

TYPED LAMBDA CALCULUS

Volume I: $\lambda_{\rightarrow}^{\mathbb{A}}$ $\lambda_{=}^{\mathcal{A}}$ $\lambda_{\cap}^{\mathcal{S}}$

Henk Barendregt

Wil Dekkers

Richard Statman

Version: August 21, 2006

Preface

DRAFT. DO NOT DISTRIBUTE.

Bla bla. yghbkjhbkn.
And we love our women.

Nijmegen and Pittsburgh

August 21, 2006

Henk Barendregt^{1,2}
Wil Dekkers¹
Rick Statman²

¹Faculty of Mathematics and Computer Science
Catholic University
Nijmegen, The Netherlands

²Departments of Mathematics and Computer Science
Carnegie Mellon University
Pittsburgh, USA

Overture

This book about typed lambda terms comes in two volumes: the present one about lambda terms typed using *simple*, *recursive* and *intersection* types and a planned second volume about *higher order*, *dependent* and *inductive* types.

In some sense this book is a sequel to Barendregt [1984]. That book is about untyped lambda calculus. Types give the untyped terms more structure: function applications are allowed only in some cases. In this way one can single out untyped terms having special properties. But there is more to it. The extra structure makes the theory of typed terms quite different from the untyped ones.

The emphasis of the book is on syntax. Models are introduced only in so far they give useful information about terms and types or if the theory can be applied to them.

The writing of the book has been different from that about the untyped lambda calculus. First of all, since many researchers are working on typed lambda calculus, we were aiming at a moving target. Also there was a wealth of material to work with. For these reasons the book has been written by several authors. Several long-term open problems had been solved during the interval these volumes were written, notably the undecidability of lambda definability in finite models, the undecidability of second order typability, the decidability of the unique maximal theory extending $\beta\eta$ -conversion and the fact that the collection of closed terms of not every simple type is finitely generated. (One of the remaining open problems is the decidability of matching at arbitrary types higher than order 4.) The book is not written as an encyclopedic volume: many topics are only partially treated. For example reducibility among types is analyzed for simple types built up from only one atom.

One of the recurring distinctions made in the two volumes is the difference between the implicit typing due to Curry versus the explicit typing due to Church. In the latter case the terms are an enhanced version of the untyped terms, whereas in the Curry theory to some of the untyped terms a collection of types is being assigned. Volume I is mainly about Curry typing, although Part I of it we also treat the for simple types equivalent Church variant in parallel.

The applications of the theory are either within the theory itself, in the theory of programming languages, in proof theory, including the technology of fully formalized proofs used for mechanical verification, or in linguistics. Often the applications are given in an exercise with hints.

We hope that the volumes will inspire readers to pursue the topic.

Contributors

Fabio Alessi	Part III, except §16.3
Henk Barendregt	All parts, except §§5.3, 5.4, 5.5
Marc Bezem	§§5.3, 5.4, 5.5
Felice Cardone	Part II
Mario Coppo	Part II
Wil Dekkers	Parts II, III
Mariangiola Dezani-Ciancaglini	Part III, except §16.3
Gilles Dowek	Chapter 4
Silvia Ghilezan	§6.2
Furio Honsell	Part III, except §16.3
Michael Moortgat	§6.4
Richard Statman	Parts I, II
Paweł Urzyczyn	§16.3

Contents in short

Part I	Simple Types λ_{\rightarrow}^A	11
1	The systems λ_{\rightarrow}	15
2	Properties	45
3	Tools	79
4	Unification and Matching	139
5	Extensions	171
6	Applications	233
7	Further reading	253
Part II	Recursive Types $\lambda_{=}^A$	255
8	Basic concepts	259
9	Properties of recursive types	289
10	Properties of terms with types	311
11	Models	327
12	Applications	357
13	Further Reading	365
Part III	Intersection types λ_{\cap}^{SS}	367
14	An exemplary system	371
15	The systems $\lambda_{\rightarrow\cap}$	381
16	Basic properties	397
17	Type Structures and Lambda Structures	429
18	Models	475
19	Applications	505
20	Further Readings	539
Indices		543
	Index of names	543
	Index of definitions	543
	Index of notations	543
References		543

Contents

I	Simple types λ_{\rightarrow}	20.8.2006:919	9
1	The systems λ_{\rightarrow}		13
1.1	The λ_{\rightarrow} systems <i>à la</i> Curry		13
1.2	Normal inhabitants		21
1.3	Representing data types		26
1.4	The λ_{\rightarrow} systems <i>à la</i> Curry, <i>à la</i> Church and <i>à la</i> de Bruijn		37
1.5	Exercises		44
2	Properties		49
2.1	First properties		49
2.2	Proofs of strong normalization		59
2.3	Checking and finding types		62
2.4	Finding inhabitants		70
2.5	Exercises		78
3	Tools		85
3.1	Typed lambda Models		85
3.2	Lambda Theories and Term Models		93
3.3	Syntactic and Semantic logical relations		99
3.4	Type reducibility		113
3.5	The five canonical term-models		128
3.6	Exercises		143
4	Definability, Unification and Matching		151
4.1	Undecidability of lambda definability		151
4.2	Undecidability of Unification		162
4.3	Decidability of matching of rank 3		167
4.4	Decidability of the maximal theory		180
4.5	Exercises		185
5	Extensions		191
5.1	Lambda delta		191
5.2	Surjective pairing 20.8.2006:919		202

5.3	Gödel's system \mathcal{T} : higher-order primitive recursion	224
5.4	Spector's system \mathcal{B} : bar recursion	240
5.5	Platek's system \mathcal{Y} : fixed point recursion	249
5.6	Exercises	252
6	Applications	257
6.1	Functional programming	257
6.2	Logic and proof-checking	258
6.3	Proof theory	266
6.4	Grammars, terms and types	276
7	Further Reading	289

Part I

Simple types $\lambda \rightarrow$

21.8.2006:951

Lambda calculus is worth studying, because it is a simple formal system that provides a model of computation with a distinctive quality. At first its expressions represent both a program and its data which on their turn evolve by simple rules (β and η -reduction) to instantaneous descriptions of intermediate computational results possibly ending in output. These features, however, are also present in Term Rewriting Systems. Lambda calculus has two extra qualities setting it apart from these other Rewriting Systems. Firstly it is applicative, in that an expression may be applied to another expression, giving them the possibility to act both as function and as argument. Secondly, Lambda Calculus has abstraction built in, meaning that the expressions are closed under explicit definitions.

Lambda calculus as model of computation can be introduced in an untyped fashion: arbitrary expressions may be applied to each other. A similarly flat approach to computing was present in the early assembly languages. Later in imperative programming languages types were introduced to keep order among the code. Exactly the same use of types happened even earlier with lambda calculus as model of computation. Types from a very partial specification of what a program does. In this sense types are somewhat comparable to dimensions in physics that provide—on the basis of meaning—an order in the wealth of quantitative notations. There are certain basic dimensions, like g (gram), m (meter) and s (second) and other dimensions can be expressed using these.

The systems of *simple types* considered in Part I are built up from atomic types \mathbb{A} using as only operator the constructor \rightarrow of forming function spaces. For example, from the atoms $\mathbb{A} = \{\alpha, \beta\}$ one can form types $\alpha \rightarrow \beta$, $(\alpha \rightarrow \beta) \rightarrow \alpha$, $\alpha \rightarrow (\alpha \rightarrow \beta)$ and so on. Two choices of the set of atoms will be made most often are $\mathbb{A} = \{\alpha_0, \alpha_1, \alpha_2, \dots\}$, an infinite set of type variables, and $\mathbb{A} = \{o\}$, consisting of only one atomic type. Particular atomic types that occur in applications are e.g. Bool, Nat, Real. Even for these simple type systems, the ordering effect is quite powerful.

As a short anthology of what is going to come in Part I we state the following. For an untyped lambda term one can find the collection of its possible types. Similarly, given a simple type, one can find the collection of its possible inhabitants (in normal form). Equality of terms of a certain type can be reduced to equality of terms in a fixed type. The problem of unification

$$\exists X:A. MX =_{\beta\eta} NX$$

is for complex enough A undecidable. That of pattern matching

$$\exists X:A. MX =_{\beta\eta} N$$

will turn out to be decidable for A simple enough. It is open if this also holds for arbitrary types. The terms of finite type are extended by δ -functions, functionals for primitive and bar recursion. Applications of the theory in computing, proof-checking and linguistics will be discussed.

Chapter 1

The systems λ_{\rightarrow}

1.1. The λ_{\rightarrow} systems *à la* Curry

Types in this part are syntactic objects built from atomic types using the operator \rightarrow . In order to classify untyped lambda terms, such types will be assigned to a subset of these terms. The main idea is that if M gets type $A \rightarrow B$ and N gets type A , then the application MN is ‘legal’ (as M is considered as a function from terms of type A to those of type B) and gets type B . In this way types help determining what terms fit together.

1.1.1. DEFINITION. (i) Let \mathbb{A} be a non-empty set of ‘atomic types’. The set of *simple types over* \mathbb{A} , notation $\mathbb{T} = \mathbb{T}_{\mathbb{A}}$, is inductively defined as follows.

$$\begin{array}{lll} \alpha \in \mathbb{A} & \Rightarrow & \alpha \in \mathbb{T} \quad \text{type atoms;} \\ A, B \in \mathbb{T} & \Rightarrow & (A \rightarrow B) \in \mathbb{T} \quad \text{function space types.} \end{array}$$

Such definitions will be used often and for these it is convenient to use the so called *abstract syntax*, see Waite and Goos [1984]. As an example we give the abstract syntax for $\mathbb{T} = \mathbb{T}_{\mathbb{A}}$.

$$\boxed{\mathbb{T} \quad = \quad \mathbb{A} \mid \mathbb{T} \rightarrow \mathbb{T}}$$

Figure 1.1: Simple types

- (ii) Let $\mathbb{A}_o = \{o\}$. Then we write $\mathbb{T}_o = \mathbb{T}_{\mathbb{A}_o}$.
- (iii) Let $\mathbb{A}_{\infty} = \{\alpha_0, \alpha_1, \alpha_2, \dots\}$. Then we write $\mathbb{T}_{\infty} = \mathbb{T}_{\mathbb{A}_{\infty}}$.

We consider that $o = \alpha_0$, hence $\mathbb{T}_o \subseteq \mathbb{T}_{\infty}$. If we write simply \mathbb{T} , then this refers to $\mathbb{T}_{\mathbb{A}}$ for an unspecified \mathbb{A} .

1.1.2. NOTATION. (i) If $A_1, \dots, A_n \in \mathbb{T}$, then

$$A_1 \rightarrow \dots \rightarrow A_n \equiv (A_1 \rightarrow (A_2 \rightarrow \dots \rightarrow (A_{n-1} \rightarrow A_n) \dots)).$$

That is, we use association to the right (here \equiv denotes syntactic equality).

- (ii) $\alpha, \beta, \gamma, \dots$ denote arbitrary elements of \mathbb{A} .
- (iii) A, B, C, \dots denote arbitrary elements of \mathbb{T} .

Remember the untyped lambda calculus denoted by λ , see e.g. B[1984]¹. It consists of a set of terms Λ defined by the following abstract syntax.

$$\begin{array}{lcl} \mathbf{V} & = & x \mid \mathbf{V}' \\ \Lambda & = & \mathbf{V} \mid \lambda \mathbf{V} \Lambda \mid \Lambda \Lambda \end{array}$$

Figure 1.2: Untyped lambda terms

This makes $\mathbf{V} = \{x, x', x'', \dots\} = \{x_0, x_1, x_2, \dots\}$.

- 1.1.3. NOTATION. (i) x, y, z, \dots denote arbitrary term variables.
(ii) M, N, L, \dots denote arbitrary lambda terms.
(iii) $MN_1 \dots N_k \equiv (..(MN_1) \dots N_k)$.
(iv) $\lambda x_1 \dots x_n. M \equiv (\lambda x_1 (..(\lambda x_n (M))..))$.

1.1.4. DEFINITION. On Λ the following equational theory $\lambda\beta\eta$ is defined by the usual equality axiom and rules (reflexivity, symmetry, transitivity, congruence), including congruence with respect to abstraction:

$$M = N \Rightarrow \lambda x. M = \lambda x. N,$$

and the following special axiom(schemes)

$$\begin{array}{ll} (\lambda x. M)N & = M[x := N] & (\beta\text{-rule}) \\ \lambda x. Mx & = M, & \text{if } x \notin \text{FV}(M) \quad (\eta\text{-rule}) \end{array}$$

Figure 1.3: The theory $\lambda\beta\eta$

As is know this theory can be analyzed by a notion of reduction.

1.1.5. DEFINITION. On Λ we define the following notions of reduction

$$\begin{array}{ll} (\lambda x. M)N & \rightarrow M[x := N] & (\beta) \\ \lambda x. Mx & \rightarrow M, & \text{if } x \notin \text{FV}(M) \quad (\eta) \end{array}$$

Figure 1.4: $\beta\eta$ -contraction rules

As usual, see B[1984], these notions of reduction generate the corresponding reduction relations $\rightarrow_\beta, \twoheadrightarrow_\beta, \rightarrow_\eta, \twoheadrightarrow_\eta, \rightarrow_{\beta\eta}$ and $\twoheadrightarrow_{\beta\eta}$. Also there are the corresponding conversion relations $=_\beta, =_\eta$ and $=_{\beta\eta}$. Terms in Λ will often be considered modulo $=_\beta$ or $=_{\beta\eta}$. If we write $M = N$, then we mean $M =_{\beta\eta} N$ by default. (In B[1984] the default was $=_\beta$.)

1.1.6. PROPOSITION. For all $M, N \in \Lambda$ one has

$$\vdash_{\lambda\beta\eta} M = N \iff M =_{\beta\eta} N.$$

¹This is an abbreviation for the reference Barendregt [1984].

PROOF. See B[1984], Proposition 3.3.2. ■

One reason why the analysis in terms of the notion of reduction $\beta\eta$ is useful is that the following holds.

1.1.7. THEOREM (Church-Rosser Theorem for $\lambda\beta\eta$). *For the notions of reduction \rightarrow_{β} and $\rightarrow_{\beta\eta}$ one has the following.*

(i) *Let $M, N \in \Lambda$. Then*

$$M =_{\beta(\eta)} N \Rightarrow \exists Z \in \Lambda. M \rightarrow_{\beta(\eta)} Z \ \& \ N \rightarrow_{\beta(\eta)} Z.$$

(ii) *Let $M, N_1, N_2 \in \Lambda$. Then*

$$M \rightarrow_{\beta(\eta)} N_1 \ \& \ M \rightarrow_{\beta(\eta)} N_2 \Rightarrow \exists Z \in \Lambda. N_1 \rightarrow_{\beta(\eta)} Z \ \& \ N_2 \rightarrow_{\beta(\eta)} Z.$$

PROOF. (i) See Theorems 3.2.8 and 3.3.9 in B[1984].

(ii) By (i). ■

1.1.8. DEFINITION ($\lambda_{\rightarrow}^{\text{Cu}}$). (i) A *(type assignment) statement* is of the form

$$M : A,$$

with $M \in \Lambda$ and $A \in \mathbb{T}$. This statement is pronounced as ‘ M in A ’. The type A is the *predicate* and the term M is the *subject* of the statement.

(ii) A *declaration* is a statement with as subject a term variable.

(iii) A *basis* is a set of declarations with distinct variables as subjects.

(iv) A statement $M:A$ is *derivable from* a basis Γ , notation

$$\Gamma \vdash_{\lambda_{\rightarrow}}^{\text{Cu}} M:A$$

(or $\Gamma \vdash_{\lambda_{\rightarrow}} M : A$, $\Gamma \vdash^{\text{Cu}} M : A$ or even $\Gamma \vdash M:A$ if there is little danger of confusion) if $\Gamma \vdash M:A$ can be produced by the following rules.

$$(x:A) \in \Gamma \Rightarrow \Gamma \vdash x : A;$$

$$\Gamma \vdash M : (A \rightarrow B), \ \Gamma \vdash N : A \Rightarrow \Gamma \vdash (MN) : B;$$

$$\Gamma, x:A \vdash M : B \Rightarrow \Gamma \vdash (\lambda x.M) : (A \rightarrow B).$$

These rules are usually written as follows.

