\Gamma \vdash \theta_1 \vee \theta_2$, for some formulas θ_1, θ_2, such that $\Gamma, \theta_1 \vdash \varphi$ and $\Gamma, \theta_2 \vdash \varphi$. These proofs are all shorter, thus we can apply the induction hypothesis to obtain $\Gamma, \psi \vdash \theta_1 \vee \theta_2$ and $\Gamma, \theta_1, \psi \vdash \varphi$ and $\Gamma, \theta_2, \psi \vdash \varphi$. It remains to use rule ($\vee$E).

Other cases are similar.

Exercise 2.7.3

1) Begin with the axiom $\bot \vdash \bot$. Apply ($\bot$E), to derive $\bot \vdash \varphi$ and ($\to$I) to derive $\vdash \bot \to \varphi$.

3) Begin with $p, p \to \bot \vdash p$ and $p, p \to \bot \vdash p \to \bot$. Apply ($\to$E) to get $p, p \to \bot \vdash \bot$, then twice ($\to$I) to get $\vdash p \to (p \to \bot) \to \bot$.

5) First show that $\neg\neg\neg p, \neg p, p \vdash \bot$ (unfold the $\neg$'s). Thus, $\neg\neg\neg p, p \vdash \neg p \to \bot$, i.e., $\neg\neg\neg p, p \vdash \neg\neg p$. But $\neg\neg\neg p = \neg\neg p \to \bot$ and one can derive $\neg\neg\neg p, p \vdash \bot$. It remains to use ($\to$I).

7) First show that $p \to q, q \to \bot, p \vdash \bot$. Then apply ($\to$I) three times.

9) What we need is $\neg p \vee \neg q, p \wedge q \vdash \bot$. With help of ($\wedge$E), derive separately $\neg p \vee \neg q, \neg p, p \wedge q \vdash \bot$ and $\neg p \vee \neg q, \neg q, p \wedge q \vdash \bot$. Apply ($\vee$E) to these two judgements.

11) Remember that $\leftrightarrow$ abbreviates a conjunction, so ($\wedge$I) will be the last rule. One part of this proof uses ($\wedge$E), the other one uses ($\wedge$I).

13) First derive $(p \vee \neg p) \to \bot, p \vdash \bot$, using rules ($\vee$E) and ($\to$E). By ($\to$I) obtain $(p \vee \neg p) \to \bot \vdash \neg p$. Then use ($\vee$E) and ($\to$E) again to derive $(p \vee \neg p) \to \bot \vdash \bot$.

Exercise 2.7.4

- First we show that $a \cup a = a$, that is $a \leq a$ holds for all a. This is because $a = 0 \cup a = (-a \cap a) \cup a = (-a \cup a) \cap (a \cup a) = 1 \cap (a \cup a) = a \cup a$.

- The relation $\leq$ is transitive because $a \cup b = b$ and $b \cup c = c$ implies that $c = b \cup c = (a \cup b) \cup c = a \cup (b \cup c) = a \cup c$.

- The antisymmetry ($a \leq b$ and $b \leq a$ implies $a = b$) follows immediately from the definition. To see that $a \cup b$ is the *lub* of a and b, assume that $a \leq c$ and $b \leq c$. Then $(a \cup b) \cup c = a \cup (b \cup c) = a \cup c = c$, i.e., $a \cup b \leq c$.

- The condition $a \cap b \leq a$ (that is $(a \cap b) \cup a = a$) is shown as follows: $(a \cap b) \cup a = (a \cap b) \cup (a \cap 1) = (a \cap b) \cup (a \cap (b \cup -b)) = (a \cap b) \cup (a \cap b) \cup (a \cap -b) = (a \cap b) \cup (a \cap -b) = a \cap (b \cup -b) = a \cap 1 = a$.

- If $a \cap b = a$ then $b = (a \cap b) \cup b = a \cup b$, i.e., $a \leq b$. On the other hand, if $a \leq b$ then $(a \cap b) \cup a = (a \cup a) \cap (b \cup a) = a \cap b$, and thus $a \leq a \cap b$. We conclude $a = a \cap b$, by the previous item and antisymmetry.

Exercise 2.7.5

- The proof that the relation $\leq$ is a partial order, and that $\cup$ is the *lub*, is similar as for Exercise 2.7.4.

- Next we show that $-a \cap a = 0$. We have $-a \leq -a$, i.e., $a \Rightarrow 0 \leq a \Rightarrow 0$. Thus, $a \cap (a \Rightarrow 0) \leq 0$, and since 0 is obviously the least element, we have $a \cap -a = a \cap (a \Rightarrow 0) = 0$.

- Now we prove $(a \cup b) \cap a = a$, in a similar way as we proved the dual law $(a \cap b) \cup a = a$ for a Boolean algebra. (Note that now we use $b \cap -b = 0$ instead of $b \cup -b = 1$.)

- We derive $(a \cap b) \cup a = a$ from the above, because $(a \cap b) \cup a = (a \cup a) \cap (b \cup a) = a \cap (b \cup a)$. Note that we obtain here the idempotency of $\cap$, since $a \cap a = a \cap (a \cup a) = a \cap a$.

- Then we proceed in a similar way as for Boolean algebra.

Exercise 2.7.6

The only property which is not immediate is the equivalence between $A \cap C \subseteq B$ and $C \subseteq \mathrm{Int}(-A \cup B)$. First note that the condition $A \cap C \subseteq B$ is equivalent to $C \subseteq -A \cup B$, for all A, B and C. For the left-to-right implication observe that $X \subseteq Y$ implies $X \subseteq \mathrm{Int}(Y)$, whenever X is an open set. The converse follows from $\mathrm{Int}(-A \cup B) \subseteq -A \cup B$.

Exercise 2.7.8 (1)⇒(2)

Let $\Gamma = \{\vartheta_1, \ldots, \vartheta_n\}$, and let v be a valuation in a Heyting algebra $\mathcal{H}$. We write $v(\Gamma)$ to denote $v(\vartheta_1) \cap \cdots \cap v(\vartheta_n)$. By induction with respect to derivations we prove the following statement: "If $\Gamma \vdash \varphi$ then $v(\Gamma) \leq v(\varphi)$, for all valuations in arbitrary Heyting algebras". The hypothesis follows from the special case when $v(\Gamma) = 1$.

For instance, consider the case of ($\to$I). To show $v(\Gamma) \leq v(\varphi \to \psi)$ recall that $v(\varphi \to \psi) = v(\varphi) \Rightarrow v(\psi)$ and use the induction hypothesis $v(\Gamma) \cap v(\varphi) \leq v(\psi)$. For the case of ($\vee$E) use the distributivity law.

Exercise 2.7.9

First we consider counterexamples with open sets. In what follows we use the convention that $v(p) = P$, $v(q) = Q$, etc, and we write $\sim A$ for $\mathrm{Int}(-A)$.

4) Take P to be the whole $\mathbb{R}^2$ without one point. Then $\sim P$ is empty and $\sim\sim P$ is the full set $\mathbb{R}^2$. Thus $\sim\sim P \Rightarrow P \neq 1$.

6) Let P be an open disk, and Q be $\mathbb{R}^2$ without one straight line crossing the middle of the disk. Then $P \Rightarrow Q$ is the whole space without the intersection of the disk and the line. The value of the right-hand side is the whole space.

8) Take the interiors of two complementary halves of $\mathbb{R}^2$.

10) Take Q and R to be the half of $\mathbb{R}^2$ where $x < 0$, and take $P = \mathbb{R}^2 - \{(0,0)\}$. Then $(P \Rightarrow Q) \cap (Q \Rightarrow P)$ is equal to Q and R. Thus the value of $(p \leftrightarrow q) \leftrightarrow r$ is $\mathbb{R}^2 \not\subseteq P$.

12) Take both P and Q to be the same open disk.

Here are counterexamples in Kripke models.

4) A two-element model with $c < c'$, where $c \nVdash p$ and $c' \Vdash p$.

6) A two-element model with $c < c'$, where $c, c' \Vdash p$ and $c' \Vdash q$ and $c \nVdash q$.

8) A three-element model with $c < c', c''$, where $c' \Vdash p$, $c'' \Vdash q$ and nothing more happens.

10) A three-element model with $c < c' < c''$, where $c' \Vdash p$, $c'' \Vdash p, q, r$ and nothing more happens.

12) A two-element model with $c < c'$, where $c' \Vdash p, q$ and c forces nothing.

Exercise 2.7.10

Use n lines beginning in $(0,0)$ to divide the space $\mathbb{R}^2$ into n disjoint angles and take their interiors as values of p_i. The point $(0,0)$ does not belong to the interpretation of this formula. Or take a Kripke model with all sets $\{c \geq c_0 : c \Vdash p_i\}$ different. Then c_0 does not force our formula.

Exercise 2.7.11

Let F be maximal and let $a \cup b \in F$, but $a, b \notin F$. We show that either $F \cup \{a\}$ or $F \cup \{b\}$ can be extended to a proper filter. First assume that there are $f_1, f_2 \in F$ such that $f_1 \cap a = f_2 \cap b = 0$. Then $(f_1 \cap f_2) \cap (a \cup b) = (f_1 \cap f_2 \cap a) \cup (f_1 \cap f_2 \cap b) = 0 \cup 0 = 0$, a contradiction. Thus either such f_1 or such f_2 does not exist. Assume for instance that $f \cap a \neq 0$, for all $f \in F$. Then the set $F_a = \{x : x \geq f \cap a \text{ for some } f \in F\}$ is a proper filter extending F.

Let F be a prime filter in a Boolean algebra, and assume that $F \subset F'$, with $a \in F' - F$. Since $a \cup -a = 1 \in F$ and F is prime, we have $-a \in F$ and $0 = a \cap -a \in F'$, a contradiction.

Exercise 2.7.12

The argument is similar to the first part of Exercise 2.7.11. Assume that G is our maximal element, and that $c \cup d \in G$, but $c, d \notin G$. The assumption of $g_1, g_2 \in G$ with $g_1 \cap a \leq b$ and $g_2 \cap a \leq c$ leads to contradiction, and thus either $G \cup \{b\}$ or $G \cup \{c\}$ can be extended to a proper filter, not containing a.

Exercise 2.7.13 $(\Rightarrow)$

We need to prove a slightly more general statement, namely:

> Let $\Gamma \vdash \varphi$. Then for every Kripke model $\mathcal{C}$ and every state c of $\mathcal{C}$, the condition $c \Vdash \Gamma$ implies $c \Vdash \varphi$.

(An alternative of this generalization is to remember that for each state c, the set of all c' with $c \leq c'$ makes a Kripke model.) As an example consider the induction step for rule $(\vee\mathrm{E})$. Assume that we have derived $\Gamma \vdash \varphi$ from the three assertions: $\Gamma, \psi \vdash \varphi$ and $\Gamma, \vartheta \vdash \varphi$ and $\Gamma \vdash \psi \vee \vartheta$. Let $c \Vdash \Gamma$. By the induction hypothesis we have $c \Vdash \psi \vee \vartheta$, and thus either $c \Vdash \psi$ or $c \Vdash \vartheta$. Assume the first case and we have $c \Vdash \Gamma, \psi$. By induction hypothesis we get $c \Vdash \varphi$.

Exercise 2.7.14

The induction fails in the case of disjunction.

Exercise 2.7.17

First do the previous two exercises. Now assume that φ is a classical tautology (the other direction is obvious), but $\neg\neg\varphi$ is not valid intuitionistically. This means that there is a Kripke model with a state c, such that $c \Vdash \neg\varphi$. Without loss of generality, we can assume that c determines all propositional variables in φ. Indeed, suppose that c does not determine a variable p. Then there is a $c' \geq c$ with $c' \Vdash p$, and we can take c' instead. From Exercise 2.7.16 we obtain that $c \Vdash \varphi$, a contradiction.

Exercise 4.6.4

14.0.12. LEMMA (Newman's Lemma). *Let $\to$ be a binary relation satisfying* SN. *If $\to$ satisfies* WCR, *then $\to$ satisfies* CR.

PROOF. We give the proof in the case where $\to$ satisfies FB, i.e., for all $M \in L$ the set $\{N \mid M \to N\}$ is finite. [This was the case the hint aimed at. See another proof at the end of the note.] Since $\to$ satisfies FB and SN, there is for any M an m so that any reduction sequence from M has length at most m.[1]

Assume $\to$ is SN and WCR. Given $M_1, M_2, M_3 \in L$ where $M_1 \twoheadrightarrow M_2$ and $M_1 \twoheadrightarrow M_3$, we must find $M_4 \in L$ such that

$$\begin{tikzcd}
& M_1 \arrow[dl, two heads] \arrow[dr, two heads] & \\
M_2 \arrow[dr, dotted, two heads] & & M_3 \arrow[dl, dotted, two heads] \\
& M_4 &
\end{tikzcd}$$

Since $\to$ is strongly normalizing, for every $M \in L$ there will be a longest reduction starting in M. Let $|M| \in \mathbb{N}$ denote the length of this reduction. Assume $M_1, M_2, M_3 \in L$ such that $M_1 \twoheadrightarrow M_2$ and $M_1 \twoheadrightarrow M_3$. We proceed by induction over $|M_1|$:

Case $|M_1| = 0$: Since the longest reduction has length 0, it must be the case that $M_1 = M_2$ and $M_1 = M_3$, and thus $M_4 = M_1$ is the desired term.

Case $|M_1| > 0$: Assume for all $N_1 \in L$ such that $|N_1| < |M_1|$, if $N_1 \twoheadrightarrow N_2$ and $N_2 \twoheadrightarrow N_3$ then there exists N_4 such that $N_2 \twoheadrightarrow N_4$ and $N_3 \twoheadrightarrow N_4$.

[1]How does this follow? Recall König's Lemma which states that if a tree—which is finitely branching, i.e., each node has finitely many children—is infinite, then it must have an infinite branch. Now, given M, consider the tree where the root is labeled with M, and for any node labeled with K, if $K \to N$ then the node labeled K has a child labeled N. Since $\to$ satisfies SN, there is no infinite branch. Also, there cannot be arbitrarily long finite sequences, because then the tree would be infinite, and then by König's Lemma there would be an infinite branch, contradicting SN.

If $M_1 \twoheadrightarrow M_2$ has length 0 the desired term is M_3:

$$\begin{tikzcd}
 & M_1 \arrow[dl, equal] \arrow[dr] & \\
M_2 \arrow[dr, dotted, two heads] & & M_3 \arrow[dl, equal] \\
 & M_3 &
\end{tikzcd}$$

Similarly if $M_1 \twoheadrightarrow M_3$ has length 0.

Thus assume that $M_1 \to N_2 \twoheadrightarrow M_2$ and $M_1 \to N_3 \twoheadrightarrow M_3$,

$$\begin{tikzcd}
 & M_1 \arrow[dl] \arrow[dr] & \\
N_2 \arrow[d, two heads] & & N_3 \arrow[d, two heads] \\
M_2 & & M_3
\end{tikzcd}$$

Since $\to$ is WCR from $M_1 \to N_2$ and $M_1 \to N_3$ we get a term $N \in L$ such that $N_2 \twoheadrightarrow N$ and $N_3 \twoheadrightarrow N$

$$\begin{tikzcd}
 & M_1 \arrow[dl] \arrow[dr] & \\
N_2 \arrow[d, two heads] \arrow[dr, dotted, two heads] & & N_3 \arrow[d, two heads] \arrow[dl, dotted, two heads] \\
M_2 & N & M_3
\end{tikzcd}$$

Since $M_1 \to N_2 \twoheadrightarrow N$, $|N_2| < |M_1|$. Applying the induction hypothesis we thus get a term $M'_4 \in L$ such that $M_2 \twoheadrightarrow M'_4$ and $N \twoheadrightarrow M'_4$

$$\begin{tikzcd}
 & M_1 \arrow[dl] \arrow[dr] & \\
N_2 \arrow[d, two heads] \arrow[dr, two heads] & & N_3 \arrow[d, two heads] \arrow[dl, two heads] \\
M_2 \arrow[dr, dotted, two heads] & N \arrow[d, dotted, two heads] & M_3 \\
 & M'_4 &
\end{tikzcd}$$

Then $N_3 \twoheadrightarrow M_4'$ and $N_3 \twoheadrightarrow M_3$ and hence by the induction hypothesis we get a term $M_4 \in L$ such that $M_4' \twoheadrightarrow M_4$ and $M_3 \twoheadrightarrow M_4$

$$\begin{tikzcd}
 & M_1 \arrow[dl] \arrow[dr] & \\
N_2 \arrow[d] \arrow[dr] & & N_3 \arrow[dl] \arrow[d] \\
M_2 \arrow[dr] & N \arrow[d] & M_3 \arrow[ddl, dotted] \\
 & M_4' \arrow[d, dotted] & \\
 & M_4 &
\end{tikzcd}$$

i.e., $M_2 \twoheadrightarrow M_4$ and $M_3 \twoheadrightarrow M_4$. This concludes the proof. □

14.0.13. Proposition. *There is a binary relation satisfying* WN *and* WCR*, but not* CR.

Proof. Consider $L = \{M, M_{\mathrm{NF}}, N, N_{\mathrm{NF}}\}$ and the relation $\to$ given by

$$\begin{tikzcd}
M \arrow[r, bend left] \arrow[d] & N \arrow[l, bend left] \arrow[d] \\
M_{\mathrm{NF}} & N_{\mathrm{NF}}
\end{tikzcd}$$

The relation is not Church-Rosser: for $M \in L$ we have two reductions $M \to M_{\mathrm{NF}}$ and $M \to N \to N_{\mathrm{NF}}$, but there is no term in L such that M_{NF} and N_{NF} both reduce to this term (because both are in "normal form").