(axiom)	$\Gamma \vdash x : A,$	if $(x:A) \in \Gamma;$
$(\rightarrow\text{-elimination})$	$\frac{\Gamma \vdash M : (A \rightarrow B) \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : B};$	
$(\rightarrow\text{-introduction})$	$\frac{\Gamma, x:A \vdash M : B}{\Gamma \vdash (\lambda x.M) : (A \rightarrow B)}.$	

Figure 1.5: The system $\lambda_{\rightarrow}^{\text{Cu}}$ of type assignment *à la* Curry

This is the modification to the lambda calculus of the system in Curry [1934], as developed in Curry et al. [1958].

NOTATION. Another way of writing these rules is sometimes found in the literature.

Introduction rule	$\frac{\begin{array}{c} x:A \\ \vdots \\ M:B \end{array}}{\lambda x.M : (A \rightarrow B)}$
Elimination rule	$\frac{M : (A \rightarrow B) \quad N : A}{MN : B}$

$\lambda_{\rightarrow}^{\text{Cu}}$ alternative version

In this version the axiom is considered as implicit and is not notated. The notation

$$\begin{array}{c} x:A \\ \vdots \\ M:B \end{array}$$

denotes that $M : B$ can be derived from $x:A$. Striking through $x:A$ means that for the conclusion $\lambda x.M : A \rightarrow B$ the assumption $x:A$ is no longer needed; it is *discharged*.

1.1.9. DEFINITION. Let $\Gamma = \{x_1:A_1, \dots, x_n:A_n\}$. Then

- (i) $\text{dom}(\Gamma) = \{x_1, \dots, x_n\}$.
- (ii) $x_1:A_1, \dots, x_n:A_n \vdash M : A$ denotes $\Gamma \vdash M : A$.
- (iii) In particular $\vdash M : A$ stands for $\emptyset \vdash M : A$.
- (iv) $x_1, \dots, x_n:A \vdash M : B$ stands for $x_1:A, \dots, x_n:A \vdash M : B$.

1.1.10. EXAMPLE. (i) $\vdash (\lambda xy.x) : (A \rightarrow B \rightarrow A)$ for all $A, B \in \mathbb{T}$.

We will use the notation of version 1 of λ_{\rightarrow} for a derivation of this statement.

$$\frac{\frac{x:A, y:B \vdash x : A}{x:A \vdash (\lambda y.x) : B \rightarrow A}}{\vdash (\lambda x \lambda y.x) : A \rightarrow B \rightarrow A}$$

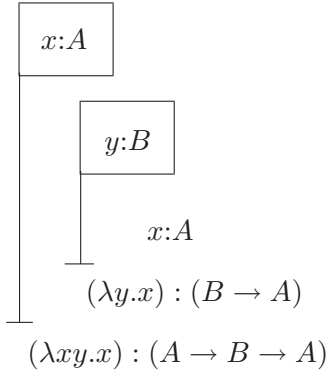
Note that $\lambda xy.x \equiv \lambda x \lambda y.x$ by definition.

(ii) A *natural deduction* derivation (for the alternative version of the system) of the same type assignment is the following.

$$\frac{\frac{\frac{x:A \quad 2 \quad y:B \quad 1}{x:A}}{(\lambda y.x) : (B \rightarrow A)} 1}{(\lambda xy.x) : (A \rightarrow B \rightarrow A)} 2$$

The indices 1 and 2 are bookkeeping devices that indicate at which application of a rule a particular assumption is being discharged.

(iii) A more explicit way of dealing with cancellations of statements is the ‘flag-notation’ used by Fitch (1952) and in the languages AUTOMATH of de Bruijn (1980). In this notation the above derivation becomes as follows.



As one sees, the bookkeeping of cancellations is very explicit; on the other hand it is less obvious how a statement is derived from previous statements in case applications are used.

(iv) Similarly one can show for all $A \in \mathbb{T}$

$$\vdash (\lambda x.x) : (A \rightarrow A).$$

(v) An example with a non-empty basis is $y:A \vdash (\lambda x.x)y : A$.

In the rest of this chapter and in fact in the rest of this book we usually will introduce systems of typed lambda calculi in the style of the first variant of λ_{\rightarrow} .

1.1.11. DEFINITION. Let Γ be a basis and $A \in \mathbb{T}$. Then

- (i) $\Lambda_{\rightarrow}^{\Gamma}(A) = \{M \in \Lambda \mid \Gamma \vdash_{\lambda_{\rightarrow}} M : A\}.$
- (ii) $\Lambda_{\rightarrow}^{\Gamma} = \bigcup_{A \in \mathbb{T}} \Lambda_{\rightarrow}^{\Gamma}(A).$
- (iii) $\Lambda_{\rightarrow}(A) = \Lambda_{\rightarrow}^{\emptyset}(A).$
- (iv) $\Lambda_{\rightarrow} = \Lambda_{\rightarrow}^{\emptyset}$

1.1.12. DEFINITION. Let Γ be a basis, $A \in \mathbb{T}$ and $M \in \Lambda$. Then

- (i) If $M \in \Lambda_{\rightarrow}(A)$, then we say that

M has type A or A is inhabited by M.
- (ii) If $M \in \Lambda_{\rightarrow}$, then M is called *typable*.
- (iii) If $M \in \Lambda_{\rightarrow}^{\Gamma}(A)$, then M has type A relative to Γ .
- (iv) If $M \in \Lambda_{\rightarrow}^{\Gamma}$, then M is called *typeable relative to Γ* .
- (v) If $\Lambda_{\rightarrow}^{(\Gamma)}(A) \neq \emptyset$, then A is *inhabited (relative to Γ)*.

1.1.13. EXAMPLE. We have

$$\begin{aligned} K &\in \Lambda_{\rightarrow}^{\emptyset}(A \rightarrow B \rightarrow A); \\ Kx &\in \Lambda_{\rightarrow}^{\{x:A\}}(B \rightarrow A). \end{aligned}$$

1.1.14. DEFINITION. Let $A \in \mathbb{T}(\lambda_{\rightarrow})$.

- (i) The *depth* of A , notation $\text{dpt}(A)$, is defined as follows.

$$\begin{aligned} \text{dpt}(\alpha) &= 0 \\ \text{dpt}(A \rightarrow B) &= \max\{\text{dpt}(A), \text{dpt}(B)\} + 1 \end{aligned}$$

- (ii) The *rank* of A , notation $\text{rk}(A)$, is defined as follows.

$$\begin{aligned} \text{rk}(\alpha) &= 0 \\ \text{rk}(A \rightarrow B) &= \max\{\text{rk}(A) + 1, \text{rk}(B)\} \end{aligned}$$

- (iii) The *order* of A , notation $\text{ord}(A)$, is defined as follows.

$$\begin{aligned} \text{ord}(\alpha) &= 1 \\ \text{ord}(A \rightarrow B) &= \max\{\text{ord}(A) + 1, \text{ord}(B)\} \end{aligned}$$

- (iv) The depth (rank or order) of a basis Γ is

$$\max_i \{\text{dpt}(A_i) \mid (x_i:A_i) \in \Gamma\},$$

(similarly for the rank and order, respectively). Note that $\text{ord}(A) = \text{rk}(A) + 1$.

1.1.15. DEFINITION. For $A \in \mathbb{T}$ we define $A^k \rightarrow B$ by recursion on k :

$$\begin{aligned} A^0 \rightarrow B &= B; \\ A^{k+1} \rightarrow B &= A \rightarrow A^k \rightarrow B. \end{aligned}$$

Note that $\text{rk}(A^k \rightarrow B) = \text{rk}(A \rightarrow B)$, for all $k > 0$.

Several properties can be proved by induction on the depth of a type. This holds for example for Lemma 1.1.18(i).

The asymmetry in the definition of rank is intended because e.g. a type like $(o \rightarrow o) \rightarrow o$ is more complex than $o \rightarrow o \rightarrow o$, as can be seen by looking to the inhabitants of these types: functionals with functions as arguments versus binary function. Sometimes one uses instead of ‘rank’ the name *type level*. This notion will turn out to be used most of the times.

In logically motivated papers one finds the notion $\text{ord}(A)$. The reason is that in first-order logic one deals with domains and their elements. In second order logic one deals with functions between first-order objects. In this terminology 0-th order logic can be identified with propositional logic.

The minimal and maximal systems λ_{\rightarrow}^o and $\lambda_{\rightarrow}^{\infty}$

The collection \mathbb{A} of type variables serves as set of base types from which other types are constructed. We have $\mathbb{T}_o = \{o\}$ with just one type atom and $\mathbb{T}_{\infty} = \{\alpha_0, \alpha_1, \alpha_2, \dots\}$ with infinitely many of them. These two sets of atoms and their resulting type systems play a major role in this Part I of the book.

1.1.16. DEFINITION. We define the following systems of type assignment.

- (i) $\lambda_{\rightarrow}^o = \lambda_{\rightarrow}^{\mathbb{T}_o}$. This system is also called λ^{τ} in the literature.
- (ii) $\lambda_{\rightarrow}^{\infty} = \lambda_{\rightarrow}^{\mathbb{T}_{\infty}}$.

If it becomes necessary to distinguish the set of atomic types, will use notations like $\Lambda_o(A) = \Lambda_{\mathbb{T}_o}(A)$ and $\Lambda_{\infty}(A) = \Lambda_{\mathbb{T}_{\infty}}(A)$.

Many of the interesting features of the ‘larger’ λ_{\rightarrow} are already present in the minimal version λ_{\rightarrow}^o . The complexity of λ_{\rightarrow} is already present in λ_{\rightarrow}^o .

1.1.17. DEFINITION. (i) The following types of $\mathbb{T}_o \subseteq \mathbb{T}_{\mathbb{A}}$ are often used.

$$0 = o, \quad 1 = o \rightarrow 0, \quad 2 = (o \rightarrow 0) \rightarrow 0, \quad \dots$$

In general

$$0 = o \text{ and } k + 1 = k \rightarrow 0.$$

Note that $\text{rk}(n) = n$.

- (ii) Define n_k by cases on n .

$$\begin{aligned} o_k &= o; \\ (n+1)_k &= n^k \rightarrow o. \end{aligned}$$

For example

$$\begin{aligned} 1_2 &= o \rightarrow o \rightarrow o; \\ 2_3 &= 1 \rightarrow 1 \rightarrow 1 \rightarrow o. \end{aligned}$$

Notice that $\text{rk}(n_k) = \text{rk}(n)$, for $k > 0$.

1.1.18. LEMMA. (i) Every type A of $\lambda_{\rightarrow}^{\infty}$ is of the form

$$A = A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n \rightarrow \alpha.$$

(ii) Every type A of λ_{\rightarrow}^o is of the form

$$A = A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n \rightarrow o.$$

(iii) $\text{rk}(A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n \rightarrow \alpha) = \max\{\text{rk}(A_i) + 1 \mid 1 \leq i \leq n\}.$

PROOF. (i) By induction on the structure (depth) of A . If $A = \alpha$, then this holds for $n = 0$. If $A = B \rightarrow C$, then by the induction hypothesis one has $C = C_1 \rightarrow \dots \rightarrow C_n \rightarrow \gamma$. Hence $A = B \rightarrow C_1 \rightarrow \dots \rightarrow C_n \rightarrow \gamma$.

(ii) By (i).

(iii) By induction on n . ■

1.1.19. NOTATION. Let $A \in \mathbb{T}_{\mathbb{A}}$ and suppose $A = A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n \rightarrow \alpha$. Then the A_i are called the *components* of A . We write

$$\begin{aligned} \text{arity}(A) &= n, \\ A(i) &= A_i, \quad \text{for } 1 \leq i \leq n; \\ \text{target}(A) &= \alpha. \end{aligned}$$

Iterated components are denoted as follows

$$A(i, j) = A(i)(j).$$

Different versions of $\lambda_{\rightarrow}^{\mathbb{A}}$

The system $\lambda_{\rightarrow}^{\mathbb{A}}$ that was introduced in Definition 1.1.8 assigns types to untyped lambda terms. These system will be referred to as the Curry system and be denoted by $\lambda_{\rightarrow, \text{Cu}}^{\mathbb{A}}$ or $\lambda_{\rightarrow}^{\text{Cu}}$, as the set \mathbb{A} often does not need to be specified. There will be introduced two variants of $\lambda_{\rightarrow}^{\mathbb{A}}$.

The first variant of $\lambda_{\rightarrow}^{\text{Cu}}$ is the *Church* version of $\lambda_{\rightarrow}^{\mathbb{A}}$, denoted by $\lambda_{\rightarrow, \text{Ch}}^{\mathbb{A}}$ or $\lambda_{\rightarrow}^{\text{Ch}}$. In this theory the types are assigned to embellished terms in which the variables (free and bound) come with types attached. For example the Curry style type assignments

$$\vdash_{\lambda_{\rightarrow}}^{\text{Cu}} (\lambda x.x) : A \rightarrow A \quad (1_{\text{Cu}})$$

$$y:A \vdash_{\lambda_{\rightarrow}}^{\text{Cu}} (\lambda x.xy) : (A \rightarrow B) \rightarrow A \rightarrow B \quad (2_{\text{Cu}})$$

now becoming

$$(\lambda x^A.x^A) \in \Lambda_{\rightarrow}^{\text{Ch}}(A \rightarrow A) \quad (1_{\text{Ch}})$$

$$(\lambda x^{A \rightarrow B}.x^{A \rightarrow B}y^A) : \Lambda_{\rightarrow}^{\text{Ch}}((A \rightarrow B) \rightarrow A \rightarrow B) \quad (2_{\text{Ch}})$$

The second variant of $\lambda_{\rightarrow}^{\text{Cu}}$ is the *de Bruijn* version of $\lambda_{\rightarrow}^{\text{A}}$, denoted by $\lambda_{\rightarrow, \text{dB}}^{\text{A}}$ or $\lambda_{\rightarrow}^{\text{dB}}$. Now only bound variables get ornamented with types, but only at the binding stage. The examples (1), (2) now become

$$\begin{array}{ll} \vdash_{\lambda_{\rightarrow}^{\text{dB}}} (\lambda x : A.x) : A \rightarrow A & (1_{\text{dB}}) \\ y:A \vdash_{\lambda_{\rightarrow}^{\text{dB}}} (\lambda x : (A \rightarrow B).xy) : (A \rightarrow B) \rightarrow A \rightarrow B & (2_{\text{dB}}) \end{array}$$

The reasons to have these variants will be explained later in Section 1.4. In the meantime we will work intuitively.

1.1.20. NOTATION. Terms like $(\lambda f x.f(fx)) \in \Lambda^{\emptyset}(1 \rightarrow o \rightarrow o)$ will often be written

$$\lambda f^1 x^0.f(fx)$$

to indicate the types of the bound variables. We will come back to this notational issue in section 1.4.

1.2. Normal inhabitants

In this section we will give an algorithm that enumerates the set of closed terms in normal form of a given type $A \in \mathbb{T}$. Since we will prove in the next chapter that all typable terms do have a nf and that reduction preserves typing, we thus have an enumeration of essentially all closed terms of that given type. We do need to distinguish various kinds of nf's.

1.2.1. DEFINITION. Let $A = A_1 \rightarrow \dots A_n \rightarrow \alpha$ and suppose $\Gamma \vdash M : A$.

(i) Then M is in long-nf, notation lnf , if $M \equiv \lambda x_1^{A_1} \dots x_n^{A_n}.x M_1 \dots M_n$ and each M_i is in lnf . By induction on the depth of the type of the closure of M one sees that this definition is well-founded.

(ii) M has a lnf if $M =_{\beta\eta} N$ and N is a lnf .

In Exercise 1.5.16 it is proved that if M has a β -nf, which according to Theorem 2.2.4 is always the case, then it also has a unique lnf and will be its unique $\beta\eta^{-1}$ nf. Here η^{-1} is the notion of reduction that is the converse of η .

1.2.2. EXAMPLES. (i) Note that $\lambda f^1.f =_{\beta\eta} \lambda f^1 \lambda x^o.f x$ and that $\lambda f^1.f$ is a $\beta\eta$ -nf but not a lnf .

(ii) $\lambda f^1 \lambda x^o.f x$ is a lnf , but not a $\beta\eta$ -nf.

(iii) $\lambda x:o.x$ is both in $\beta\eta$ -nf and lnf .

(iv) The β -nf $\lambda F:2_2 \lambda f:1.F f(\lambda x:o.f x)$ is neither in $\beta\eta$ -nf nor lnf .

(v) A variable of atomic type α is a lnf , but of type $A \rightarrow B$ not.

(vi) A variable $f : 1 \rightarrow 1$ has as lnf $\lambda g^1 \lambda x^o.f(\lambda y^o.g y)x$.

1.2.3. PROPOSITION. Every β -nf M has a lnf M^ℓ such that $M^\ell \twoheadrightarrow_{\eta} M$.

PROOF. Define M^ℓ by induction on the depth of the type of the closure of M as follows.

$$M^\ell \equiv (\lambda \vec{x}.y.M_1 \dots M_n)^\ell = \lambda \vec{x}\vec{z}.y.M_1^\ell \dots M_n^\ell \vec{z}^\ell.$$

Then M^ℓ does the job. ■

Now we will define a 2-level grammar for obtaining the collection of all Inf's of a given type A .

1.2.4. DEFINITION. Let $N = \{L(A; \Gamma) \mid A \in \Pi_{\mathbb{A}}; \Gamma \text{ a context of } \lambda_{\rightarrow}\}$. Let Σ be the alphabet of the terms of the $\lambda_{\rightarrow}^{\text{Ch}}$. Define the following two-level grammar, see van Wijngaarden et al. [1976], as a notion of reduction over words over $N \cup \Sigma$. The elements of N are the non-terminals (unlike in a context-free language there are now infinitely many of them).

$$\begin{aligned} L(\alpha; \Gamma) &\Rightarrow xL(B_1; \Gamma) \dots L(B_n; \Gamma), & \text{if } (x:\vec{B} \rightarrow \alpha) \in \Gamma; \\ L(A \rightarrow B; \Gamma) &\Rightarrow \lambda x^A.L(B; \Gamma, x:A). \end{aligned}$$

Typical productions of this grammar are the following.

$$\begin{aligned} L(3; \emptyset) &\Rightarrow \lambda F^2.L(o; F^2) \\ &\Rightarrow \lambda F^2.FL(1; F^2) \\ &\Rightarrow \lambda F^2.F(\lambda x^o.L(o; F^2, x^o)) \\ &\Rightarrow \lambda F^2.F(\lambda x^o.x). \end{aligned}$$

But one has also

$$\begin{aligned} L(o; F^2, x^o) &\Rightarrow FL(1; F^2, x^o) \\ &\Rightarrow F(\lambda x_1^o.L(o; F^2, x^o, x_1^o)) \\ &\Rightarrow F(\lambda x_1^o.x_1). \end{aligned}$$

Hence (\Rightarrow denotes the transitive reflexive closure of \Rightarrow)

$$L(3; \emptyset) \Rightarrow \lambda F^2.F(\lambda x^o.F(\lambda x_1^o.x_1)).$$

In fact, $L(3; \emptyset)$ reduces to all possible closed Inf's of type 3. Like in abstract syntax we do not produce parentheses from the $L(A; \Gamma)$, but write them when needed.

1.2.5. PROPOSITION. *Let Γ, M, A be given. Then*

$$L(A, \Gamma) \Rightarrow M \iff \Gamma \vdash M : A \text{ \& } M \text{ is in Inf.}$$

Now we will modify the 2-level grammar and the inhabitation machines in order to produce all β -nf's.

1.2.6. DEFINITION. The 2-level grammar N is defined as follows.

$$\begin{aligned} N(A; \Gamma) &\Rightarrow xN(B_1; \Gamma) \dots N(B_n; \Gamma), & \text{if } (x:\vec{B} \rightarrow A) \in \Gamma; \\ N(A \rightarrow B; \Gamma) &\Rightarrow \lambda x^A. N(B; \Gamma, x:A). \end{aligned}$$

Now the β -nf's are being produced. As an example we make the following production. Remember that $1 = o \rightarrow o$.

$$\begin{aligned} L(1 \rightarrow o \rightarrow o; \emptyset) &\Rightarrow \lambda f^1. L(o \rightarrow o; f: o \rightarrow o) \\ &\Rightarrow \lambda f^1. f. \end{aligned}$$

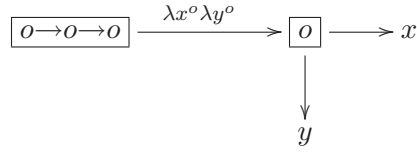
1.2.7. PROPOSITION. Let Γ, M, A be given. Then

$$N(A, \Gamma) \Rightarrow M \iff \Gamma \vdash M : A \text{ \& } M \text{ is in } \beta\text{-nf}.$$

Inhabitation machines

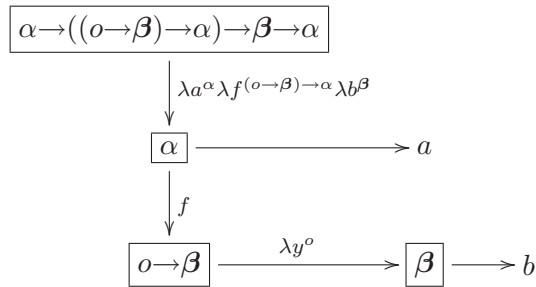
Inspired by this proposition one can introduce for each type A a machine M_A producing the set of closed terms of that type. If one is interested in terms containing variables $x_1^{A_1}, \dots, x_n^{A_n}$, then one can also find these terms by considering the machine for the type $A_1 \rightarrow \dots \rightarrow A_n \rightarrow A$ and look at the subproduction at node A .

1.2.8. EXAMPLES. (i) $A = o \rightarrow o \rightarrow o$. Then M_A is



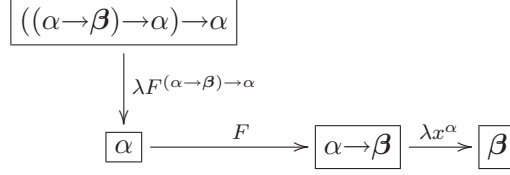
This shows that the type 1_2 has two closed inhabitants: $\lambda xy.x$ and $\lambda xy.y$. We see that the two arrows leaving \boxed{o} represent a choice.

(ii) $A = \alpha \rightarrow ((o \rightarrow \beta) \rightarrow \alpha) \rightarrow \beta \rightarrow \alpha$. Then M_A is



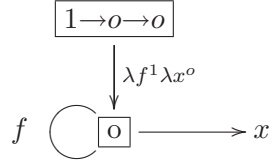
Again there are only two inhabitants, but now the production of them is rather different: $\lambda a f b.a$ and $\lambda a f b.f(\lambda x^o.b)$.

(iii) $A = ((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha$. Then M_A is



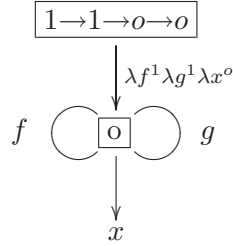
This type, corresponding to Peirce's law, does not have any inhabitants.

(iv) $A = 1 \rightarrow o \rightarrow o$. Then M_A is



This is the type **Nat** having the Church's numerals $\lambda f^1 x^o . f^n x$ as inhabitants.

(v) $A = 1 \rightarrow 1 \rightarrow o \rightarrow o$. Then M_A is

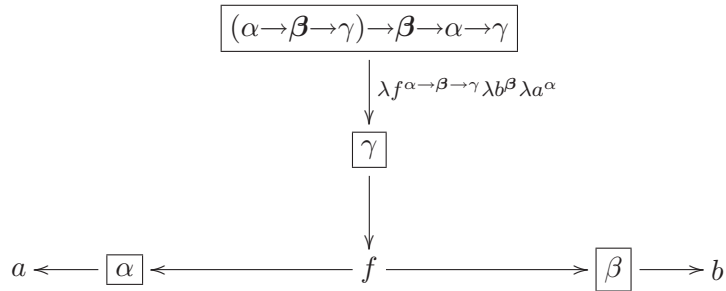


Inhabitants of this type represent words over the alphabet $\Sigma = \{f, g\}$, for example

$$\lambda f^1 g^1 x^o . f g f f g f g g x,$$

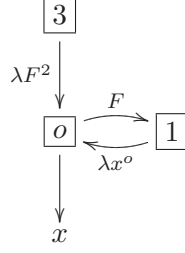
where we have to insert parentheses associating to the right.

(vi) $A = (\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \beta \rightarrow \alpha \rightarrow \gamma$. Then M_A is



giving as term $\lambda f^{\alpha \rightarrow \beta \rightarrow \gamma} \lambda b^\beta \lambda a^\alpha . f a b$. Note the way an interpretation should be given to paths going through f : the outgoing arcs (to $\boxed{\alpha}$ and $\boxed{\beta}$) should be completed both separately in order to give f its two arguments.

(vii) $A = 3$. Then M_A is

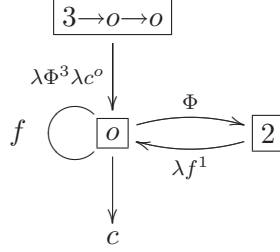


This type 3 has inhabitants having more and more binders:

$$\lambda F^2.F(\lambda x_0^o.F(\lambda x_1^o.F(\dots(\lambda x_n^o.x_i))))).$$

The novel phenomenon that the binder λx^o may go round and round forces us to give new incarnations $\lambda x_0^o, \lambda x_1^o, \dots$ each time we do this (we need a counter to ensure freshness of the bound variables). The ‘terminal’ variable x can take the shape of any of the produced incarnations x_k . As almost all binders are dummy, we will see that this potential infinity of binding is rather innocent and the counter is not yet really needed here.

(viii) $A = 3 \rightarrow o \rightarrow o$. Then M_A is

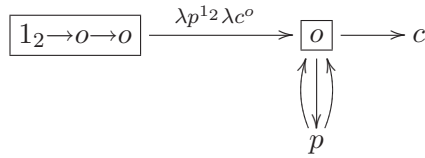


This type, called the *monster* M , does have a potential infinite amount of binding, having as terms e.g.

$$\lambda \Phi^3 c^o . \Phi \lambda f_1^1 . f_1 \Phi \lambda f_2^1 . f_2 f_1 \Phi \dots \lambda f_n^1 . f_n \dots f_2 f_1 c,$$

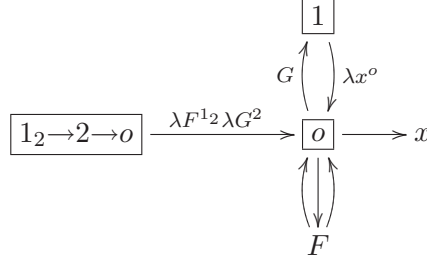
again with inserted parentheses associating to the right. Now a proper bookkeeping of incarnations (of f^1 in this case) becomes necessary, as the f going from \boxed{o} to itself needs to be one that has already been incarnated.

(ix) $A = 1_2 \rightarrow o \rightarrow o$. Then M_A is



This is the type of binary trees, having as elements, e.g. $\lambda p^{1_2} c^o . c$ and $\lambda p^{1_2} c^o . pc(pcc)$. Again, as in example (vi) the outgoing arcs from p (to \boxed{o}) should be completed both separately in order to give p its two arguments.

(x) $A = 1_2 \rightarrow 2 \rightarrow o$. Then M_A is



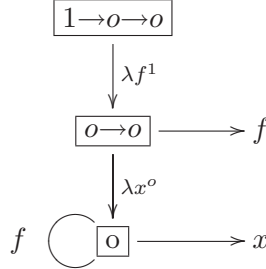
This is the type L corresponding to untyped lambda terms. For example the untyped terms $\omega \equiv \lambda x.xx$ and $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ can be translated to $(\omega)^t \equiv \lambda F^{1_2} G^2. G(\lambda x^o. Fxx)$ and

$$\begin{aligned} (\Omega)^t &\equiv \lambda F^{1_2} G^2. F(G(\lambda x^o. Fxx))(G(\lambda x^o. Fxx)) \\ &=_{\beta} \lambda F G. F((\omega)^t F G)((\omega)^t F G) \\ &=_{\beta} (\omega)^t \cdot_L (\omega)^t, \end{aligned}$$

where for $M, N \in L$ one defines $M \cdot_L N = \lambda F G. F(MFG)(NFG)$. All features of producing terms inhabiting types (bookkeeping bound variables, multiple paths) are present here.

Following the 2-level grammar N one can make inhabitation machines for β -nf M_A^{β} .

1.2.9. EXAMPLE. We show how the production machine for β -nf's differs from the one for lnf's. Let $A = 1 \rightarrow o \rightarrow o$. Then $\lambda f^1. f$ is the (unique) β -nf of type A that is not a lnf. It will come out from the following machine M_A^{β} .



So in order to obtain the β -nf's, one has to allow output at types that are not atomic.

1.3. Representing data types

In this section it will be shown that first order algebraic data types can be represented in λ_{\rightarrow}^o . We start with several examples: Booleans, the natural numbers, the free monoid over n generators (words over a finite alphabet with n elements) and trees with at the leafs labels from a type A . The following definitions depend on a given type A . So in fact $\text{Bool} = \text{Bool}_A$ etcetera. Often one takes $A = o$.

Booleans

1.3.1. DEFINITION. Define

$$\begin{aligned}\text{Bool} &\equiv A \rightarrow A \rightarrow A; \\ \text{true} &\equiv \lambda xy.x; \\ \text{false} &\equiv \lambda xy.y.\end{aligned}$$

Then $\text{true} \in \Lambda(\text{Bool})$ and $\text{false} \in \Lambda(\text{Bool})$.

1.3.2. PROPOSITION. *There are terms not, &, or, imp, iff with the expected behavior on Booleans. For example $\text{not} \in \Lambda(\text{Bool} \rightarrow \text{Bool})$ and*

$$\begin{aligned}\text{not true} &=_{\beta} \text{false}, \\ \text{not false} &=_{\beta} \text{true}.\end{aligned}$$

PROOF. Take $\text{not} \equiv \lambda axy.ayx$ and $\text{or} \equiv \lambda abxy.ax(bxy)$. From these two operations the other Boolean functions can be defined. For example, implication can be represented by

$$\text{imp} \equiv \lambda ab.\text{or}(\text{not } a)b.$$

A shorter representation is $\lambda abxy.a(bxy)x$, the normal form of imp . ■

Natural numbers

Following Church the set of natural numbers can be represented as a type

$$\text{Nat} \equiv (A \rightarrow A) \rightarrow A \rightarrow A.$$

For each natural number $n \in \mathbb{N}$ we define its representation

$$\mathbf{c}_n \equiv \lambda fx.f^n x,$$

where

$$\begin{aligned}f^0 x &\equiv x \\ f^{n+1} x &\equiv f(f^n x).\end{aligned}$$

1.3.3. PROPOSITION. (i) *There exists a term $S^+ \in \Lambda(\text{Nat} \rightarrow \text{Nat})$ such that*

$$S^+ \mathbf{c}_n =_{\beta} \mathbf{c}_{n+1}, \text{ for all } n \in \mathbb{N}.$$

(ii) *There is a term $\text{zero?} \in \Lambda(\text{Nat} \rightarrow \text{Bool})$ such that*

$$\begin{aligned}\text{zero? } \mathbf{c}_0 &=_{\beta} \text{true} \\ \text{zero? } (S^+ x) &=_{\beta} \text{false}.\end{aligned}$$

PROOF. (i) Take $S^+ \equiv \lambda n \lambda f x. f(nfx)$. Then

$$\begin{aligned} S^+ c_n &=_{\beta} \lambda f x. f(c_n f x) \\ &=_{\beta} \lambda f x. f(f^n x) \\ &\equiv \lambda f x. f^{n+1} x \\ &\equiv c_{n+1}. \end{aligned}$$

(ii) Take $\text{zero}_? \equiv \lambda n \lambda a b. n(Kb)a$. Then

$$\begin{aligned} \text{zero}_? c_0 &=_{\beta} \lambda a b. c_0(Kb)a \\ &=_{\beta} \lambda a b. a \\ &\equiv \text{true}; \\ \text{zero}_?(S^+ x) &=_{\beta} \lambda a b. S^+ x(Kb)a \\ &=_{\beta} \lambda a b. (\lambda f y. f(xfy))(Kb)a \\ &=_{\beta} \lambda a b. Kb(x(Kb)a) \\ &=_{\beta} \lambda a b. b \\ &\equiv \text{false}. \blacksquare \end{aligned}$$

Addition and multiplication are definable in λ_{\rightarrow} .