The relation *is* Weak Church-Rosser: if $M \to M_{\mathrm{NF}}$ and $M \to N$, then M_{NF} is a common reduct. Similarly for N.

Finally, we have that $\to$ is weakly normalizing since any reduction can always end in either M_{NF} or N_{NF}. (It is obviously not strongly normalizing because we have an infinite reduction $M \to N \to M \to \cdots$.) □

14.0.14. Corollary. *Let $M_1 \in \Lambda$ be typable in $\lambda\!\to$ á la Curry and assume that $M_1 \twoheadrightarrow_\beta M_2$ and $M_1 \twoheadrightarrow_\beta M_3$. Then there is an $M_4 \in \Lambda$ such that $M_2 \twoheadrightarrow_\beta M_4$ and $M_3 \twoheadrightarrow_\beta M_4$.*

Proof. Let $L = \{M \in \Lambda | \exists\sigma.\ \vdash M : \sigma\}$ and consider $\to = \to_\beta$. By Theorem 4.10 $\to$ satisfies SN and by Exercise 1.72 $\to$ satisfies WCR, thus by Newman's Lemma $\to$ satisfies CR, i.e., $\to_\beta$ is Church-Rosser on L—the set of Curry-typable terms. □

How does one prove Newman's Lemma in case $\to$ does not necessarily satisfy FB? As follows.

Proof. Let $\to$ be a relation on L satisfying SN and WCR. As usual, a *normal form* is an $M \in L$ such that for all $N \in L$, $M \not\to N$.

Since $\to$ satisfies SN, any $M \in L$ reduces to a normal form. Call M *ambiguous* if M reduces to two distinct normal forms. It is easy to see that $\to$ satisfies CR if there are no ambiguous terms.

Now, for any ambiguous M there is another ambiguous M' such that $M \to M'$. Indeed, suppose $M \twoheadrightarrow N_1$ and $M \twoheadrightarrow N_2$. Both of these reductions must make at least one step since N_1 and N_2 are distinct, so the reductions have form $M \to M_1 \twoheadrightarrow N_1$ and $M \to M_2 \twoheadrightarrow N_2$. If $M_1 = M_2$ we can choose $M' = M_1 = M_2$. If $M_1 \neq M_2$ we now by WCR that for some N_3, $M_1 \twoheadrightarrow N_3$ and $M_2 \to N_3$. We can assume that N_3 is a normal form. Since N_1 and N_2 are distinct, N_3 is different from N_1 or N_2 so we can choose $M' = M_1$ or $M' = M_2$.

Thus, M has an infinite reduction sequence, contradicting SN. Hence, there are no ambiguous terms. □

Exercise 5.6.1

One possibility is $\mathbf{S(S(KS)K)I}$.

Exercise 5.6.9

It should be easy to see that all types that can be assigned to $\mathbf{K}$, $\mathbf{I}$ and $\mathbf{S}^\circ$ must be respectively of the form:

- $\tau \to \sigma \to \tau$;
- $\tau \to \tau$;
- $(\tau \to \tau) \to (\tau \to \tau \to \tau) \to (\tau \to \tau) \to \tau \to \tau$.

We can take all instances of the above formulas as our Hilbert style axioms. But we can easily simplify the the system, replacing the last axiom by:

- $(\tau \to \tau \to \tau) \to (\tau \to \tau) \to \tau \to \tau$.

A Hilbert style proof will now correspond to a combinatory term built from the combinators of Exercise 5.6.8, and without loss of generality, we can only deal with terms in normal forms.

Suppose we have a proof of $(p \to p \to q) \to p \to q$, where p and q are propositional (type) variables. Let M be the corresponding combinatory term in the language of Exercise 5.6.8, and assume that M is in a normal form.

Clearly, M is neither $\mathbf{K}_{\sigma,\tau}$, nor $\mathbf{S}_{\tau,\tau,\tau}$, nor $\mathbf{I}_\tau$. If $M = \mathbf{K}_{\sigma,\tau}N$, then N proves $p \to q$, which is not a valid tautology. Also $M = \mathbf{S}_{\tau,\tau,\tau}PQ$ or $M = \mathbf{S}_{\tau,\tau,\tau}N$ is impossible, because types of these expressions do not match $(p \to p \to q) \to p \to q$.

Exercise 6.8.1

a) To reduce $? \vdash M : ?$ to $\vdash M : ?$, observe that a term M with free variables $x_1, \dots, x_n$ is typable if and only if $\lambda x_1 \dots x_n.M$ is typable.

b) In order to find out if a closed term M is typable, ask if $x : \alpha \vdash \mathbf{K}xM : \alpha$.

Exercise 6.8.2

Of course $t_\alpha = x_\alpha$, and then we proceed by induction as follows: $t_{\sigma\to\tau} = \lambda x.\mathbf{K}t_\tau(\lambda zy.z(yx)(yt_\sigma))$.

Exercise 6.8.3 (Author's hint)

Adopt the technique of Exercise 6.8.2 to write lambda-terms $t_{[x:\tau]}$ such that $x \in FV(t_{[x:\tau]})$ and $t_{[x:\tau]}$ is typable in a context Δ if and only if $\Delta(x) = S(\tau)$, for some S. (Use a fresh variable x_α, for every type variable α occurring in τ.) Then reduce the problem of $\Gamma \vdash M : ?$ with $\Gamma = \{(x_1 : \tau_1), \dots, (x_n : \tau_n)\}$ to the problem $? \vdash M' : ?$, where $M' = \lambda z.zMt_{[x_1:\tau_1]} \dots t_{[x_n:\tau_n]}$.

Exercise 6.8.3 (Solution by Henning Makholm)

(This is a fragment of Henning's solution containing the reduction of problem (3) to problem (4).)

Definition. Let Γ^0 denote a type environment of the form discussed in exercise 6.29. That is, one that maps all variables to distinct type variables.

Definitions. I consider the following problems:

(3) $? \vdash M : \tau$;

(4) $? \vdash M : ?$;

($5\frac{1}{2}$) $\Gamma^0 \vdash M : ?$.

In each of the following reductions, x_1 up to x_n will be understood to be the free variables of the given M.

Reduction from (3) to ($5\frac{1}{2}$). To answer $? \vdash M : \tau$, ask

$$\Gamma^0 \vdash \lambda x_1 \cdots x_n yz.y(zM)(zt_\tau) : ?.$$

Reduction from $(5\frac{1}{2})$to (4). This is the harder part. Without the Γ^0 it is not trivial to design a subterm that is forced to have a primitive type (one that cannot be destructed).

Solution with product types. One simple solution can be obtained if we assume that we have product types at our disposal, together with pair and projection operators. To answer $\Gamma^0 \vdash M : ?$, ask:

$$\begin{aligned}? \vdash \mathbf{K}\ &((\lambda x_1 \cdots x_n.M)\,\langle x_1, x_1\rangle\,\langle x_2, x_2\rangle \ldots \langle x_n, x_n\rangle)\\ &((\lambda x_1 \cdots x_n.M)(\lambda y_1.y_1)(\lambda y_1 y_2.y_1) \ldots (\lambda y_1 \cdots y_n.y_1)) : ?\end{aligned}$$

Clearly, if M has a typing in Γ^0 then this complicated term is also typable. For the other direction, I exploit the existence of unique principal types. If the long term is typable, then $\lambda x_1 \cdots x_n.M$ will have a principal type

$$\tau_1 \to \cdots \to \tau_n \to \sigma.$$

Since that principal type have instances where the τ_i's are arrow types as well as where they are pair types, we conclude that the τ_i's must be type variables. Furthermore, because $\lambda y_1 \cdots y_i.y_1$ and $\lambda y_1 \cdots y_j.y_1$ cannot have equal type for $i \neq j$, the τ_i's must be *different* type variables. Thus, modulo renaming of type variables we have $\Gamma^0 \vdash M : \sigma$.

Implicational solution. My solution is based on an "encoding" $\underline{\bullet}$ of types and terms[2]:

$$\begin{aligned}\underline{\alpha} &= \alpha;\\ \underline{\sigma \to \tau} &= (\underline{\tau} \to \underline{\tau}) \to (\underline{\sigma} \to \underline{\tau});\\[1ex] \underline{x} &= x;\\ \underline{\lambda x.M} &= \lambda e x.e(e(\underline{M}));\\ \underline{MN} &= (\lambda e.e(\underline{M}e\underline{N}))(\lambda x.x).\end{aligned}$$

Lemma.

- If $\Gamma \vdash M : \tau$ then $\underline{\Gamma} \vdash \underline{M} : \underline{\tau}$.
- If $\Delta \vdash \underline{M} : \sigma$ then $\Gamma \vdash M : \tau$, for some Γ and τ with $\underline{\Gamma} = \Delta$ and $\underline{\tau} = \sigma$.

The proof of the first part is easy. The other part follows by induction with respect to M. □

[2]Never mind that $\underline{\tau}$ is not LOGSPACE-computable because $\underline{\tau}$ can be exponentially bigger than τ. It is not used in the actual reduction, just in the argument for its correctness. The important thing is that $\underline{M}$ *is* LOGSPACE-computable by having a treeless, linear definition.

Now, to answer $\Gamma^0 \vdash M : ?$, ask

$$? \vdash (\lambda x_1 \cdots x_n.\underline{M})(\lambda z y_1.z)(\lambda x y_1 y_2.z)\ldots(\lambda z y_1 \cdots y_n.z) : ?.$$

It is easy to see that this holds whenever M is typable in Γ^0. Conversely, assume that the massaged and wrapped M is typable. We know then that $\underline{M}$ is typable and thus so is M. Let (Γ, τ) be the principal pair of M.

What remains to be shown is that Γ is Γ^0. We know that $\underline{\Gamma}(x_i) = \underline{\Gamma(x_i)}$ has an instance which is a type of $\lambda z y_1 \cdots y_i.z$. But by inspection of the definition of $\underline{\bullet}$ we see that this can only be the case when $\Gamma(x_i)$ is a type variable. (This is because all compound types are translated into types with "symmetric" first arguments.) Furthermore the type variables in Γ have to be different, because $\lambda z y_1 \cdots y_i.z$ and $\lambda z y_1 \cdots y_j.z$ do not have a common type for $i \neq j$. Thus Γ is a Γ^0 which completes the reduction. □

Exercise 6.8.6

Take the equations "$\alpha_1 = \alpha_2 \to \alpha_2$", "$\alpha_2 = \alpha_3 \to \alpha_3$",..., "$\alpha_{n-1} = \alpha_n \to \alpha_n$".

Exercise 6.8.7

Use non-unifiable terms as different flags. Represent n-ary operators as combinations of arrows, using flags to identify operators.

Exercise 6.8.8

Label all subterms occurring in the equations and keep labels unchanged in τ in steps (a) and (d). A variable x is called *defined* iff there is an equation of the form "$x = t$" or "$t = x$". The occurrence of x as a left-hand or a right-hand side of such an equation is called *main*. Note that "$x = y$" has two main occurrences.

Proceed by induction w.r.t. the following parameters: 1) number of labels; 2) number of equations; 3) number of main occurrrences of defined variables; 4) number of all occurrrences of defined variables. Each application of rule (c) decreases (1) and does not increase the other parameters. Rule (e) decreases (2) without affecting (1), and rule (a) decreases (3) without increasing (1) or (2). Rule (d), used with caution (i.e., only when x occurs in "$r = s$" and when t is not a defined variable), will not increase any of the parameters (1)–(3) and will decrease (4). Thus, after a finite number of applications of rules (a), (c), (d) and (e) we obtain a system of equations, that is not normal only for the following reasons: there may be some equations of the form "$t = x$" and there may be some equations between defined variables. Use rules (b) and (d) for a clean-up.

Exercise 6.8.10

Use acyclic graphs instead of trees to represent algebraic terms. Proceed by identifying nodes of these graphs as long as either a contradiction is found (a loop or an attempt to place two different operation labels at a node) or no more nodes need to be identified.

Exercise 6.8.11

Just add one more equation.

Exercise 6.8.12

Take the term $(\lambda v.\mathbf{K}v(\lambda x_{\alpha_1} \dots x_{\alpha_n}.t_{[v:\tau]}))N$, where N is any inhabitant of τ, and $t_{[v:\tau]}$ is as in Exercise 6.8.3.

Exercise 6.8.14

Let a pair (Γ, τ) be called *semi-principal* iff it has the properties mentioned in Exercise 6.8.13. We show that if (Γ, τ) is semi-principal then there is at most one Church-style BCK-term M in long normal form such that $\Gamma \vdash M : \tau$. The proof is by induction w.r.t. the total number of arrows in (Γ, τ). Assume first that $\tau = \tau_1 \to \tau_2$. Then M cannot be of the form $xM_1 \dots M_n$, as it would not be fully-applied (x has too few arguments). Thus, $M = \lambda y.N$, with fully applied N, and we apply the induction hypothesis to the pair $(\Gamma \cup \{y : \tau_1\}; \tau_2)$.

The remaining case is when τ is a type variable α. There is at most one variable x declared in Γ to be of type $\sigma_1 \to \cdots \to \sigma_p \to \alpha$, since α occurs at most twice. Thus $M = xN_1 ... N_p$. Apply the induction hypothesis to pairs (Γ', σ_j), where Γ' is Γ without the declaration $(x : \sigma_1 \to \cdots \to \sigma_p \to \alpha)$.

Exercise 6.8.18

Modify the algorithm of Lemma 6.6.1 so that it searches for all inhabitants rather than for only one. This process may loop by asking a question of the form $\Gamma \vdash ? : \alpha$, which has already been asked. We can effectively identify all loop-free solutions and all loops. If we have a loop caused by a question that has a loop-free solution, then the answer is: "infinite". (Note the similarity of this argument to the pumping lemma for CF-languages.)

Exercise 6.8.20

Assume all τ_i's are inhabited. We prove indirectly that then φ is not inhabited. Assume $\vdash N_i : \tau_i$ and $\vdash M : \varphi$. Then $\vdash M\, N_1 \cdots N_n : \alpha$, which is a contradiction because type variables are not inhabited.

The proof that φ is inhabited if there is a τ_i that is not inhabited is by induction on the size of φ. Since α is the only type variable in φ, type τ_i has the form $\sigma_1 \to \cdots \to \sigma_m \to \alpha$. If any of the σ_j's were not inhabited then by the induction hypothesis τ_i would be inhabited, but we know it is not. Thus for some N_j's we have $\vdash N_j : \sigma_j$. Then $\vdash \lambda x_1 \cdots x_n . x_i N_1 \ldots N_m : \varphi$.

Exercise 6.8.21

The formula φ must have the form $\psi_1 \to \cdots \to \psi_n \to p$. Classically, this is equivalent to $(\psi_1 \wedge \cdots \wedge \psi_n) \to p$. For this to be a tautology, some ψ_i must be false in a valuation that sets p to be false. Hence that ψ_i is not classically valid, so it cannot be intuitionistically valid either. That means that ψ_i is not inhabited when viewed as a type. Then φ *is* inhabited by Exercise 6.8.20, and φ is thus intuitionistically valid.

Exercise 6.8.22

First reduce the general case to the case when τ is a type variable α. To prove that $\tau_1, \ldots, \tau_n \vdash \alpha$, consider a substitution S such that $S(\alpha) = \tau_1 \to \cdots \to \tau_n \to \alpha$, and $S(\beta) = \beta$, for $\beta \neq \alpha$. Show that all formulas $S(\tau_1), \ldots, S(\tau_n)$ are valid, and conclude what you need.

Exercise 7.7.6

Consider the term $M = < \pi_1(F\mathbf{K}), \pi_2(F\mathbf{K}) >$, where $F = \lambda x. < x, \lambda y.x >$. Then F can be assigned all instances of the type $\alpha \to (\alpha \wedge (\beta \to \alpha))$, and since $\mathbf{K}$ has both the types $\gamma \to \gamma \to \gamma$ and $\gamma \to (\gamma \to \gamma) \to \gamma$, we can derive that $M : (\gamma \to \gamma \to \gamma) \wedge (\beta \to \gamma \to (\gamma \to \gamma) \to \gamma)$. But $M \to_\eta F\mathbf{K}$, and $F\mathbf{K}$ cannot be assigned this type.

Exercise 7.7.8

The example term in the above solution is an untyped η-redex that is *not* an erasure of a typed η-redex. The Church-style version of that term uses two different $\mathbf{K}$'s.

Exercise 7.7.9

The problem is that the conclusion of the elimination rule is not necessarily identical to a premise of any of the introduction rules. Thus, in an elimination-introduction pair, one eliminates and introduces possibly different things.

Exercise 7.7.10

There is no canonical object of type $\perp$, so one can argue that no eta rule makes sense. By analogy to the eta rule for $\vee$ one may only postulate that an "artificial" use of $\perp$-introduction should be avoided. This leads to something like $\varepsilon_\perp(M) \to M$, for $M : \perp$.