1.3.4. PROPOSITION. (i) *There is a term $\text{plus} \in \Lambda(\text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat})$ satisfying*

$$\text{plus } c_n c_m =_{\beta} c_{n+m}.$$

(ii) *There is a term $\text{times} \in \Lambda(\text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat})$ such that*

$$\text{times } c_n c_m =_{\beta} c_{n \cdot m}$$

PROOF. (i) Take $\text{plus} \equiv \lambda n m \lambda f x. n f(mfx)$. Then

$$\begin{aligned} \text{plus } c_n c_m &=_{\beta} \lambda f x. c_n f(c_m f x) \\ &=_{\beta} \lambda f x. f^n(f^m x) \\ &\equiv \lambda f x. f^{n+m} x \\ &\equiv c_{n+m}. \end{aligned}$$

(ii) Take $\text{times} \equiv \lambda n m \lambda f x. m(\lambda y. n f y)x$. Then

$$\begin{aligned} \text{times } c_n c_m &=_{\beta} \lambda f x. c_m(\lambda y. c_n f y)x \\ &=_{\beta} \lambda f x. c_m(\lambda y. f^n y)x \\ &=_{\beta} \lambda f x. \underbrace{(f^n(f^n(\dots(f^n x)\dots)))}_{m \text{ times}} \\ &\equiv \lambda f x p. f^{n \cdot m} x \\ &\equiv c_{n \cdot m}. \blacksquare \end{aligned}$$

1.3.5. COROLLARY. *For every polynomial $p \in \mathbb{N}[x_1, \dots, x_k]$ there is a closed term $M_p : \Lambda(\mathbf{Nat}^k \rightarrow \mathbf{Nat})$ such that $\forall n_1, \dots, n_k \in \mathbb{N}. M_p \mathbf{c}_{n_1} \dots \mathbf{c}_{n_k} =_{\beta} \mathbf{c}_{p(n_1, \dots, n_k)}$. ■*

From the results obtained so far it follows that the polynomials extended by case distinctions (being equal or not to zero) are definable in λ_{\rightarrow} . In Statman [1976] or Schwichtenberg [1976] it is proved that exactly these so-called extended polynomials are definable in λ_{\rightarrow} . Hence primitive recursion cannot be defined in λ_{\rightarrow} ; in fact not even the predecessor function, see Proposition 2.4.22.

Words over a finite alphabet

Let $\Sigma = \{a_1, \dots, a_k\}$ be a finite alphabet. Then Σ^* the collection of words over Σ can be represented in λ_{\rightarrow} .

1.3.6. DEFINITION. (i) The type for words in Σ^* is

$$\mathbf{Sigma}^* \equiv (o \rightarrow o)^k \rightarrow o \rightarrow o.$$

(ii) Let $w = a_{i_1} \dots a_{i_p}$ be a word. Define

$$\begin{aligned} \underline{w} &\equiv \lambda a_1 \dots a_k x. a_{i_1} (\dots (a_{i_p} x) \dots) \\ &\equiv \lambda a_1 \dots a_k x. (a_{i_1} \circ \dots \circ a_{i_p}) x. \end{aligned}$$

Note that $\underline{\epsilon} \in \Lambda(\mathbf{Sigma}^*)$. If ϵ is the empty word $()$, then naturally

$$\begin{aligned} \underline{\epsilon} &\equiv \lambda a_1 \dots a_k x. x \\ &\equiv \mathbf{K}^k \mathbf{I}. \end{aligned}$$

Now we show that the operation concatenation can be defined in λ_{\rightarrow} .

1.3.7. PROPOSITION. *There exists a term $\text{concat} \in \Lambda(\mathbf{Sigma}^* \rightarrow \mathbf{Sigma}^* \rightarrow \mathbf{Sigma}^*)$ such that for all $w, v \in \Sigma^*$*

$$\text{concat } \underline{w} \underline{v} = \underline{wv}.$$

PROOF. Define

$$\text{concat} \equiv \lambda w v. \vec{a} x. w \vec{a} (v \vec{a} x).$$

Then the type is correct and the definition equation holds. ■

1.3.8. PROPOSITION. (i) *There exists a term $\text{empty} \in \Lambda(\mathbf{Sigma}^*)$ such that*

$$\begin{aligned} \text{empty } \underline{\epsilon} &= \text{true} \\ \text{empty } \underline{w} &= \text{false} \quad \text{if } w \neq \epsilon. \end{aligned}$$

(ii) Given a (represented) word $w_0 \in \Lambda(\text{Sigma}^*)$ and a term $G \in \Lambda(\text{Sigma}^* \rightarrow \text{Sigma}^*)$ there exists a term $F \in \Lambda(\text{Sigma}^* \rightarrow \text{Sigma}^*)$ such that

$$\begin{aligned} F \underline{\epsilon} &= w_0; \\ F \underline{w} &= G\underline{w}, \quad \text{if } w \neq \epsilon. \end{aligned}$$

PROOF. (i) Take $\text{empty} \equiv \lambda wpq.w(\text{K}q) \sim^k p$.

(ii) Take $F \equiv \lambda w \lambda x \vec{a}. \text{empty} w(w_0 \vec{a} x)(G w \vec{a} x)$. ■

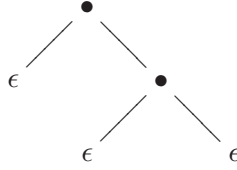
One cannot define a terms ‘car’ or ‘cdr’ such that $\text{car } \underline{aw} = w$ and $\text{cdr } \underline{aw} = w$.

Trees

1.3.9. DEFINITION. The set of binary trees, notation T_2 , is defined by the following abstract syntax

$$t = \epsilon \mid p(t, t)$$

Here ϵ is the ‘empty tree’ and P is the constructor that puts two trees together. For example $p(\epsilon, p(\epsilon, \epsilon)) \in T_2$ can be depicted as



Now we will represent T_2 as a type in λ_{\rightarrow} .

1.3.10. DEFINITION. (i) The set T_2 will be represented by the type

$$\top_2 \equiv (o^2 \rightarrow o) \rightarrow o \rightarrow o.$$

(ii) Define for $t \in T_2$ its representation \underline{t} inductively as follows.

$$\begin{aligned} \underline{\epsilon} &= \lambda pe.e; \\ \underline{p(t, s)} &= \lambda p(tpe)(spe). \end{aligned}$$

(iii) Write

$$\begin{aligned} E &= \lambda pe.e; \\ P &= \lambda tspe.p(tpe)(spe). \end{aligned}$$

Note that Note that for $t \in T_2$ one has $\underline{t} \in \Lambda(\top_2)$

The following follows immediately from this definition.

1.3.11. PROPOSITION. The map $_ : T_2 \rightarrow \top_2$ can be defined inductively as follows

$$\begin{aligned}\epsilon &= E; \\ \underline{p(t, s)} &= P \underline{t} \underline{s}.\end{aligned}$$

Interesting functions, like the one that selects one of the two branches of a tree cannot be defined in λ_{\rightarrow} .

The type \top_2 will play an important role in Section 3.4.

Representing Free algebras with a handicap

Now we will see that all the examples are special cases of a general construction. It turns out that first order algebraic data types \mathcal{A} can be represented in λ_{\rightarrow}^0 . The representations are said to have a handicap because not all primitive recursive functions on \mathcal{A} are representable. Mostly the destructors cannot be represented. In a special cases one can do better. Every finite algebra can be represented with all possible functions on them. Pairing with projections can be represented.

1.3.12. DEFINITION. (i) An *algebra* is a set A with a specific finite set of operators of different arity:

$$\begin{aligned}c_1, c_2, \dots &\in A && \text{(constants, we may call these 0-ary functions);} \\ f_1, f_2, \dots &\in A \rightarrow A && \text{(unary functions);} \\ g_1, g_2, \dots &\in A^2 \rightarrow A && \text{(binary function);} \\ &\dots && \\ h_1, h_2, \dots &\in A^n \rightarrow A && \text{(n-ary functions).}\end{aligned}$$

(ii) An n-ary function $k : A^n \rightarrow A$ is called *algebraic* iff k can be defined explicitly from the constructors. For example

$$k = \lambda a_1 a_2. g_1(a_1, (g_2(h_1(a_2), c_2)))$$

is a binary algebraic function.

(iii) An element a of A is called *algebraic* iff a is an algebraic 0-ary function. Algebraic elements of A can be denoted by first-order terms over the algebra.

(iv) The algebra A is *free (ly generated)* if every element of A is algebraic and moreover if for two first-order terms t, s one has

$$t = s \Rightarrow t \equiv s.$$

In a free algebra the given operators are called *constructors*.

For example \mathbb{N} with constructors $0, s$ (s is the successor) is a free algebra. But \mathbb{Z} with $0, s, p$ (p is the predecessor) is not free. Indeed, $0 = p(s(0))$, but $0 \not\equiv p(s(0))$ as syntactic expressions.

1.3.13. THEOREM. For a free algebra \mathcal{A} there is a type $\underline{\mathcal{A}} \in \mathbb{T}_o$ and a map $\lambda a. \underline{a} : \mathcal{A} \rightarrow \Lambda(\underline{\mathcal{A}})$ satisfying the following.

- (i) \underline{a} is a lnf, for every $a \in \mathcal{A}$.
- (ii) $\underline{a} =_{\beta\eta} \underline{b} \iff a = b$.
- (iii) $\Lambda(\underline{\mathcal{A}}) = \{\underline{a} \mid a \in \mathcal{A}\}$, up to $\beta\eta$ -conversion.
- (iv) For k -ary algebraic functions f on \mathcal{A} there is an $\underline{f} \in \Lambda(\underline{\mathcal{A}} \rightarrow \underline{\mathcal{A}})$ such that

$$\underline{f} \underline{a}_1 \dots \underline{a}_k = \underline{f(a_1, \dots, a_k)}.$$

(v) There is a representable discriminator distinguishing between elements of the form $c, f_1(a), f_2(a, b), \dots, f_n(a_1, \dots, a_n)$. More precisely, there is a term $\text{test} \in \Lambda(\underline{\mathcal{A}} \rightarrow \underline{\mathbb{N}})$ such that for all $a, b \in \mathcal{A}$

$$\begin{aligned} \text{test } \underline{c} &= \mathbf{c}_0; \\ \text{test } \underline{f_1(a)} &= \mathbf{c}_1; \\ \text{test } \underline{f_2(a, b)} &= \mathbf{c}_2 \\ &\vdots \\ \text{test } \underline{f_n(\text{vectan})} &= \mathbf{c}_n \end{aligned}$$

PROOF. We show this by a representative example. Let \mathcal{A} be freely generated by, say, the 0-ary constructor c , the 1-ary constructor f and the 2-ary constructor g . Then an element like

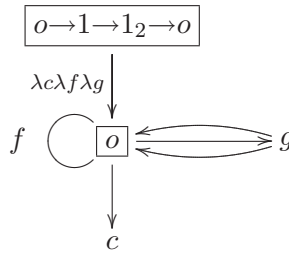
$$a = g(c, f(c))$$

is represented by

$$\underline{a} = \lambda c f g. g c (f c) \in \Lambda(o \rightarrow 1 \rightarrow 1_2 \rightarrow o).$$

Taking $\underline{\mathcal{A}} = o \rightarrow 1 \rightarrow 1_2 \rightarrow o$ we will verify the claims. First realize that \underline{a} is constructed from a via $a^\sim = g c (f c)$ and then taking the closure $\underline{a} = \lambda c f g. a^\sim$.

- (i) Clearly the \underline{a} are in lnf.
- (ii) If a and b are different, then their representations $\underline{a}, \underline{b}$ are different lns, hence $\underline{a} \neq_{\beta\eta} \underline{b}$.
- (iii) The inhabitation machine $M_{\underline{\mathcal{A}}} = M_{o \rightarrow 1 \rightarrow 1_2 \rightarrow o}$ looks like



It follows that for every $M \in \Lambda(\underline{\mathcal{A}})$ one has $M =_{\beta\eta} \lambda c f g. a^\sim = \underline{a}$ for some $a \in \mathcal{A}$. This shows that $\Lambda(\underline{\mathcal{A}}) \subseteq \{\underline{a} \mid a \in \mathcal{A}\}$. The converse inclusion is trivial. In the general case (for other data types \mathcal{A}) one has that $\text{rk}(\underline{\mathcal{A}}) = 2$. Hence the lnf inhabitants of $\Lambda(\underline{\mathcal{A}})$ have the form $\lambda c f g. P$, where P is a combination of the variables c, f, g . This means that the corresponding inhabitation machine is similar and the argument goes through in general.

(iv) An algebraic function is explicitly defined from the constructors. We first define representations for the constructors.

$$\begin{aligned}\underline{c} &= \lambda c f g. c && : \underline{A}; \\ \underline{f} &= \lambda a c f g. f(a c f g) && : \underline{A} \rightarrow \underline{A}; \\ \underline{g} &= \lambda a b c f g. g(a c f g)(b c f g) && : \underline{A}^2 \rightarrow \underline{A}.\end{aligned}$$

$$\begin{aligned}\text{Then } \underline{f} \underline{a} &= \lambda c f g. f(a c f g) \\ &= \lambda c f g. f(a^\sim) \\ &\equiv \lambda c f g. (f(a))^\sim, \text{ (tongue in cheek),} \\ &\equiv \underline{f}(\underline{a}).\end{aligned}$$

Similarly one has $\underline{g} \underline{a} \underline{b} = \underline{g}(\underline{a}, \underline{b})$.

Now if e.g. $h(a, b) = g(a, f(b))$, then we can take

$$\underline{h} \equiv \lambda a b. \underline{g} \underline{a} (\underline{f} \underline{b}) : \underline{A}^2 \rightarrow \underline{A}.$$

Then clearly $\underline{h} \underline{a} \underline{b} = \underline{h}(\underline{a}, \underline{b})$.

(v) Take $\text{test} \equiv \lambda a f c. a(\mathbf{c}_0 f c)(\lambda x. \mathbf{c}_1 f c)(\lambda x y. \mathbf{c}_2 f c)$. ■

1.3.14. DEFINITION. The notion of free algebra can be generalized to a free *multi-sorted* algebra. We do this by giving an example. The collection of lists of natural numbers, notation $L_{\mathbb{N}}$ can be defined by the 'sorts' \mathbb{N} and $L_{\mathbb{N}}$ and the constructors

$$\begin{aligned}0 &\in \mathbb{N}; \\ s &\in \mathbb{N} \rightarrow \mathbb{N}; \\ \text{nil} &\in L_{\mathbb{N}}; \\ \text{cons} &\in \mathbb{N} \rightarrow L_{\mathbb{N}} \rightarrow L_{\mathbb{N}}.\end{aligned}$$

In this setting the list $[0, 1] \in L_{\mathbb{N}}$ is

$$\text{cons}(0, \text{cons}(s(0), \text{nil})).$$

More interesting multisorted algebras can be defined that are 'mutually recursive', see Exercise 1.5.15.

1.3.15. COROLLARY. *Every freely generated multi-sorted first-order algebra can be represented in a way similar to that in Theorem 1.3.13.*

PROOF. Similar to that of the Theorem. ■

Finite Algebras

For finite algebras one can do much better.

1.3.16. THEOREM. *For every finite set $A = \{a_1, \dots, a_n\}$ there exists a type $\underline{A} \in \mathbb{T}_o$ and elements $\underline{a}_1, \dots, \underline{a}_n \in \Lambda(\underline{A})$ such that the following holds.*

- (i) $\Lambda(\underline{A}) = \{\underline{a} \mid a \in A\}$.
- (ii) For all k and $f : A^k \rightarrow A$ there exists an $\underline{f} \in \Lambda(\underline{A}^k \rightarrow \underline{A})$ such that

$$\underline{f} \underline{b}_1 \dots \underline{b}_k = \underline{f(b_1, \dots, b_k)}.$$

PROOF. Take $\underline{A} = 1_n = o^n \rightarrow o$ and $\underline{a}_i = \lambda b_1 \dots b_n. b_i \in \Lambda(1_n)$.