Exercise 8.9.1 (Hint)

For the right-to-left direction, proceed as follows:

1. Show that $\vdash (\varphi \to \psi) \to (\neg\varphi \to \psi) \to \psi$.

2. Let $\alpha_1, \dots, \alpha_n$ be the propositional variables of φ. Let ρ be a Boolean valuation. Now let

$$\alpha_i' = \begin{cases} \alpha_i & \text{if } \rho(\alpha) = 1 \\ \neg\alpha_i & \text{if } \rho(\alpha) = 0 \end{cases}$$

   Also, let

$$\varphi' = \begin{cases} \varphi & \text{if } \rho(\varphi) = 1 \\ \neg\varphi & \text{if } \rho(\varphi) = 0 \end{cases}$$

   where ρ is lifted to formulas according to the usual truth-table semantics. Show that $\{\alpha_1', \dots, \alpha_n'\} \vdash \varphi'$.

3. Prove the right-to-left direction.

Exercise 9.5.2

The construction of φ' is by induction w.r.t. the length of the quantifier prefix of φ. If φ is quantifier-free then $\varphi' = \varphi$. If $\varphi = \forall x\, \psi(x)$ then $\varphi' = \psi'(x)$. If $\varphi = \exists x\, \psi(x)$ then $\varphi' = \psi'(c)$, where c is a constant such that $\vdash \psi(c)$. (Here we apply the induction hypothesis to $\psi(c)$.)

Exercise 9.5.3

For a given formula there is a finite number of possible replacements of existentially quantified variables by constants. To verify provability of a prenex formula one checks provability of these replacements (provability of open formulas is decidable, like for the propositional case).

Exercise 9.5.4 (Intuitionistic case)

The argument is of course as follows: if every intuitionistic first-order formula was equivalent to a prenex formula, then intuitionistic first-order logic would be decidable by the previous exercise. Note however that to apply this argument one needs to know that the translation to prenex normal form must be effective, and we can assume only the existence of a prenex formula equivalent to any given φ. But one can effectively list all proofs until a proof of $\varphi \leftrightarrow \psi$, for a prenex formula ψ is found, and this gives an effective translation.

Exercise 9.5.5

Consider the formula $\exists x(P(0) \vee P(1) \to P(x))$, where P is a unary predicate symbol and 0 and 1 are constants. It should be clear that no formula of the form $P(0) \vee P(1) \to P(t)$ is classically valid. The proof of Corollary 9.3.2 breaks down in the "obvious" and omitted part, and the confusion is created by the principle of *contraction* (two occurrences of the same formula in a sequent are treated as a single occurrence). In a sequent calculus with explicit contraction rules, the last rule of our proof would be the right contraction rule. In our version of sequent-calculus, the contraction is implicit, and the last rule must indeed be ($\exists$R). But the premise of this rule may be for instance of the form:

$$\vdash \exists x(P(0) \vee P(1) \to P(x)), P(0) \vee P(1) \to P(1),$$

because a classical sequent may have two formulas at the right-hand side. The second last rule is also ($\exists$R) with premise

$$\vdash P(0) \vee P(1) \to P(0), P(0) \vee P(1) \to P(1),$$

a classical proof of which should be easy.

Exercise 9.5.7 (Hint)

The proof is similar to that of Proposition 2.3.4. Choose $c \in X$. We transform a model $\mathcal{A}$ where relations are valued over $P(X)$ into an ordinary model $\bar{\mathcal{A}}$, by defining

$$(a_1, \dots, a_n) \in r^{\bar{\mathcal{A}}} \quad \text{iff} \quad c \in r^{\mathcal{A}}(a_1, \dots, a_n).$$

Both our models have a common domain A. Let v be a valuation in $\mathcal{A}$. We extend v to terms and formulas in the usual way (the values of formulas range over $P(X)$). One proves by induction the following claim:

$$c \in v(\varphi) \quad \text{iff} \quad \bar{\mathcal{A}}, v \models \varphi,$$

for all φ. (Note that the right-hand side makes sense, because v can also be regarded as ordinary two-valued valuation in $\bar{\mathcal{A}}$.) Note that in the right-to-left direction in the case of $\forall$, one uses the fact that lower bounds in $P(X)$ are actually intersections (if c belongs to all sets in a family then it belongs to their glb).[3]

Now if $\mathcal{A}, v \not\models \varphi$ then there is $c \notin v(\varphi)$, and thus $\bar{\mathcal{A}}, v \not\models \varphi$ as well.

[3]One has to use ultrafilters for the case of arbitrary Boolean algebras.

Exercise 9.5.8

Algebraic counterexamples, after [88]

The domain of individuals A is the set of all positive integers. The Heyting algebra $\mathcal{H}$ is the set of all open sets of $\mathbb{R}^2$. We take φ and ψ as atomic formulas $p(x)$ and $q(y)$, and we abbreviate $p(x)^{\mathcal{A}}(n)$ by $P(n)$ and $q(y)^{\mathcal{A}}(m)$ by Q (the latter will not depend on m). The symbol w_i denotes the i-th point with rational coefficients, according to some fixed order.

2) Take $P(n) = \mathbb{R}^2 - \{w_n\}$, a full space without one rational point. The intersection of all sets $P(n)$ is $\mathbb{R}^2$ with all rational points deleted. This set has an empty interior, and thus the value of the formula $\neg\forall x\, p(x)$ is the full space. On the other hand, the complement in $\mathcal{H}$ of every $P(n)$ is empty and the value of the formula at the right-hand side is the union of empty sets.

4) Let Q be the whole space without the point $(0,0)$, and let $P(n)$ be the set of all points with distance from $(0,0)$ greater than $\frac{1}{n}$ (full space without a disk). The union of all $P(n)$'s is equal to Q, and the value of the left-hand side is $Q \Rightarrow Q = \mathbb{R}^2$. But $Q \Rightarrow P(n) = P(n)$, for all n, and the union of these sets (the value of the right-hand side) does not contain $(0,0)$.

6) Take Q as above and let $P(n)$ be the open disk centered at $(0,0)$, with radius equal to $\frac{1}{n}$. The value of the left-hand side is $\mathbb{R}^2$ because each set $Q \cup P(n)$ covers the whole space. The value of the right-hand side is Q because the glb in $\mathcal{H}$ of all sets $P(n)$ is empty. (Their intersection is a one-point set which is not open.)

8) Let Q be empty, and let $P(n)$ be as in (2). The value of the left-hand side is the full space, because the glb of all sets $P(n)$ is empty. But the value of the right-hand side is empty.

10) Let $P(n)$ be as in (2) and (8). Then $\sim P(n)$ is empty and thus $P(n) \cup \sim P(n) = P(n)$. The intersection of these sets has an empty interior, thus the value of our formula is empty.

12) Take $P(n) = A$ for even n and $P(n) = B$ for odd n where A and B are complementary open halves of the space. (Note that this formula has a propositional counterpart: $(p \vee q \to p) \vee (p \vee q \to q)$, which is also not valid.)

Counterexamples in Kripke models

2) Hint: adopt the solution of Exercise 2.7.9(8).

4) Hint: first make a counterexample for the propositional version of this law: $(p \to q \vee r) \to (p \to q) \vee (p \to r)$.

6) The model consists of two states $c \leq d$, with $\mathcal{A}_c = \{1\}$ and $\mathcal{A}_d = \{1, 2\}$. Let $c, 1 \Vdash \varphi(x)$ and $c \nVdash \psi$, and let $d \Vdash \psi$ and $d, 1 \Vdash \varphi(x)$, but $d, 2 \nVdash \varphi(x)$. Then $c \Vdash \forall x(\psi \vee \varphi(x))$, because $d, 2 \Vdash \psi$, but $c \nVdash \psi \vee \forall x\, \varphi(x)$, because $d, 2 \nVdash \varphi(x)$.

8) Generalize (2).

10) The set of states is $\{c_n : n \in \mathbb{N}\}$, the domain for each state is the set of integers, and $c_i, j \Vdash \varphi$ if and only if $j > i$. Then $c_1 \nVdash \neg\neg\forall x(\varphi \vee \neg\varphi)$. Otherwise there would be c_i with $c_i \Vdash \forall x(\varphi \vee \neg\varphi)$, in particular $c_i, i \Vdash \varphi \vee \neg\varphi$, and this does not hold.

12) The model consists of two states $c \leq d$, with $\mathcal{A}_c = \{1\}$ and $\mathcal{A}_d = \{1, 2\}$. Let $c, 1 \nVdash \varphi(x)$, and let $d, 1 \nVdash \varphi(x)$ and $d, 2 \Vdash \varphi(x)$. Then $c \nVdash \exists x(\exists y\, \varphi(y). \to \varphi(x))$, as otherwise $c, 1 \Vdash \exists y\, \varphi(y). \to \varphi(x)$, which implies $d, 1 \Vdash \varphi(x)$.

Exercise 9.5.9

- $\exists x(\varphi(x) \to \forall x\, \varphi(x))$

Heyting counterexample as for Exercise 9.5.8(2). Kripke counterexample as for Exercise 9.5.8(6).

- $\exists x(\varphi(0) \vee \varphi(1) \to \varphi(x))$

Similar to Exercise 9.5.8(12).

- $\forall x\, \neg\neg\varphi(x). \leftrightarrow \neg\neg\forall x\, \varphi(x)$

Heyting counterexample as for Exercise 9.5.8(2). Kripke counterexample as for Exercise 9.5.8(10).

- $\exists x\, \neg\neg\varphi(x). \leftrightarrow \neg\neg\exists x\, \varphi(x)$.

Heyting counterexample as for Exercise 9.5.8(12). A Kripke counterexample consists of three states $c \leq d, e$ (with d, e incomparable), all with domain $\{1, 2\}$. The forcing relation consists only of $d, 1 \Vdash \varphi(x)$ and $e, 2 \Vdash \varphi(x)$.

Exercise 9.5.10

The implication from right to left is always valid, so we only consider the implication $\forall x(\psi \vee \varphi(x)) \to \psi \vee \forall x\, \varphi(x)$. Assume that $\mathcal{A}_c = \mathcal{A}$ for all c in our model. Let $c, \vec{\mathtt{a}} \Vdash \forall x(\psi \vee \varphi(x))$, and suppose that $c, \vec{\mathtt{a}} \nVdash \psi$. Thus for all $\mathtt{b} \in \mathcal{A}$ it must be that $c, \mathtt{b}, \vec{\mathtt{a}} \Vdash \varphi(x)$. By monotonicity, $c', \mathtt{b}, \vec{\mathtt{a}} \Vdash \varphi(x)$, for all $c' \geq c$. It follows that $c, \vec{\mathtt{a}} \Vdash \forall x\, \varphi(x)$.

Exercise 9.5.11

First observe that $c'', \mathsf{b}, \vec{\mathsf{a}} \not\Vdash \varphi \vee \neg\varphi$ always holds if c'' is a maximal state. Now suppose $c, \vec{\mathsf{a}} \Vdash \neg\neg\forall x(\varphi \vee \neg\varphi)$ in a finite Kripke model. There is $c' \geq c$ with $c', \vec{\mathsf{a}} \Vdash \neg\forall x(\varphi \vee \neg\varphi)$. Take a maximal state $c'' \geq c'$ and we obtain $c'', \mathsf{b}, \vec{\mathsf{a}} \not\Vdash \varphi \vee \neg\varphi$.

Exercise 10.7.11

Consider a logic where all formulas of the form $\forall x\, \varphi$ are equivalent to *true*. Then the other two axioms remain sound but $\forall x\, \varphi(x). \to \varphi(t)$ is not valid.

Exercise 11.8.1

- $\forall x\, (\neg(x = 0) \to \exists y(x = y + 1))$

We use induction scheme applied to the formula $\neg(x = 0) \to \exists y(x = y+1)$. This means we have to show that the following formulas are provable:

$\neg(0 = 0) \to \exists y(0 = y + 1)$;
$\forall x\, [(\neg(x = 0) \to \exists y(x = y + 1)) \to \neg(x + 1 = 0) \to \exists y(x + 1 = y + 1)]$,

and then it suffices to apply *modus ponens* twice. The first formula is proven with help of the first axiom, which can be generalized to $0 = 0$. The second formula is easily derived from $\exists y(x + 1 = y + 1)$, and this is a consequence of $y + 1 = y + 1$ (first axiom).

- $\forall x \forall y \forall z\, (x = y \to y = z \to x = z)$;

This is an instance of the third axiom.

- $\forall x \forall y \forall z\, ((x + y) + z = x + (y + z))$

We prove $\forall z\, ((x+y)+z = x+(y+z))$ by induction and then generalize over x and y. The first step is $(x+y)+0 = x+(y+0)$ and is an easy application of the axiom $\forall x\, (x + 0 = x)$ and the transitivity of equality. Then we must derive $(x+y)+(z+1) = x+(y+(z+1))$ from $(x+y)+z = x+(y+z)$. With the axiom $\forall x \forall y\, (x + (y + 1) = (x + y) + 1)$ and the transitivity of equality, we can formalize the calculation $(x + y) + (z + 1) = ((x + y) + z) + 1 = (x + (y + z)) + 1 = x + ((y + z) + 1) = x + (y + (z + 1))$.

- $\forall x \forall y\, (x + y = y + x)$

Hint: First prove by induction that $\forall y\, (0 + y = y + 0)$. This is the base of the main induction (with respect to x). The induction step is to derive $\forall y\, ((x + 1) + y = y + (x + 1))$ from $\forall y\, (x + y = y + x)$. This also goes by induction (with respect to y). Have fun!

Exercise 11.8.3

Suppose a function f is representable by a formula φ. Then f is strongly representable by the formula

$$\psi(\vec{x}, y) = (\exists! z\, \varphi(\vec{x}, z). \wedge \varphi(\vec{x}, y)) \vee (\neg\exists! z\, \varphi(\vec{x}, z). \wedge y = 0).$$

We show PA $\vdash \forall\vec{x}\, \exists! y\, \psi(\vec{x}, y)$. (Then ψ implies φ and representability follows easily.) Begin with PA $\vdash \vartheta \vee \neg\vartheta$, where $\vartheta = \exists! z\, \varphi(\vec{x}, z)$. Then one can show that PA $\vdash \vartheta \to \exists! y(\vartheta \wedge \varphi(\vec{x}, y))$ and PA $\vdash \neg\vartheta \to \exists! y(\neg\vartheta \wedge y = 0)$. Thus PA $\vdash \exists! y(\vartheta \wedge \varphi(\vec{x}, y)) \vee \exists! y(\neg\vartheta \wedge y = 0)$ and it remains to permute $\vee$ with $\exists! y$.

Exercise 11.8.8

By translation, we have HA $\vdash \neg\neg\varrho$ (note that $t_f(\vec{x}, y) = 0$ is an atomic formula), and thus HA $\vdash (\varrho^\varrho \to \varrho) \to \varrho$. Now, the formula ϱ^ϱ is equivalent to ϱ (because $\vdash \exists y(\psi \vee \exists y\psi) \leftrightarrow \exists y\psi$) and thus HA $\vdash (\varrho \to \varrho) \to \varrho$. Then HA $\vdash \varrho$, because $\vdash \varrho \to \varrho$.

Exercise 12.7.1 (Hint)

Cf. Exercise 9.5.10.

Exercise 12.7.2

For a model with a constant domain D we have

$$c, v \Vdash \forall p\, \varphi \quad \text{iff} \quad c, v_p^x \Vdash \varphi, \text{ for all } x \in D.$$

(One does not need to refer to $c' \geq c$ because of monotonicity.)

Exercise 12.7.3

This exercise contains a little trap: the notion of a complete model refers to all formulas, including those containing $\vee$, $\wedge$ and $\exists$. But one can also consider Kripke models for the language containing only $\to$ and $\forall$, satisfying the appropriate (weaker) notion of completeness. Then the semantics of the formula

$$\tau + \sigma := \forall\alpha((\tau \to \alpha) \to (\sigma \to \alpha) \to \alpha)$$

does not have to coincide with the expected semantics of $\tau \vee \sigma$. Indeed, consider a model of three states c_0, c_1 and c_2 with a constant domain $D = \{\{\,\}, \{c_1\}, \{c_2\}, \{c_0, c_1, c_2\}\}$ and with $c_1 \Vdash p$ and $c_2 \Vdash q$, and no other forcing. This model is complete with respect to $\to$ and $\forall$, and we have $c_0 \Vdash p + q$.

The case of $\wedge$ is different: if a model is complete with respect to $\to$ and $\forall$, then $c \Vdash \forall\alpha((\tau \to \sigma \to \alpha) \to \alpha)$ iff $c \Vdash \tau$ and $c \Vdash \sigma$.

Exercise 12.7.13

In the context $\{y : \bot, z : \forall\gamma(\alpha \to \gamma \to \beta)\}$ one can derive $zyz : \beta$. Now consider the context $\{y : \bot, x : \forall\alpha\beta(\forall\gamma(\alpha \to \gamma \to \beta). \to \beta)$ and derive $xx : \bot$. Thus our term has type $\forall\alpha\beta(\forall\gamma(\alpha \to \gamma \to \beta). \to \beta). \to \bot$ in the context $\{y : \bot\}$.

Exercise 12.7.15

Think of types as finite binary trees with leaves labeled by type variables and internal nodes corresponding to arrows). Some of the internal nodes are labeled by quantifiers.