- (i) By a simple argument using the inhabitation machine M_{1_n} .
- (ii) By induction on k . If $k = 0$, then f is an element of A , say $f = a_i$. Take $\underline{f} = \underline{a}_i$. Now suppose we can represent all k -ary functions. Given $f : A^{k+1} \rightarrow A$, define for $b \in A$

$$f_b(b_1, \dots, b_k) = f(b, b_1, \dots, b_k).$$

Each f_b is a k -ary function and has a representative \underline{f}_b . Define

$$\underline{f} = \lambda b \vec{b}. b(\underline{f}_{a_1} \vec{b}) \dots (\underline{f}_{a_n} \vec{b}),$$

where $\vec{b} = b_2, \dots, b_{k+1}$. Then

$$\begin{aligned} \underline{f} \underline{b}_1 \dots \underline{b}_{k+1} &= \underline{b}_1 (\underline{f}_{a_1} \vec{b}) \dots (\underline{f}_{a_n} \vec{b}) \\ &= \underline{f_{b_1} b_2 \dots b_{k+1}} \\ &= \underline{f_{b_1}(b_2, \dots, b_{k+1})}, && \text{by the induction hypothesis,} \\ &= \underline{f(b_1, \dots, b_{k+1})}, && \text{by definition of } f_{b_1}. \blacksquare \end{aligned}$$

One even can faithfully represent the full type structure over A as closed terms of λ_{\rightarrow}^o , see Exercise ??.

Examples as free or finite algebras

The examples in the beginning of this section all can be viewed as free or finite algebras. The Booleans form a finite set and its representation is type 1_2 . For this reason all Boolean functions can be represented. The natural numbers \mathbb{N} and the trees T are examples of free algebras with a handicapped representation. Words over a finite alphabet $\Sigma = \{a_1, \dots, a_n\}$ can be seen as an algebra with constant ϵ and further constructors $f_{a_i} = \lambda w. a_i w$. The representations given are particular cases of the theorems about free and finite algebras.

Pairing

In the untyped lambda calculus arbitrary two elements can be put in a pair

$$[M, N] \equiv \lambda z. z M N.$$

From the pairs the components can be retrieved:

$$[M_1, M_2](\lambda x_1 x_2. x_i) = M_i.$$

As a handicapped representation this works. But if we want to retrieve the elements from a pair of terms of different types, then this representation does not work for λ_o^o , because no proper type can be given to z in order to make the projections work. If, however, M, N have the same type, then they can be paired using only β -conversion. This will enable us to represent also heterogenous pairs.

1.3.17. LEMMA. *For every type $A \in \mathbb{T}_o$ there is a type $A \times A \in \mathbb{T}_o$ such that there are terms $\text{pair}_o^A, \text{left}_o^A$ and right_o^A such that $\text{rk}(A \times A) = \text{rk}(A) + 2$ and*

$$\begin{aligned} \text{pair}_o^A &\in \Lambda_o^\emptyset(A \rightarrow A \rightarrow A \times A); \\ \text{left}_o^A &\in \Lambda_o^\emptyset(A \times A \rightarrow A); \\ \text{right}_o^A &\in \Lambda_o^\emptyset(A \times A \rightarrow A), \end{aligned}$$

and for $M, N \in \Lambda_o^\Gamma(A)$ one has

$$\begin{aligned} \text{left}_o^A(\text{pair}_o^A MN) &=_{\beta} M; \\ \text{right}_o^A(\text{pair}_o^A MN) &=_{\beta} N. \end{aligned}$$

PROOF. Take

$$\begin{aligned} A \times A &= (A \rightarrow A \rightarrow A) \rightarrow A; \\ \text{pair}_o^A &= \lambda mnz. zmn; \\ \text{left}_o^A &= \lambda p. pK; \\ \text{right}_o^A &= \lambda p. pK_*. \blacksquare \end{aligned}$$

Using $\beta\eta$ -conversion one can define a cartesian product for all pairs of types.

1.3.18. PROPOSITION (Gandy [1950?], Grzegorzczuk [1964]). *Let $A, B \in \mathbb{T}_o$ be arbitrary types. Then there is a type $A \times B \in \mathbb{T}_o$ with*

$$\text{rk}(A \times B) = \max\{\text{rk}(A), \text{rk}(B), 2\}$$

such that there are terms $\text{pair}_o^{A,B}, \text{left}_o^{A,B}$ and $\text{right}_o^{A,B}$ such that

$$\begin{aligned} \text{pair}_o^{A,B} &\in \Lambda_o^\emptyset(A \rightarrow A \rightarrow A \times B); \\ \text{left}_o^{A,B} &\in \Lambda_o^{\{z:o\}}(A \times B \rightarrow A); \\ \text{right}_o^{A,B} &\in \Lambda_o^{\{z:o\}}(A \times B \rightarrow B), \end{aligned}$$

and for $M \in \Lambda_o^\Delta(A), N \in \Lambda_o^\Delta(B)$ one has

$$\begin{aligned} \text{left}_o^{A,B}(\text{pair}_o^{A,B} MN) &=_{\beta\eta} M; \\ \text{right}_o^{A,B}(\text{pair}_o^{A,B} MN) &=_{\beta\eta} N. \end{aligned}$$

PROOF. Write $n = \text{arity}(A), m = \text{arity}(B)$. Define

$$A \times B = A(1) \rightarrow \dots \rightarrow A(n) \rightarrow B(1) \rightarrow \dots \rightarrow B(m) \rightarrow o \underline{\times} o,$$

where $o \underline{\times} o = (o \rightarrow o \rightarrow o) \rightarrow o$. Then

$$\begin{aligned} \text{rk}(A \times B) &= \max_{i,j} \{ \text{rk}(A_i) + 1, \text{rk}(B_j) + 1, \text{rk}(o^2 \rightarrow o) + 1 \} \\ &= \max \{ \text{rk}(A), \text{rk}(B), 2 \}. \end{aligned}$$

Define z_A inductively: $z_o = z$; $z_{A \rightarrow B} = \lambda a. z_B$. Then $z_A \in \Lambda_o^{z:o}(A)$. Write $\vec{x} = x_1, \dots, x_n, \vec{y} = y_1, \dots, y_m, \vec{z}_A = z_{A(1)}, \dots, z_{A(n)}$ and $\vec{z}_B = z_{B(1)}, \dots, z_{B(m)}$. Now define

$$\begin{aligned} \text{pair}_o^{A,B} &= \lambda mn. \lambda \vec{x} \vec{y}. \text{pair}_o^o(m\vec{x})(n\vec{y}); \\ \text{left}_o^{A,B} &= \lambda p. \lambda \vec{x}. \text{left}_o^o(p\vec{x}\vec{z}_B); \\ \text{right}_o^{A,B} &= \lambda p. \lambda \vec{x}. \text{right}_o^o(p\vec{z}_A\vec{y}). \end{aligned}$$

Then e.g. and

$$\begin{aligned} \text{left}_o^{A,B}(\text{pair}_o^{A,B} MN) &=_{\beta} \lambda \vec{x}. \text{left}_o^o(\text{pair}_o^o MN \vec{x} \vec{z}_B) \\ &=_{\beta} \lambda \vec{x}. \text{left}_o^o[\text{pair}_o^o(M\vec{x})(N\vec{z}_B)] \\ &=_{\beta} \lambda \vec{x}. (M\vec{x}) \\ &=_{\eta} M. \blacksquare \end{aligned}$$

In Barendregt [1974] it is proved that η -conversion is essential: with β -conversion one can pair only certain combinations of types. Also it is shown that there is no *surjective* pairing in the theory with $\beta\eta$ -conversion. Surjectivity states that

$$\text{pair}(\text{left}z)(\text{right}z) =_{\beta\eta} z.$$

In Section 5.2 we will discuss systems extended with surjective pairing. With similar techniques as in mentioned paper it can be shown that in $\lambda_{\rightarrow}^{\infty}$ there is no pairing function $\text{pair}_o^{\alpha,\beta}$ for base types (even with η -conversion). In section 2.3 we will encounter other differences between $\lambda_{\rightarrow}^{\infty}$ and λ_{\rightarrow}^o .

1.3.19. PROPOSITION. *Let $A_1, \dots, A_n \in \mathbb{T}_o$. There are closed terms*

$$\begin{aligned} \text{tuple}^n &: A_1 \rightarrow \dots \rightarrow A_n \rightarrow (A_1 \times \dots \times A_n) \\ \text{proj}_k^n &: A_1 \times \dots \times A_n \rightarrow A_k \end{aligned}$$

such that for M_1, \dots, M_n of the right type one has

$$\text{proj}_k^n(\text{tuple}^n M_1 \dots M_n) =_{\beta\eta} M_k.$$

If there is little danger of confusion and the \vec{M}, N are of the right type we write

$$\begin{aligned} \langle M_1, \dots, M_n \rangle &\equiv \text{tuple}^n M_1 \dots M_n; \\ N \cdot k &\equiv \text{proj}_k^n N. \end{aligned}$$

PROOF. By iterating pairing. \blacksquare

1.4. The λ_{\rightarrow} systems à la Curry, à la Church and à la de Bruijn

The Curry version of λ_{\rightarrow} is called *implicitly typed* because an expression like

$$\lambda x.xK$$

has a type, but it requires work to find it. In §2.2 we will see that this work is feasible. In systems more complex than λ_{\rightarrow} , finding types in the implicit version is more complicated and may even not be computable. This will be the case with second and higher order types, like $\lambda 2$ (system F), treated in Volume II.

Therefore there are also versions of the systems that are typed explicitly *à la Church*. The explicitly typed version of λ_{\rightarrow} will be denoted by $\lambda_{\rightarrow}^{\mathbb{A}}_{\text{Ch}}$ or more often by $\lambda_{\rightarrow}^{\text{Ch}}$. In this system one writes for example for the term above

$$\lambda x^{(A \rightarrow B \rightarrow A) \rightarrow C}.x^{(A \rightarrow B \rightarrow A) \rightarrow C}(\lambda y^A z^B.y^A).$$

In the system *à la de Bruijn* this is written as

$$\lambda x:((A \rightarrow B \rightarrow A) \rightarrow C).x(\lambda y:A \lambda z:B.y).$$

So in both cases terms are not just elements of Λ , but versions ornamented with elements of \mathbb{T} .

1.4.1. DEFINITION. Let \mathbb{A} be a set of type atoms. The Church version of λ_{\rightarrow} , notation $\lambda_{\rightarrow}^{\mathbb{A}}_{\text{Ch}}$ or $\lambda_{\rightarrow}^{\text{Ch}}$ if \mathbb{A} is not emphasized, is defined as follows. The system has the same set of types $\mathbb{T}_{\mathbb{A}}$ as $\lambda_{\rightarrow}^{\mathbb{A}}_{\text{Cu}}$.

(i) The set of term variables is different: each such variable is coupled with a unique type. Let

$$\mathbf{V}^{\mathbb{T}} = \mathbf{V} \times \mathbb{T}.$$

(ii) Notation. $\mathbf{x}, \mathbf{y}, \mathbf{z}, \dots$ range over $\mathbf{V}^{\mathbb{T}}$. If $\mathbf{x} = \langle x, A \rangle$, then we also write

$$\mathbf{x} = x^A = x:A.$$

(iii) Terms of type A , notation $\Lambda_{\rightarrow}^{\text{Ch}}(A)$, are defined as follows.

$x^A \in \Lambda_{\rightarrow}(A);$	
$M \in \Lambda_{\rightarrow}(A \rightarrow B), N \in \Lambda_{\rightarrow}(A)$	$\Rightarrow (MN) \in \Lambda_{\rightarrow}(B);$
$M \in \Lambda_{\rightarrow}(B)$	$\Rightarrow (\lambda x^A.M) \in \Lambda_{\rightarrow}(A \rightarrow B).$

Figure 1.6: The system $\lambda_{\rightarrow}^{\text{Ch}}$ of typed terms *à la Church*

(iv) The set of terms of $\lambda_{\rightarrow}^{\text{Ch}}$, notation $\Lambda_{\rightarrow}^{\text{Ch}}$, is defined as

$$\Lambda_{\rightarrow}^{\text{Ch}} = \bigcup_{A \in \mathbb{T}} \Lambda_{\rightarrow}^{\text{Ch}}(A).$$

For example

$$\begin{aligned} y^{B \rightarrow A} x^B &\in \Lambda_{\rightarrow}^{\text{Ch}}(A) \\ \lambda x^A. y^{B \rightarrow A} &\in \Lambda_{\rightarrow}^{\text{Ch}}(A \rightarrow B \rightarrow A) \\ \lambda x^A. x^A &\in \Lambda_{\rightarrow}^{\text{Ch}}(A \rightarrow A) \end{aligned}$$

Substitution of a term N for a typed variable x^B is defined as usual. If also N has type B , then the resulting term keeps its type.

1.4.2. PROPOSITION. (i) Let $M \in \Lambda_{\rightarrow}(A)$, $N \in \Lambda_{\rightarrow}(B)$. Then

$$(M[x^B := N]) \in \Lambda_{\rightarrow}(A).$$

(ii) Let $A, B \in \mathbb{T}$. Then

$$M \in \Lambda_{\rightarrow}^{\text{Ch}}(A) \Rightarrow M[\alpha := B] \in \Lambda_{\rightarrow}^{\text{Ch}}(A[\alpha := B]).$$

PROOF. (i), (ii) By induction on the structure of M . ■

1.4.3. DEFINITION. On $\Lambda_{\mathbb{A}}^{\text{Ch}}$ we define the following notions of reduction.

$\begin{aligned} (\lambda x^A. M)N &\rightarrow M[x := N] & (\beta) \\ \lambda x^A. Mx &\rightarrow M, & \text{if } x \notin \text{FV}(M) \quad (\eta) \end{aligned}$

Figure 1.7: $\beta\eta$ -contraction rules for $\lambda_{\rightarrow}^{\mathbb{A}}_{\text{Ch}}$

As usual, see B[1984], these notions of reduction generate the corresponding reduction relations. Also there are the corresponding conversion relations $=_{\beta}$, $=_{\eta}$ and $=_{\beta\eta}$. Terms in $\lambda_{\rightarrow}^{\text{Ch}}$ will often be considered modulo $=_{\beta}$ or $=_{\beta\eta}$. The notation $M = N$, means $M =_{\beta\eta} N$ by default.

For every type A the set $\Lambda_{\rightarrow}^{\text{Ch}}(A)$ is closed under reduction.

1.4.4. PROPOSITION. (i) $((\lambda x^B. M)N) \in \Lambda_{\rightarrow}^{\text{Ch}}(A) \Rightarrow M^A[x := N^B] \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$.

(ii) $\lambda x^B. Mx^B \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$ and $x^B \notin \text{FV}(M)$, then $M \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$.

(iii) $M \in \Lambda_{\rightarrow}^{\text{Ch}}$ and $M \rightarrow_{\beta\eta} N$. Then $N \in \Lambda_{\rightarrow}^{\text{Ch}}$.

PROOF. (i) If $(\lambda x^B. M)N \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$, then $\lambda x^B. M \in \Lambda_{\rightarrow}^{\text{Ch}}(B' \rightarrow A)$, so $B = B'$, and $N \in \Lambda_{\rightarrow}^{\text{Ch}}(B)$. Now Lemma 1.4.2 applies.

(ii) Similarly. If $(\lambda x^B. Mx^B) \in \Lambda_{\rightarrow}^{\text{Ch}}(C)$, then $C = B \rightarrow A$ and $Mx^B \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$. But then $M \in \Lambda_{\rightarrow}^{\text{Ch}}(B \rightarrow A)$.

(iii) By induction on the relation $\rightarrow_{\beta\eta}$, using (i), (ii). ■

The reasoning needed in the proof of this Lemma is made more explicit in the generation lemmas in Section 2.1.

Relating the Curry and Church systems

There are canonical translations between $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{Cu}}$.