Suppose that xx is typable in an environment containing the declaration $(x : \tau)$. The type τ must begin with one or more universal quantifiers, and one of these quantifiers must bind a type variable occurring at the very end of the leftmost path of the type. (Otherwise self-application is impossible.) Thus, a type assigned to $\lambda x.\, xx$ must have the form $(\forall\vec{\alpha}\tau) \to \sigma$ with one of the $\vec{\alpha}$'s at the end of the leftmost path. This observation applies to both copies of $\lambda x.\, xx$ which results in that two different quantifiers attempt to bind the same variable — a contradiction.

Exercise 12.7.16 (After [111])

The type σ assigned to $\mathbf{K}$ must have the form:

$$\sigma = \forall\vec{\alpha}(\tau \to \forall\vec{\beta}(\rho \to \tau')),$$

where τ' is an instance of τ. The rightmost path of τ' must be at least as long as the rightmost path in τ. In addition, one of the variables $\vec{\alpha}$, say α, must occur at the end of the rightmost path in τ. The same α must remain at the end of the rightmost path in the instance τ', at the same depth.

The type of the second c_2 in c_2c_2K must have the form $\forall\vec{\gamma}(\sigma_0 \to \varrho)$, where σ can be obtained from σ_0 by instantiating $\vec{\gamma}$. It begins with $\forall\vec{\alpha}$ and has occurrences of α at the same places as σ does. In particular there is an occurrence of α at some depth n at the rightmost path of the left subtree of σ_0 and at depth $n+1$ at the rightmost path of the right subtree of σ_0. Now, σ_0 is the type of f in $f(fx)$ and ϱ is the type of $\lambda x.\, f(fx)$. No matter what is the type of x, we can note that the asymetry of σ_0 is doubled in ϱ, and thus the rightmost path in ϱ must be of length at least $n+3$. Although $\forall\vec{\gamma}(\sigma_0 \to \varrho)$ may still be a good type for c_2, a term of this type cannot be composed with itself, as the positions of α cannot be changed by just instantiating $\vec{\gamma}$.

Exercise 12.7.18(12.48)

An infinite sequence of Church-style beta reductions $M_i \to_\beta M_{i+1}$ erases to an infinite sequence of Curry-style terms $|M_i|$, where at each step we either have $|M_i| \to_\beta |M_{i+1}|$ or $|M_i| = |M_{i+1}|$. The latter case must hold for almost all i, and is only possible when almost all steps $M_i \to_\beta M_{i+1}$ are caused by type reductions of the form $(\Lambda\alpha.M)\tau \longrightarrow_\beta M[\alpha := \tau]$. But each of these steps decreases the number of Λ's in our term, so this process must also terminate.

Bibliography

[1] A.R. Anderson and N.A. Belnap. *Entailment. The Logic of Relevance and Necessity*, volume I. Princeton University Press, 1975.

[2] A.R. Anderson, N.A. Belnap, and J.M. Dunn. *Entailment. The Logic of Relevance and Necessity*, volume II. Princeton University Press, 1992.

[3] T. Arts. Embedding first order predicate logic in second order propositional logic. Master's thesis, Katholieke Universiteit Nijmegen, 1992.

[4] T. Arts and W. Dekkers. Embedding first order predicate logic in second order propositional logic. Technical Report 93-02, Katholieke Universiteit Nijmegen, 1993.

[5] S. van Bakel, L. Liquori, S Ronchi della Rocca, and P. Urzyczyn. Comparing cubes of typed and type assignment systems. *Annals of Pure and Applied Logic*, 86:267–303, 1997.

[6] F. Barbanera, M. Dezani-Ciancaglini, and U. de' Liguoro. Intersection and union types: syntax and semantics. *Information and Computation*, 119:202–230, 1995.

[7] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, second, revised edition, 1984.

[8] H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 117–309. Oxford University Press, 1992.

[9] H.P. Barendregt, M. Bunder, and W. Dekkers. Completeness of some systems of illative combinatory logic for first-order propositional and predicate calculus. To appear in *Archive für Mathematische Logik*, 1996.

[10] H.P. Barendregt, M. Bunder, and W. Dekkers. Completeness of the propositions-as-types interpretation of intuitionistic logic into illative combinatory logic. To appear in the *Journal of Symbolic Logic*, 1996.

[11] J. Barwise. *Handbook of Mathematical Logic.* North-Holland, 1977.

[12] S. Berardi. *Type Dependence and Constructive Mathematics.* PhD thesis, Universita di Torino, 1990.

[13] M. Bezem and J. Springintveld. A simple proof of the undecidability of inhabitation in λP. *Journal of Functional Programming*, 6(5):757–761, 1996.

[14] R. Bloo and K. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95 - Computer Science in the Netherlands*, pages 62–72, 1995.

[15] V. Breazu Tannen, D. Kesner, and L. Puel. A typed pattern calculus. In *Logic in Computer Science*, pages 262–274, 1993.

[16] S. Broda and L. Damas. On principal types of stratified combinators. Technical Report DCC-97-4, Departamento de Cincia de Computadores, Universidade do Porto, 1997.

[17] L.E.J. Brouwer. Intuïtionistische splitsing van mathematische grondbegrippen. *Nederl. Akad. Wetensch. Verslagen*, 32:877–880, 1923.

[18] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[19] R. Constable. Constructive mathematics and automatic program writers. In *Proceddings of the IFIP Congress*, pages 229–233, Ljubljana, 1971.

[20] R. Constable. Programs as proofs: A synopsis. *Information Processing Letters*, 16(3):105–112, 1983.

[21] H.B. Curry. Grundlagen der Kombinatorischen Logik. teil I. *American Journal of Mathematics*, LII:509–536, 1930.

[22] H.B. Curry. Grundlagen der Kombinatorischen Logik. teil II. *American Journal of Mathematics*, LII:789–834, 1930.

[23] H.B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Science USA*, 20:584–590, 1934.

[24] H.B. Curry and R. Feys. *Combinatory Logic.* North-Holland, 1958.

[25] H.B. Curry, J.R. Hindley, and J.P. Seldin. *Combinatory Logic II*, volume 65 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1972.

[26] D. van Dalen. Intuitionistic logic. In *Handbook of Philosophical Logic*, volume III, pages 225–339. Reidel Publ. Co., 1986.

[27] L. Damas and R. Milner. Principal type schemes for functional programs. In *Conference Record of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 207–212, Jan. 1982.

[28] N.G. de Bruijn. A survey of the project AUTOMATH. In Seldin and Hindley [98], pages 579–606.

[29] A. Degtyarev and A. Voronkov. Decidability problems for the prenex fragment of intuitionistic logic. In *Logic in Computer Science*, pages 503–512, 1996.

[30] W. Dekkers. Inhabitation of types in the simply typed λ-calculus. *Information and Computation*, 119:14–17, 1995.

[31] G. Dowek. The undecidability of typability in the lambda-pi-calculus. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculus and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 139–145. Springer-Verlag, 1993.

[32] A. Dragalin. A completeness theorem for higher-order intuitionistic logic. an intuitionistic proof. In D. Skordev, editor, *Mathematical Logic and its Applications*, pages 107–124. Plenum Press, New York, 1987.

[33] C. Dwork, P.C. Kanellakis, and J.C. Mitchell. On the sequential nature of unification. *Journal of Logic Programming*, 1:35–50, 1984.

[34] J.E. Fenstad, editor. *Proc. Second Scandinavian Logic Symposium.* North-Holland, Amsterdam, 1971.

[35] S. Fortune, D. Leivant, and M. O'Donnell. The expresssiveness of simple and second-order type structures. *Journal of the Association for Computing Machinery*, 30:151–185, 1983.

[36] D.M. Gabbay. On 2nd order intuitionistic propositional calculus with full comprehension. *Archiv für Mathematische Logik und Grundlagenforschung*, 16:177–186, 1974.

[37] D.M. Gabbay. *Semantical Investigations in Heyting's Intuitionistic Logic.* D. Reidel Publ. Co, 1981.

[38] J.H. Gallier. Constructive logics, part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110:249–339, 1993.

[39] G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[40] J.H. Geuvers. Conservativity between logics and typed lambda-calculi. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs*, volume 806 of *Lecture Notes in Computer Science*, pages 79–107. Springer-Verlag, 1993.

[41] J.H. Geuvers. *Logics and Type Systems.* PhD thesis, University of Nijmegen, 1993.

[42] J.H. Geuvers and M.J. Nederhof. A modular proof of strong normalization for the calculus of constructions. *Journal of Functional Programming*, 1(2):155–189, 1991.

[43] P. Giannini and S. Ronchi Della Rocca. Characterization of typings in polymorphic type discipline. In *Proc. Symp. on Logic in Computer Sciene*, pages 61–70. Computer Society Press, 1988.