1.4.5. DEFINITION. There is a ‘forgetful’ map $|\cdot| : \Lambda_{\mathbb{A}} \rightarrow \Lambda$ defined as follows:

$$\begin{aligned} |x^A| &\equiv x; \\ |MN| &\equiv |M||N|; \\ |\lambda x:A.M| &\equiv \lambda x.|M|. \end{aligned}$$

The map $|\cdot|$ just erases all type ornamentations of a term in $\Lambda_{\mathbb{A}}^{\text{Ch}}$. The following result states that ornamented legal terms in the Church version ‘project’ to legal terms in the Curry version of λ_{\rightarrow} . Conversely, legal terms in $\lambda_{\rightarrow}^{\text{Cu}}$ can be ‘lifted’ to legal terms in $\lambda_{\rightarrow}^{\text{Ch}}$. There is however a condition needed. The term

$$\lambda x^{A \rightarrow A \rightarrow B} \lambda x^A . x^{A \rightarrow B} x^A x^A \in \Lambda((A \rightarrow A \rightarrow B) \rightarrow (A \rightarrow B))$$

projects to $\lambda x \lambda x . x x x$ which does not have a type.

1.4.6. DEFINITION. (i) A collection of type variables $\mathcal{X} \subseteq \mathbf{V}^{\mathbb{T}}$ is called well-named iff $x^A, x^B \in \mathcal{X} \Rightarrow A = B$. (If we consider x as the first name of x^A and A as its family name, then the notion that \mathcal{X} is well named means that first names uniquely describe the variable.)

(ii) A term is *well-named* if $\text{FV}(M)$ is well-named.

1.4.7. PROPOSITION. (i) Let $M \in \Lambda_{\mathbb{A}}$ be well-named. Then

$$M \in \Lambda_{\mathbb{A}}^{\text{Ch}}(A) \Rightarrow \Gamma_M \vdash_{\lambda_{\rightarrow}}^{\text{Cu}} |M| : A,$$

where $\Gamma_M = \{x:A \mid x^A \in \text{FV}(M)\}$

(ii) Let $M \in \Lambda$. Then

$$\Gamma \vdash_{\lambda_{\rightarrow}}^{\text{Cu}} M : A \iff \exists M' \in \Lambda_{\mathbb{A}}^{\text{Ch}}(A). |M'| \equiv M \text{ and } M' \text{ is well-named.}$$

PROOF. (i) By induction on the generation of $\Lambda_{\mathbb{A}}^{\text{Ch}}$. The assumption that M is well-named insures that Γ_M is well-defined and that $\Gamma_P \cup \Gamma_Q = \Gamma_{PQ}$.

(ii) (\Rightarrow) By induction on the proof of $\Gamma \vdash M : A$ with the induction loading that $\Gamma_{M'} = \Gamma$. (\Leftarrow) By (i). ■

Notice that the converse of Proposition 1.4.7(i) is not true: one has

$$\vdash_{\lambda_{\rightarrow}}^{\text{Cu}} |\lambda x^A \lambda y^A . x^A y^A| : (A \rightarrow A) \rightarrow (A \rightarrow A),$$

but $(\lambda x^A \lambda y^A . x^A y^A) \notin \Lambda^{\text{Ch}}((A \rightarrow A) \rightarrow (A \rightarrow A))$.

1.4.8. COROLLARY. In particular, for a type $A \in \mathbb{T}$ one has

$$A \text{ is inhabited in } \lambda_{\rightarrow}^{\text{Cu}} \iff A \text{ is inhabited in } \lambda_{\rightarrow}^{\text{Ch}}.$$

PROOF. Immediate. ■

The translation preserves reduction and conversion.

1.4.9. PROPOSITION. *Let $R = \beta, \eta$ or $\beta\eta$. Then*

(i) *Let $M, N \in \Lambda_{\rightarrow}^{\text{Ch}}$. Then*

$$M \rightarrow_R N \Rightarrow |M| \rightarrow_R |N|.$$

(ii) *Let $M_1, M_2 \in \Lambda_{\rightarrow, \text{Cu}}^{\Gamma}(A)$, $M_1 = |M_1'|$, with $M_1' \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$. Then*

$$\begin{aligned} M_1 \rightarrow_R M_2 &\Rightarrow \exists M_2' \in \Lambda_{\rightarrow}^{\text{Ch}}(A). \\ |M_i'| &\equiv M_i \text{ \& } M_1' \rightarrow_R M_2'. \end{aligned}$$

(iii) *The same results hold for \twoheadrightarrow_R and R -conversion.*

PROOF. Easy. ■

The systems λ_{\rightarrow} à la de Bruijn

There is the following disadvantage about the Church systems. Consider

$$I \equiv \lambda x^A. x^A.$$

In the next volume we will consider dependent types coming from the Authomath language family, see Nederpelt et al. [1994], designed for formalizing arguments and proof-checking. These are types that depend on a term variable (ranging over another type). An intuitive example is A^n , where n is a variable ranging over natural numbers. A more formal example is Px , where $x : A$ and $P : A \rightarrow \mathbb{T}$. In this way types may contain redexes and we may have the following reduction

$$I \equiv \lambda x^A. x^A \rightarrow_{\beta} \lambda x^A. x^{A'},$$

by reducing only the second A to A' . The question now is whether λx^A binds the $x^{A'}$. If we write I as

$$I \equiv \lambda x:A. x,$$

then this problem disappears

$$\lambda x:A. x \twoheadrightarrow \lambda x:A'. x.$$

In the following system $\lambda_{\rightarrow, \text{dB}}^{\mathbb{A}}$ this idea is formalized.

1.4.10. DEFINITION. The system $\lambda_{\rightarrow, \text{dB}}^{\mathbb{A}}$ starts with a collection of *pseudo-terms*, notation $\Lambda_{\rightarrow}^{\text{dB}}$, is defined by the following grammar.

$$\Lambda_{\mathbb{A}} = \mathbb{A} \mid \Lambda_{\mathbb{A}} \Lambda_{\mathbb{A}} \mid \lambda \mathbb{A}:\mathbb{T}. \Lambda_{\mathbb{A}}.$$

1.4. THE λ_{\rightarrow} SYSTEMS À LA CURRY, À LA CHURCH AND À LA DE BRUIJN 41

For example $\lambda x:\alpha.x$ and $(\lambda x:\alpha.x)(\lambda y:\beta.y)$ are pseudo-terms. As we will see, the first one is a *legal*, i.e. actually typeable term in $\lambda_{\rightarrow}^{\text{dB}}$, whereas the second one is not.

1.4.11. DEFINITION. (i) Let Γ be a basis consisting of a set of declarations $x:A$ with distinct term variables x and types $A \in \mathbb{T}_{\mathbb{A}}$. This is exactly the same as for $\lambda_{\rightarrow}^{\text{Cu}}$.

(ii) The system of type assignment obtaining statements $\Gamma \vdash M : A$ with Γ a basis, M a pseudoterm and A a type, is defined as follows.

(axiom)	$\Gamma \vdash x : A,$	if $(x:A) \in \Gamma;$
(\rightarrow -elimination)	$\frac{\Gamma \vdash M : (A \rightarrow B) \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : B};$	
(\rightarrow -introduction)	$\frac{\Gamma, x:A \vdash M : B}{\Gamma \vdash (\lambda x:A.M) : (A \rightarrow B)}.$	

Figure 1.8: $\lambda_{\rightarrow}^{\text{dB}}$

Provability in $\lambda_{\rightarrow}^{\text{dB}}$ is denoted by $\vdash_{\lambda_{\rightarrow}}^{\text{dB}}$. Thus the legal terms of $\lambda_{\rightarrow}^{\text{dB}}$ are defined by making a selection from the context-free language $\Lambda_{\rightarrow}^{\text{dB}}$. That $\lambda x:\alpha.x$ is legal follows from $x:\alpha \vdash_{\lambda_{\rightarrow}}^{\text{dB}} x : \alpha$ using the \rightarrow -introduction rule. That $(\lambda x:\alpha.x)(\lambda y:\beta.y)$ is not follows systematically from section 2.3. These legal terms do not form a context-free language, do exercise 1.5.8.

There is a close connection between $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{dB}}$. First we need the following.

1.4.12. LEMMA. *Let $\Gamma \subseteq \Gamma'$ be bases of $\lambda_{\rightarrow}^{\text{dB}}$. Then*

$$\Gamma \vdash_{\lambda_{\rightarrow}}^{\text{dB}} M : A \Rightarrow \Gamma' \vdash_{\lambda_{\rightarrow}}^{\text{dB}} M : A.$$

PROOF. By induction on the derivation of the first statement. ■

1.4.13. DEFINITION. (i) Let $M \in \Lambda_{\rightarrow}^{\text{Ch}}$ and suppose $\text{FV}(M) \subseteq \text{dom}(\Gamma)$. Define M^{Γ} inductively as follows.

$$\begin{aligned} x^{\Gamma} &= x^{\Gamma(x)}; \\ (MN)^{\Gamma} &= M^{\Gamma} N^{\Gamma}; \\ (\lambda x:A.M)^{\Gamma} &= \lambda x^A.M^{\Gamma, x:A}. \end{aligned}$$

(ii) Let $M \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$ in $\lambda_{\rightarrow}^{\text{Ch}}$. Define M^{-} , a pseudo-term of $\lambda_{\rightarrow}^{\text{dB}}$, as follows.

$$\begin{aligned} (x^A)^{-} &= x; \\ (MN)^{-} &= M^{-} N^{-}; \\ (\lambda x^A.M)^{-} &= \lambda x:A.M^{-}. \end{aligned}$$

(iii) For M a term of $\lambda_{\rightarrow}^{\text{dB}}$ define the basis Γ_M as follows.

$$\begin{aligned}\Gamma_{x^A} &= \{x:A\}; \\ \Gamma_{MN} &= \Gamma_M \cup \Gamma_N; \\ \Gamma_{\lambda x^A.M} &= \Gamma_M \setminus \{x:A\}\end{aligned}$$

1.4.14. EXAMPLE. To get the (easy) intuition, consider the following.

$$\begin{aligned}(\lambda x:A.x)^{\emptyset} &\equiv (\lambda x^A.x^A); \\ (\lambda x^A.x^A)^{-} &\equiv (\lambda x:A.x); \\ (\lambda x:A \rightarrow B.xy)^{\{y:A\}} &\equiv \lambda x^{A \rightarrow B}.x^{A \rightarrow B}y^A; \\ \Gamma_{(\lambda x^{A \rightarrow B}.x^{A \rightarrow B}y^A)} &= \{y:A\}.\end{aligned}$$

1.4.15. PROPOSITION. (i) Let $M \in \Lambda_{\mathbb{A}}^{\text{Ch}}$ and let M be well-named. Let Γ a basis of $\lambda_{\rightarrow}^{\text{dB}}$. Then

$$M \in \Lambda_{\mathbb{A}}^{\text{Ch}}(A) \iff \Gamma_M \vdash_{\lambda_{\rightarrow}}^{\text{dB}} M^{-} : A.$$

$$(ii) \quad \Gamma \vdash_{\lambda_{\rightarrow}}^{\text{dB}} M : A \iff M^{\Gamma} \in \Lambda_{\rightarrow}^{\text{Ch}}(A).$$

PROOF. (i), (ii)(\Rightarrow) By induction on the proof or the definition of the LHS.

(i)(\Leftarrow) By (ii)(\Rightarrow), using $(M^{-})^{\Gamma_M} \equiv M$.

(ii)(\Leftarrow) By (i)(\Rightarrow), using $(M^{\Gamma})^{-} \equiv M$, $\Gamma_{M^{\Gamma}} \subseteq \Gamma$ and proposition 1.4.12. ■

1.4.16. COROLLARY. In particular, for a type $A \in \mathbb{T}$ one has

$$A \text{ is inhabited in } \lambda_{\rightarrow}^{\text{Ch}} \iff A \text{ is inhabited in } \lambda_{\rightarrow}^{\text{dB}}.$$

PROOF. Immediate. ■

Again the translation preserves reduction and conversion

1.4.17. PROPOSITION. (i) Let $M, N \in \Lambda_{\rightarrow}^{\Gamma, \text{Ch}}$. Then

$$M \rightarrow_R N \iff M^{\Gamma} \rightarrow_R N^{\Gamma},$$

where $R = \beta, \eta$ or $\beta\eta$.

(ii) Let $M_1, M_2 \in \Lambda(A)$ and R as in (i). Then

$$M_1 \rightarrow_R M_2 \iff M_1^{-} \rightarrow_R M_2^{-}.$$

(iii) The same results hold for conversion.

PROOF. Easy. ■

Comparing $\lambda_{\rightarrow}^{\text{dB}}$ and $\lambda_{\rightarrow}^{\text{Cu}}$

1.4.18. PROPOSITION. (i) Let $M \in \Lambda_{\mathbb{A}}$ be well-named. Then

$$M \in \Lambda_{\rightarrow}^{\text{dB}}(A) \Rightarrow \Gamma_M \vdash_{\lambda_{\rightarrow}}^{\text{Cu}} |M| : A,$$

where $\Gamma_M = \{x:A \mid x^A \in \text{FV}(M)\}$

(ii) Let $M \in \Lambda$. Then

$$\Gamma \vdash_{\lambda_{\rightarrow}}^{\text{Cu}} M : A \iff \exists M' \in \Lambda_{\mathbb{A}}^{\text{dB}}(A). |M'| \equiv M \text{ and } M' \text{ is well-named.}$$

PROOF. As for Proposition 1.4.7. ■

Again the implication in (i) cannot be reversed.

The three systems compared

Now we can harvest a comparison between the three systems $\lambda_{\rightarrow}^{\text{Ch}}$, $\lambda_{\rightarrow}^{\text{dB}}$ and $\lambda_{\rightarrow}^{\text{Cu}}$.

1.4.19. THEOREM. Let M be a well-named term in $\Lambda_{\mathbb{T}_{\mathbb{A}}}$. Then we have

$$\begin{aligned} M \in \Lambda_{\mathbb{A}}^{\text{Ch}}(A) &\iff \Gamma_M \vdash_{\lambda_{\rightarrow}}^{\text{dB}} M^- : A \\ &\Rightarrow \Gamma_M \vdash_{\lambda_{\rightarrow}}^{\text{Cu}} |M| : A \\ &\Rightarrow M' \in \Lambda_{\rightarrow}^{\text{Ch}}(A). |M'| \equiv M \text{ \& } M' \text{ is well-named.} \end{aligned}$$

PROOF. By Propositions 1.4.7 and 1.4.18 and the fact that $|M^-| = |M|$. ■

Again the second and third arrows cannot be reversed.

It may seem a bit exaggerated to have three versions of the simply typed lambda calculus: $\lambda_{\rightarrow}^{\text{Cu}}$, $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{dB}}$. But this is indeed necessary.

The Curry version corresponds to implicitly typed programming languages like ML. Since implicit typing makes programming more easy, we want to consider this system.

For extensions of $\lambda_{\rightarrow}^{\text{Cu}}$, like $\lambda 2$ with second order (polymorphic) types, type checking is not decidable, see Wells [1999], and hence one needs the explicit versions. The two explicitly typed systems $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{dB}}$ are basically isomorphic as shown above. These systems have a very canonical semantics if the version $\lambda_{\rightarrow}^{\text{Ch}}$ is used. Nevertheless we want two versions because the version $\lambda_{\rightarrow}^{\text{dB}}$ can be extended more naturally to more powerful type systems in which there is a notion of reduction on the types (those with ‘dependent types’ and those with higher order types to be treated in Volume II) generated simultaneously. Also there are important extensions in which there is a reduction relation on types, e.g. in the system $\lambda\omega$ with higher order types. The classical version of λ_{\rightarrow} gives problems. For example, if $A \twoheadrightarrow B$, does one have that $\lambda x^A.x^A \twoheadrightarrow \lambda x^A.x^B$? Moreover, is the x^B bound by the λx^A ? By denoting $\lambda x^A.x^A$ as $\lambda x:A.x$, as is done in $\lambda_{\rightarrow}^{\text{Ch}}$, these problems do not arise. This is so important that for explicitly typed extensions of λ_{\rightarrow} ,

we will need to use the Ch-versions, even if for these systems the model theory will be a bit more complicated to express.