[44] J.-Y. Girard. Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur. Thèse d'État, Université Paris VII, 1972.

[45] J.-Y. Girard. The system F of variable types, fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[46] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.

[47] C.A. Goad. Monadic infinitary propositional logic. *Reports on Mathematical Logic*, 10, 1978.

[48] K. Gödel. über eine bisher noch nicht benüntze erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1980. (English translation: *J. Philos. Logic*, 9:133–142, 1980.).

[49] T.G. Griffin. A formulae-as-types notion of control. In *Conference Record of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 47–58. ACM Press, 1990.

[50] K. Grue. Map theory. *Theoretical Computer Science*, 102:1–133, 1992.

[51] R. Harper, F. Honsell, and F. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[52] H. Herbelin. A λ-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *Computer Science Logic 1994*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 1995.

[53] A. Heyting. *Mathematische Grundlagenforschung. Intuitionismus. Beweisteorie.* Springer, 1934.

[54] J.R. Hindley. *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1997.

[55] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus.* Cambridge University Press, 1986.

[56] S. Hirokawa. Principal types of BCK-lambda terms. *Theoretical Computer Science*, 107:253–276, 1993.

[57] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[58] W. Howard. The formulae-as-types notion of construction. In Seldin and Hindley [98], pages 479–490.

[59] G. Huet and G. Plotkin. *Logical Frameworks.* Cambridge University Press, 1991.

[60] S.C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[61] S.C. Kleene. *Introduction to Metamathematics.* Van Nostrand, 1952.

[62] S.C. Kleene. Origins of recursive function theory. *Annals of the History of Computing*, 3(1):52–67, 1981.

[63] A. Kolmogorov. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35:58–65, 1932.

[64] G. Kreisel. Monadic operators defined by means of propositional quantification in intuitionistic logic. *Reports on Mathematical Logic*, 12:9–15, 1981.

[65] P. Kremer. On the complexity of propositional quantification in intuitionistic logic. *Journal of Symbolic Logic*, 62(2):529–544, 1997.

[66] J.-L. Krivine. *Lambda-Calculus, Types and Models.* Ellis Horwood Series in Computers and their Applications. Masson and Ellis Horwood, English Edition, 1993.

[67] M. Löb. Embedding first order predicate logic in fragments of intuitionistic logic. *Journal of Symbolic Logic*, 41(4):705–718, 1976.

[68] H.G. Mairson. A simple proof of a theorem of Statman. *Theoretical Computer Science*, 103(2):387–394, 1992.

[69] S.D. Marlow. *Deforestation for Higher-Order Functional Languages.* PhD thesis, University of Glasgow, 1996.

[70] E. Mendelson. *Introduction to Mathematical Logic.* Wadswoth & Brooks/Cole Advanced Books and Software, third edition, 1987.

[71] E. Mendelson. *Introduction to Mathematical Logic.* Chapman & Hall, London, fourth edition, 1997.

[72] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[73] J. Mitchell. Polymorphic type inference and containment. *Information and Control*, 76:211–249, 1988.

[74] J.C. Mitchell. *Foundations for Programming Languages.* MIT Press, Cambridge, 1996.

[75] C.R. Murthy. *Extracting Constructive Contents from Classical Proofs.* PhD thesis, Cornell University, 1990.

[76] C.R. Murthy. Control operators, hierachies, and pseudo-classical type systems: A-translation at work. In *ACM SIGPLAN Workshop on Continuations*, 1992.

[77] B. Norström, K. Petersson, and J.M. Smith. *Programming in Martin-Löf's Type Theory, An Introduction.* Oxford University Press, 1990.

[78] C.-H. L. Ong. A semantic view of classical proofs: Type-theoretic, categorical, and denotational characterizations. In *Logic in Computer Science*, pages 230–241, 1996.

[79] M. Parigot. Free deduction: An analysis of "computations" in classical logic. In *Second Russian Conference on Logic programming*, volume 592 of *Lecture Notes in Artificial Intelligence*, pages 361–380. Springer-Verlag, 1991.

[80] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.

[81] M. Parigot. Classical proofs as programs. In *Kurt Gödel Colloquium*, volume 713 of *Lecture Notes in Computer Science*, pages 263–276. Springer-Verlag, 1993.

[82] M. Parigot. Strong normalization for second order classical natural deduction. In *Logic in Computer Science*, 1993.

[83] G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[84] G. Pottinger. Normalization as a homomorphic image of cut-elimination. *Annals of Mathematical Logic*, 12:323–357, 1977.

[85] D. Prawitz. *Natural Deduction: A Proof Theoretical Study.* Almquist & Wiksell, 1965.

[86] D. Prawitz. Some results for intuitionistic logic with second order quantification. pages 259–270. North-Holland, Amsterdam, 1970.

[87] D. Prawitz. Ideas and results of proof theory. In Fenstad [34], pages 235–307.

[88] H. Rasiowa and R. Sikorski. *The Mathematics of Metamathematics.* PWN, Warsaw, 1963.

[89] N.J. Rehof and M.H. Sørensen. The λ_Δ calculus. In M. Hagiya and J. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 516–542. Springer-Verlag, 1994.

[90] J. Reynolds. Towards a theory of type structure. In B Robinet, editor, *Proceedings of the Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag, 1974.

[91] A. Rezus. Classical proofs: Lambda calculus methods in elementary proof theory, 1991. Manuscript.

[92] A. Rezus. Beyond BHK, 1993. Manuscript.

[93] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965.

[94] J.B. Rosser. Highlights of the history of the lambda-calculus. *Annals of the History of Computing*, 6(4):337–349, 1984.

[95] A. Schubert. Second-order unification and type inference for church-style polymorphism. In *Proc. 25th ACM Symposium on Principles of Programming Languages*, pages 233–244, January 1998.

[96] H. Schwichtenberg. Elimination of higher type levels in definitions of primitive recursive function by means of transfinite recursion. In H.E. Rose, editor, *Logic Colloquium '73*, pages 279–303. North-Holland, 1975.

[97] H. Schwichtenberg. Definierbare Funktionen im Lambda-Kalkul mit Typen. *Archiv Logik Grundlagenforsch.*, 17:113–114, 1976.

[98] J.P. Seldin and J.R. Hindley, editors. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press Limited, 1980.

[99] D. Skvortsov. Non-axiomatizable second-order intuitionistic propositional logic. *Annals of Pure and Applied Logic*, 86:33–46, 1997.

[100] S.K. Sobolev. On the intuitionistic propositional calculus with quantifiers (russian). *Mat. Zamietki AN SSSR*, 22(1):69–76, 1977.

[101] M.H. Sørensen. *Normalization in λ-Calculus and Type Theory.* PhD thesis, Department of Computer Science, University of Copenhagen, 1997.

[102] M.H. Sørensen. Strong normalization from weak normalization in typed λ-calculi. *Information and Computation*, 133(1):35–71, 1997.

[103] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.

[104] W.W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):190–212, 1967.

[105] W.W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer-Verlag, 1975.

[106] A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics.* Springer-Verlag, 1973.

[107] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics, An Introduction, Volume I*, volume 121 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1988.

[108] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics, An Introduction, Volume II*, volume 123 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1988.

[109] A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1996.

[110] J. Tyszkiewicz. Złożoność problemu wyprowadzania typów w rachunku lambda. Master's thesis, Warsaw, 1988.

[111] P. Urzyczyn. Positive recursive type assigment. In J. Wiedermann and P. Hájek, editors, *Mathematical Foundations of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, pages 382–391. Springer-Verlag, 1995.

[112] P. Urzyczyn. Type inhabitation in typed lambda calculi (a syntactic approach). In de Groote P. and J.R. Hindley, editors, *Typed Lambda Calculus and Applications*, volume 1210 of *Lecture Notes in Computer Science*, pages 373–389. Springer-Verlag, 1995.

[113] R. Vestergaard. The cut rule and explicit substitutions. Manuscript, 1998.

[114] A. Voronkov. Proof search in intuitionistic logic with equality or back to simultaneous rigid E-unification. In M.A. Mc Robbie and J.K. Slaney, editors, *CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1996.

[115] P.L. Wadler. Deforestation: Transforming programs to eliminate intermediate trees. *Theoretical Computer Science*, 73:231–248, 1990.

[116] P.L. Wadler. A Curry-Howard isomorphism for sequent calculus. Manuscript, 1993.

[117] J. Wells. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *Proc. Symp. on Logic in Computer Sciene*, pages 176–185. IEEE, Computer Society, Computer Society Press, 1994.

[118] J.I. Zucker. Correspondence between cut-elimination and normalization. *Annals of Mathematical Logic*, 7:1–156, 1974.