The situation is not so bad as it may seem, since the three systems and their differences are easy to memorize. Just look at the following examples.

$$\begin{aligned} \lambda x.xy &\in \Lambda_{\rightarrow \text{Cu}}^{\{y:o\}}((o \rightarrow o) \rightarrow o) && \text{(Curry);} \\ \lambda x:(o \rightarrow o).xy &\in \Lambda_{\rightarrow \text{dB}}^{\{y:o\}}((o \rightarrow o) \rightarrow o) && \text{(de Bruijn);} \\ \lambda x^{o \rightarrow o}.x^{o \rightarrow o}y^o &\in \Lambda_{\rightarrow \text{Ch}}((o \rightarrow o) \rightarrow o) && \text{(Church).} \end{aligned}$$

We have chosen to present the three versions of λ_{\rightarrow} because one finds them like this in the literature.

In this Part I of the book we are interested in untyped lambda terms that can be typed using simple types. We will see that up to substitution this typing is unique. For example

$$\lambda f x.f(fx)$$

can have as type $(o \rightarrow o) \rightarrow o \rightarrow o$, but also $(A \rightarrow A) \rightarrow A \rightarrow A$ for any type A . Moreover there is a simple algorithm to find all possible types for an untyped lambda term, see Section 2.3.

We are interested in typable terms M . This can be terms in the set of untyped lambda terms Λ with Curry typing. Since we are at the same time also interested in the types of the subterms of M , the Church typing is a convenient notation. Moreover, this information is almost uniquely determined once the type A of M is known or required. By this we mean that the Church typing is uniquely determined by A for the nf M^{nf} of M , if M contains some redexes of the form $(\lambda x.M)N$ with $x \notin \text{FV}(M)$ (K-redexes), otherwise for M itself. For example the Church typing of $M \equiv \text{Kl}y$ of type $\alpha \rightarrow \alpha$ is $(\lambda x^{\alpha \rightarrow \alpha}y^{\beta}.x^{\alpha \rightarrow \alpha})(\lambda z^{\alpha}.z^{\alpha})y^{\beta}$. The type β is not determined. But for the $\beta\eta$ -nf of M , the term l , the Church typing is uniquely determined and is $\text{l}_{\alpha} \equiv \lambda z^{\alpha}.z^{\alpha}$. See Exercise 2.5.3.

If A is not available, then the type information for M is schematic in the groundtypes. By this we mean that e.g. the term $\text{l} \equiv \lambda x.x$ has a Church version $\lambda x^{\alpha}.x^{\alpha}$ and type $\alpha \rightarrow \alpha$, where one can substitute both in the term and type any $A \in \mathbb{T}_{\mathbb{A}}$ for α . We will study this in greater detail in Section 2.3.

1.5. Exercises

1.5.1. Show that the well-known combinators

$$\begin{aligned} \text{I} &\equiv \lambda x.x, \\ \text{K} &\equiv \lambda xy.x, \\ \text{S} &\equiv \lambda xyz.xz(yz) \end{aligned}$$

respectively have the following types.

$$\begin{aligned} \text{I} &: A \rightarrow A; \\ \text{K} &: A \rightarrow B \rightarrow A; \\ \text{S} &: (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C). \end{aligned}$$

1.5.2. Find types for

$$\begin{aligned} B &\equiv \lambda xyz.x(yz); \\ C &\equiv \lambda xyz.xzy; \\ C_* &\equiv \lambda xy.yx; \\ K_* &\equiv \lambda xy.y; \\ W &\equiv \lambda xy.xyy. \end{aligned}$$

1.5.3. Find types for \mathbf{SKK} , $\lambda xy.y(\lambda z.zxx)x$ and $\lambda fx.f(f(fx))$.

1.5.4. Show that $\text{rk}(A \rightarrow B \rightarrow C) = \max\{\text{rk}(A) + 1, \text{rk}(B) + 1, \text{rk}(C)\}$.

1.5.5. Show that if $M \equiv P[x := Q]$ and $N \equiv (\lambda x.P)Q$, then M may have a type in $\lambda_{\rightarrow}^{\text{Cu}}$ but N not. A similar observation can be made for pseudo-terms of $\lambda_{\rightarrow}^{\text{dB}}$.

1.5.6. Show the following.

- (i) $\lambda xy.(xy)x \notin \Lambda_{\rightarrow \text{Cu}}^{\emptyset}$.
- (ii) $\lambda xy.x(yx) \in \Lambda_{\rightarrow \text{Cu}}^{\emptyset}$.

1.5.7. Find inhabitants in $(A \rightarrow B \rightarrow C) \rightarrow B \rightarrow A \rightarrow C$ and $(A \rightarrow A \rightarrow B) \rightarrow A \rightarrow B$.

1.5.8. [van Benthem] Show that $\Lambda_{\rightarrow \text{Ch}}(A)$ and $\Lambda_{\rightarrow \text{Cu}}^{\emptyset}(A)$ is for some $A \in \mathbb{T}_{\mathbb{A}}$ not a context free language.

1.5.9. Define in λ_{\rightarrow}^o the ‘pseudo-negation’ $\sim A \equiv A \rightarrow o$. Construct an inhabitant of $\sim \sim \sim A \rightarrow \sim A$.

1.5.10. Let X be a finite set. Give a representation of X in λ_{\rightarrow}^o such that every function $f: X^k \rightarrow X$ can be lambda defined in λ_{\rightarrow}^o .

1.5.11. Prove the following, see definition 1.4.13.

- (i) Let $M \in \Lambda_{\rightarrow \text{Ch}}$ be such that $\text{FV}(M) \subseteq \text{dom}(\Gamma)$, then $(M^{\Gamma})^{-} \equiv M$ and $\Gamma_{M^{\Gamma}} \subseteq \Gamma$.
- (ii) Let $M \in \Lambda_{\rightarrow \text{dB}}$, then $(M^{-})^{\Gamma_M} \equiv M$.

1.5.12. Construct a term F with $\vdash_{\lambda_{\rightarrow}^o} F : T_2 \rightarrow T_2$ such that for trees t one has $F \underline{t} =_{\beta} \underline{t^{\text{mir}}}$, where t^{mir} is the mirror image of t , defined by

$$\begin{aligned} \epsilon^{\text{mir}} &= \epsilon; \\ (p(t, s))^{\text{mir}} &= p(s^{\text{mir}}, t^{\text{mir}}). \end{aligned}$$

1.5.13. A term M is called *proper* if all λ 's appear in the prefix of M , i.e. $M \equiv \lambda \vec{x}.N$ and there is no λ occurring in N . Let A be a type such that $\Lambda^{\emptyset}(A)$ is not empty. Show that

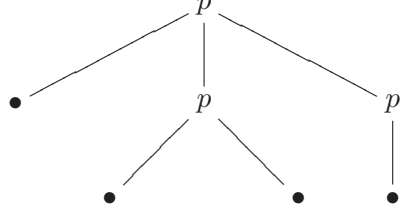
$$\text{Every nf of type } A \text{ is proper} \iff \text{rk}(A) \leq 2.$$

1.5.14. Determine the class of closed inhabitants of the types 4 and 5.

1.5.15. The collection of multi-ary trees can be seen as part of a multi-sorted algebra with sorts MTree and L_{MTree} as follows.

$$\begin{aligned} \text{nil} &\in L_{\text{Mtree}}; \\ \text{cons} &\in \text{Mtree} \rightarrow L_{\text{Mtree}} \rightarrow L_{\text{Mtree}}; \\ p &\in L_{\text{Mtree}} \rightarrow \text{Mtree}. \end{aligned}$$

Represent this multi-sorted free algebra in λ_{\rightarrow}^o . Construct the lambda term representing the tree



1.5.16. In this exercise it will be proved that each term (having a β -nf) has a unique lnf. A term M (typed or untyped) is always of the form $\lambda x_1 \dots x_n. y M_1 \dots M_m$ or $\lambda x_1 \dots x_r. (\lambda x. M_0) M_1 \dots M_s$. Then $y M_1 \dots M_m$ (or $(\lambda x. M_0) M_1 \dots M_m$) is the *matrix* of M and the $(M_0,) M_1, \dots, M_m$ are its *components*. A typed term $M \in \Lambda^\Gamma(A)$ is said to be *fully eta* (f.e.) *expanded* if its matrix is of type o and its components are f.e. expanded. Show the following for typed terms. (For untyped terms there is no finite f.e. expanded form, but the Nakajima tree, see B[1984] Exercise 19.4.4, is the corresponding notion for the untyped terms.)

- (i) M is in lnf iff M is a β -nf and f.e. expanded.
- (ii) If $M =_{\beta\eta} N_1 =_{\beta\eta} N_2$ and N_1, N_2 are β -nfs, then $N_1 =_\eta N_2$. [Hint. Use η -postponement, see B[1984] Proposition 15.1.5.]
- (iii) $N_1 =_\eta N_2$ and N_1, N_2 are β -nfs, then there exist $N\downarrow$ and $N\uparrow$ such that $N_i \twoheadrightarrow_\eta N\downarrow$ and $N\uparrow \twoheadrightarrow_\eta N_i$, for $i = 1, 2$. [Hint. Show that both \rightarrow_η and $\eta\leftarrow$ satisfy the diamond lemma.]
- (iv) If M has a β -nf, then it has a unique lnf.
- (v) If N is f.e. expanded and $N \twoheadrightarrow_\beta N'$, then N' is f.e. expanded.
- (vi) For all M there is a f.e. expanded M^* such that $M^* \twoheadrightarrow_\eta M$.
- (vii) If M has a β -nf, then the lnf of M is the β -nf of M^* , its full eta expansion.

1.5.17. Like in B[1984], the terms in this book are *abstract terms*, considered modulo α -conversion. Sometimes it is useful to be explicit about α -conversion and even to violate the variable convention that in a subterm of a term the names of free and bound variables should be distinct. For this it is useful to modify the system of type assignment.

- (i) Show that $\vdash_{\lambda_{\rightarrow}}$ is not closed under α -conversion. I.e.

$$\Gamma \vdash M:A, M \equiv_\alpha M' \not\Rightarrow \Gamma \vdash M':A.$$

[Hint. Consider $M' \equiv \lambda x.x(\lambda x.x)$.]

- (ii) Consider the following system of type assignment to untyped terms.

$$\boxed{
 \begin{array}{l}
 \{x:A\} \vdash x : A; \\
 \frac{\Gamma_1 \vdash M : (A \rightarrow B) \quad \Gamma_2 \vdash N : A}{\Gamma_1 \cup \Gamma_2 \vdash (MN) : B}, \quad \text{provided } \Gamma_1 \cup \Gamma_2 \text{ is a basis;} \\
 \frac{\Gamma \vdash M : B}{\Gamma - \{x:A\} \vdash (\lambda x.M) : (A \rightarrow B)}, \quad \text{provided } \Gamma \cup \{x:A\} \text{ is a basis.}
 \end{array}
 }$$

Provability in this system will be denoted by $\Gamma \vdash' M : A$.

- (iii) Show that \vdash' is closed under α -conversion.

- (iv) Show that

$$\Gamma \vdash' M : A \iff \exists M' \equiv_\alpha M. \Gamma \vdash M' : A.$$

- 1.5.18. (i) Let $M = \lambda x_1 \dots x_n. x_i M_1 \dots M_m$ be a β -nf. Define by induction on the length of M its Φ -normal form, notation $\Phi(M)$, as follows.

$$\Phi(\lambda \vec{x}. x_i M_1 \dots M_m) := \lambda \vec{x}. x_i (\Phi(\lambda \vec{x}. M_1) \vec{x}) \dots (\Phi(\lambda \vec{x}. M_m) \vec{x}).$$

- (ii) Compute the Φ -nf of $S = \lambda xyz. xz(yz)$.

- (iii) Write $\Phi^{n,m,i} := \lambda y_1 \dots y_m \lambda x_1 \dots x_n. x_i (y_1 \vec{x}) \dots (y_m \vec{x})$. Then

$$\Phi(\lambda \vec{x}. x_i M_1 \dots M_m) = \Phi^{n,m,i}(\Phi(\lambda \vec{x}. M_1)) \dots (\Phi(\lambda \vec{x}. M_m)).$$

Show that the $\Phi^{n,m,i}$ are typable.

- (iv) Show that every closed nf of type A can be written as a product of the $\Phi^{n,m,i}$.
 (v) Write S in such a manner.

Chapter 2

Properties

2.1. First properties

In this section we will treat simple properties of the various systems λ_{\rightarrow} . Deeper properties—like strong normalization of typeable terms—will be considered in Section 2.2.

Properties of $\lambda_{\rightarrow}^{\text{Cu}}$, $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{dB}}$

Unless stated otherwise, properties stated for λ_{\rightarrow} apply to both systems.

2.1.1. PROPOSITION (Weakening lemma for λ_{\rightarrow}).

Suppose $\Gamma \vdash M : A$ and Γ' is a basis with $\Gamma \subseteq \Gamma'$. Then $\Gamma' \vdash M : A$.

PROOF. By induction on the derivation of $\Gamma \vdash M : A$. ■

2.1.2. LEMMA (Free variable lemma). (i) *Suppose $\Gamma \vdash M : A$. Then $FV(M) \subseteq \text{dom}(\Gamma)$.*

(ii) *If $\Gamma \vdash M : A$, then $\Gamma \upharpoonright FV(M) \vdash A : M$, where for a set X of variables one has $\Gamma \upharpoonright FV(M) = \{x:A \in \Gamma \mid x \in X\}$.*

PROOF. (i), (ii) By induction on the generation of $\Gamma \vdash M : A$. ■

The following result is related to the fact that the system λ_{\rightarrow} is ‘syntax directed’, i.e. statements $\Gamma \vdash M : A$ have a unique proof.

2.1.3. PROPOSITION (Generation lemma for $\lambda_{\rightarrow}^{\text{Cu}}$).

- (i) $\Gamma \vdash x : A \Rightarrow (x:A) \in \Gamma.$
- (ii) $\Gamma \vdash MN : A \Rightarrow \exists B \in \mathbb{T} [\Gamma \vdash M : B \rightarrow A \ \& \ \Gamma \vdash N : B].$
- (iii) $\Gamma \vdash \lambda x.M : A \Rightarrow \exists B, C \in \mathbb{T} [A \equiv B \rightarrow C \ \& \ \Gamma, x:B \vdash M : C].$

PROOF. (i) Suppose $\Gamma \vdash x : A$ holds in λ_{\rightarrow} . The last rule in a derivation of this statement cannot be an application or an abstraction, since x is not of the right form. Therefore it must be an axiom, i.e. $(x:A) \in \Gamma$.

(ii), (iii) The other two implications are proved similarly. ■

2.1.4. PROPOSITION (Generation lemma for $\lambda_{\rightarrow}^{\text{dB}}$).

- (i) $\Gamma \vdash x : A \Rightarrow (x:A) \in \Gamma.$
- (ii) $\Gamma \vdash MN : A \Rightarrow \exists B \in \mathbb{T} [\Gamma \vdash M : B \rightarrow A \ \& \ \Gamma \vdash N : B].$
- (iii) $\Gamma \vdash \lambda x:B.M : A \Rightarrow \exists C \in \mathbb{T} [A \equiv B \rightarrow C \ \& \ \Gamma, x:B \vdash M : C].$

PROOF. Similarly. ■

2.1.5. PROPOSITION (Generation lemma for $\lambda_{\rightarrow}^{\text{Ch}}$).

- (i) $x^B \in \Lambda_{\rightarrow}^{\text{Ch}}(A) \Rightarrow B = A.$
- (ii) $(MN) \in \Lambda_{\rightarrow}^{\text{Ch}}(A) \Rightarrow \exists B \in \mathbb{T}. [M \in \Lambda_{\rightarrow}^{\text{Ch}}(B \rightarrow A) \ \& \ N \in \Lambda_{\rightarrow}^{\text{Ch}}(B)].$
- (iii) $(\lambda x^B.M) \in \Lambda_{\rightarrow}^{\text{Ch}}(A) \Rightarrow \exists C \in \mathbb{T}. [A = (B \rightarrow C) \ \& \ M \in \Lambda_{\rightarrow}^{\text{Ch}}(C)].$

PROOF. As before. ■

The following two results hold for $\lambda_{\rightarrow}^{\text{Cu}}$ and $\lambda_{\rightarrow}^{\text{dB}}$. Variants already have been proved for $\lambda_{\rightarrow}^{\text{Ch}}$, Propositions 1.4.2 and 1.4.4(iii).

2.1.6. PROPOSITION (Substitution lemma for $\lambda_{\rightarrow}^{\text{Cu}}$ and $\lambda_{\rightarrow}^{\text{dB}}$).

- (i) $\Gamma, x:A \vdash M : B \ \& \ \Gamma \vdash N : A \Rightarrow \Gamma \vdash M[x := N] : B.$
- (ii) $\Gamma \vdash M : A \Rightarrow \Gamma[\alpha := B] \vdash M : A[\alpha := B].$

PROOF. The proof will be given for $\lambda_{\rightarrow}^{\text{Cu}}$, for $\lambda_{\rightarrow}^{\text{dB}}$ it is similar.

(i) By induction on the derivation of $\Gamma, x:A \vdash M : B$. Write $P^* \equiv P[x := N]$.

Case 1. $\Gamma, x:A \vdash M : B$ is an axiom, hence $M \equiv y$ and $(y:B) \in \Gamma \cup \{x:A\}$.

Subcase 1.1. $(y:B) \in \Gamma$. Then $y \neq x$ and $\Gamma \vdash M^* \equiv y[x:N] \equiv y : B$.

Subcase 1.2. $y:B \equiv x:A$. Then $y \equiv x$ and $B \equiv A$, hence $\Gamma \vdash M^* \equiv N : A \equiv B$.

Case 2. $\Gamma, x:A \vdash M : B$ follows from $\Gamma, x:A \vdash F : C \rightarrow B$, $\Gamma, x:A \vdash G : C$ and $FG \equiv M$. By the induction hypothesis one has $\Gamma \vdash F^* : C \rightarrow B$ and $\Gamma \vdash G^* : C$. Hence $\Gamma \vdash (FG)^* \equiv F^*G^* : B$.

Case 3. $\Gamma, x:A \vdash M : B$ follows from $\Gamma, x:A, y:D \vdash G : E$, $B \equiv D \rightarrow E$ and $\lambda y.G \equiv M$. By the induction hypothesis $\Gamma, y:D \vdash G^* : E$, hence $\Gamma \vdash (\lambda y.G)^* \equiv \lambda y.G^* : D \rightarrow E \equiv B$.

(ii) Similarly. ■

2.1.7. PROPOSITION (Subject reduction property for $\lambda_{\rightarrow}^{\text{Cu}}$ and $\lambda_{\rightarrow}^{\text{dB}}$).

Suppose

$M \rightarrow_{\beta\eta} M'$. Then $\Gamma \vdash M : A \Rightarrow \Gamma \vdash M' : A$.

PROOF. The proof will be given for $\lambda_{\rightarrow}^{\text{dB}}$, for $\lambda_{\rightarrow}^{\text{Cu}}$ it is similar. Suppose $\Gamma \vdash M : A$ and $M \rightarrow M'$ in order to show that $\Gamma \vdash M' : A$; then the result follows by induction on the derivation of $\Gamma \vdash M : A$.

Case 1. $\Gamma \vdash M : A$ is an axiom. Then M is a variable, contradicting $M \rightarrow M'$. Hence this case cannot occur.

Case 2. $\Gamma \vdash M : A$ is $\Gamma \vdash FN : A$ and is a direct consequence of $\Gamma \vdash F : B \rightarrow A$ and $\Gamma \vdash N : B$. Since $FN \equiv M \rightarrow M'$ we can have three subcases.

Subcase 2.1. $M' \equiv F'N$ with $F \rightarrow F'$.

Subcase 2.2. $M' \equiv FN'$ with $N \rightarrow N'$.

In these two subcases it follows by the induction hypothesis that $\Gamma \vdash M' : A$.

Subcase 2.3. $F \equiv \lambda x:B.G$ and $M' \equiv G[x = N]$. Since

$$\Gamma \vdash \lambda x.G : B \rightarrow A \text{ \& } \Gamma \vdash N : B$$

it follows by the generation lemma 2.1.3 for λ_{\rightarrow} that

$$\Gamma, x:B \vdash G : A \text{ \& } \Gamma \vdash N : B.$$

Therefore by the substitution lemma 2.1.6 for λ_{\rightarrow} it follows that

$\Gamma \vdash G[x = N] : A$, i.e. $\Gamma \vdash M' : A$.

Case 3. $\Gamma \vdash M : A$ is $\Gamma \vdash \lambda x:B.N : B \rightarrow C$ and follows from $\Gamma, x:B \vdash N : C$. Since $M \rightarrow M'$ we have $M' \equiv \lambda x:B.N'$ with $N \rightarrow N'$. By the induction hypothesis one has $\Gamma, x:B \vdash N' : C$, hence $\Gamma \vdash \lambda x:B.N' : B \rightarrow C$, i.e. $\Gamma \vdash M' : A$. ■

The following result also holds for $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{dB}}$, Exercise 2.5.4.

2.1.8. COROLLARY (Church-Rosser Theorem for $\lambda_{\rightarrow}^{\text{Cu}}$). *On typable terms of $\lambda_{\rightarrow}^{\text{Cu}}$ the Church-Rosser theorem holds for the notions of reduction \rightarrow_{β} and $\rightarrow_{\beta\eta}$.*

(i) Let $M, N \in \Lambda_{\rightarrow}^{\Gamma}(A)$. Then

$$M =_{\beta(\eta)} N \Rightarrow \exists Z \in \Lambda_{\rightarrow}^{\Gamma}(A). M \rightarrow_{\beta(\eta)} Z \text{ \& } N \rightarrow_{\beta(\eta)} Z.$$

(ii) Let $M, N_1, N_2 \in \Lambda_{\rightarrow}^{\Gamma}(A)$. Then

$$M \rightarrow_{\beta\eta} N_1 \text{ \& } M \rightarrow_{\beta(\eta)} N_2 \Rightarrow \exists Z \in \Lambda_{\rightarrow}^{\Gamma}(A). N_1 \rightarrow_{\beta(\eta)} Z \text{ \& } N_2 \rightarrow_{\beta(\eta)} Z.$$

PROOF. By the Church-Rosser theorems for \rightarrow_{β} and $\rightarrow_{\beta\eta}$ on untyped terms, Theorem 1.1.7, and Proposition 2.1.7. ■

The following property of uniqueness of types only holds for the Church and de Bruijn versions of λ_{\rightarrow} . It is instructive to find out where the proof brakes down for $\lambda_{\rightarrow}^{\text{Cu}}$ and also that the two contexts in (ii) should be the same.

2.1.9. PROPOSITION (Unicity of types for $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{dB}}$).

- (i) $M \in \Lambda_{\rightarrow}^{\text{Ch}}(A) \text{ \& } M \in \Lambda_{\rightarrow}^{\text{Ch}}(B) \Rightarrow A = B.$
- (ii) $\Gamma \vdash_{\lambda_{\rightarrow}}^{\text{dB}} M : A \text{ \& } \Gamma \vdash_{\lambda_{\rightarrow}}^{\text{dB}} M : B \Rightarrow A = B.$

PROOF. (i), (ii) By induction on the structure of M , using the generation lemma 2.1.4. ■

Normalization

For several applications, for example for the problem to find all possible inhabitants of a given type, we will need the weak normalization theorem, stating that all typable terms do have a $\beta\eta$ -nf (normal form). The result is valid for all versions of λ_{\rightarrow} and *a fortiori* for the subsystems λ_{\rightarrow}^o . The proof is due to Turing and is published posthumously in Gandy [1980]. In fact all typable terms in these systems are $\beta\eta$ strongly normalizing, which means that all $\beta\eta$ -reductions are terminating. This fact requires more work and will be proved in §12.2.

The notion of ‘abstract reduction system’, see Klop [1992], is useful for the understanding of the proof of the normalization theorem.

2.1.10. DEFINITION. (i) An *abstract reduction system* is a pair (X, \rightarrow_R) , where X is a set and \rightarrow_R is a binary relation on X .

(ii) An element $x \in X$ is said to be in R -normal form (R -nf) if for no $y \in X$ one has $x \rightarrow_R y$.

(iii) (X, R) is called *weakly normalizing* (R -WN, or simply WN) if every element has an R -nf.

(iv) (X, R) is said to be *strongly normalizing* (R -SN, or simply SN) if every R -reduction path

$$x_0 \rightarrow_R x_1 \rightarrow_R x_2 \rightarrow_R \dots$$

is finite.

2.1.11. DEFINITION. (i) A *multiset over nat* can be thought of as a generalized set S in which each element may occur more than once. For example

$$S = \{3, 3, 1, 0\}$$

is a multiset. We say that 3 occurs in S with multiplicity 2; that 1 has multiplicity 1; etcetera.

More formally, the above multiset S can be identified with a function $f \in \mathbb{N}^{\mathbb{N}}$ that is almost everywhere 0, except

$$f(0) = 1, f(1) = 1, f(3) = 2.$$

This S is finite if f has *finite support*, where

$$\text{support}(f) = \{x \in \mathbb{N} \mid f(x) \neq 0\}.$$

(ii) Let $\mathcal{S}(\mathbb{N})$ be the collection of all finite multisets over \mathbb{N} . $\mathcal{S}(\mathbb{N})$ can be identified with $\{f \in \mathbb{N}^{\mathbb{N}} \mid \text{support}(f) \text{ is finite}\}$.

2.1.12. DEFINITION. Let $S_1, S_2 \in \mathcal{S}(\mathbb{N})$. Write

$$S_1 \rightarrow_{\mathcal{S}} S_2$$

if S_2 results from S_1 by replacing some elements (just one occurrence) by finitely many lower elements (in the usual ordering of \mathbb{N}). For example

$$\{3, 3, 1, 0\} \rightarrow_S \{3, 2, 2, 2, 1, 1, 0\}.$$

2.1.13. LEMMA. *We define a particular (non-deterministic) reduction strategy F on $\mathcal{S}(\mathbb{N})$. A multi-set S is contracted to $F(S)$ by taking a maximal element $n \in S$ and replacing it by finitely many numbers $< n$. Then F is a normalizing reduction strategy, i.e. for every $S \in \mathcal{S}(\mathbb{N})$ the \mathcal{S} -reduction sequence*

$$S \rightarrow_S F(S) \rightarrow_S F^2(S) \rightarrow_S \dots$$

is terminating.

PROOF. By induction on the highest number n occuring in S . If $n = 0$, then we are done. If $n = k + 1$, then we can successively replace in S all occurrences of n by numbers $\leq k$ obtaining S_1 with maximal number $\leq k$. Then we are done by the induction hypothesis. ■

In fact $(\mathcal{S}(\mathbb{N}), \rightarrow_S)$ is SN. Although we do not strictly need this fact, we will give even two proofs of it. In the first place it is something one ought to know; in the second place it is instructive to see that the result does not imply that λ_{\rightarrow} satisfies SN.

2.1.14. LEMMA. *The reduction system $(\mathcal{S}(\mathbb{N}), \rightarrow_S)$ is SN.*

We will give two proofs of this lemma. The first one uses ordinals; the second one is from first principles.

PROOF₁. Assign to every $S \in \mathcal{S}(\mathbb{N})$ an ordinal $\#S < \omega^\omega$ as suggested by the following examples.

$$\begin{aligned} \#\{3, 3, 1, 0, 0, 0\} &= 2\omega^3 + \omega + 3; \\ \#\{3, 2, 2, 2, 1, 1, 0\} &= \omega^3 + 3\omega^2 + 2\omega + 1. \end{aligned}$$

More formally, if S is represented by $f \in \mathbb{N}^{\mathbb{N}}$ with finite support, then

$$\#S = \sum_{i \in \mathbb{N}} f(i) \cdot \omega^i.$$

Notice that

$$S_1 \rightarrow_S S_2 \Rightarrow \#S_1 > \#S_2$$

(in the example because $\omega^3 > 3\omega^2 + \omega$). Hence by the well-foundedness of the ordinals the result follows. ■₁

PROOF₂. Define

$$\begin{aligned} \mathcal{F}_k &= \{f \in \mathbb{N}^{\mathbb{N}} \mid \forall n \geq k \ f(n) = 0\}; \\ \mathcal{F} &= \cup_{k \in \mathbb{N}} \mathcal{F}_k. \end{aligned}$$

The set \mathcal{F} is the set of functions with finite support. Define on \mathcal{F} the relation $>$ corresponding to the relation $\rightarrow_{\mathcal{S}}$ for the formal definition of $\mathcal{S}(\mathbb{N})$.

$$f > g \iff f(k) > g(k), \text{ where } k \in \mathbb{N} \text{ is largest such that } f(k) \neq g(k).$$

It is easy to see that $(\mathcal{F}, >)$ is a linear ordering. We will show that it is even a well-ordering, i.e. for every non-empty set $X \subseteq \mathcal{F}$ there is a least element $f_0 \in X$. This implies that there are no infinite descending chains in \mathcal{F} .

To show this claim it suffices to prove that each \mathcal{F}_k is well-ordered, since

$$\dots > (\mathcal{F}_{k+1} \setminus \mathcal{F}_k) > \mathcal{F}_k$$

element-wise. This will be proved by induction on k . If $k = 0$, then this is trivial, since $\mathcal{F}_0 = \{\lambda n.0\}$. Now assume (induction hypothesis) that \mathcal{F}_k is well-ordered in order to show the same for \mathcal{F}_{k+1} . Let $X \subseteq \mathcal{F}_{k+1}$ be non-empty. Define

$$\begin{aligned} X(k) &= \{f(k) \mid f \in X\} \subseteq \mathbb{N}; \\ X_k &= \{f \in X \mid f(k) \text{ minimal in } X(k)\} \subseteq \mathcal{F}_{k+1}; \\ X_k|k &= \{g \in \mathcal{F}_k \mid \exists f \in X_k f|k = g\} \subseteq \mathcal{F}_k, \end{aligned}$$

where

$$\begin{aligned} f|k(i) &= f(i), & \text{if } i < k; \\ &= 0, & \text{else.} \end{aligned}$$

By the induction hypothesis $X_k|k$ has a least element g_0 . Then $g_0 = f_0|k$ for some $f_0 \in X_k$. This f_0 is then the least element of X_k and hence of X . ■₂

2.1.15. REMARK. The second proof shows in fact that if $(D, >)$ is a well-ordered set, then so is $(\mathcal{S}(D), >)$, defined analogously to $(\mathcal{S}(\mathbb{N}), >)$. In fact the argument can be carried out in Peano Arithmetic, showing

$$\vdash_{\mathbf{PA}} \text{TI}(\alpha) \rightarrow \text{TI}(\alpha^\omega),$$

where $\text{TI}(\alpha)$ is the principle of transfinite induction for the ordinal α . Since $\text{TI}(\omega)$ is in fact ordinary induction we have in PA

$$\text{TI}(\omega), \text{TI}(\omega^\omega), \text{TI}(\omega^{\omega^\omega}), \dots$$

This implies that the proof of $\text{TI}(\alpha)$ can be carried out in Peano Arithmetic for every $\alpha < \epsilon_0$. Gentzen [1936] shows that $\text{TI}(\epsilon_0)$, where $\epsilon_0 = \omega^{\omega^{\omega^{\dots}}}$, cannot be carried out in PA.

In order to prove the λ_{\rightarrow} is WN it suffices to work with $\lambda_{\rightarrow}^{\text{Ch}}$. We will use the following notation. We write terms with extra type information, decorating each subterm with its type. For example, instead of $(\lambda x^A.M)N \in \mathbf{term}_B$ we write $(\lambda x^A.M^B)^{A \rightarrow B} N^A$.

2.1.16. DEFINITION. (i) Let $R \equiv (\lambda x^A.M^B)^{A \rightarrow B} N^A$ be a redex. The *depth* of R , notation $\#R$, is defined as follows.

$$\#R = \#(A \rightarrow B)$$

where $\#$ on types is defined inductively by

$$\begin{aligned} \#\alpha &= 0; \\ \#(A \rightarrow B) &= \max(\#A, \#B) + 1. \end{aligned}$$

(ii) To each M in $\lambda_{\rightarrow}^{\text{Ch}}$ we assign a multi-set S_M as follows

$$S_M = \{\#R \mid R \text{ is a redex occurrence in } M\},$$

with the understanding that the multiplicity of R in M is copied in S_M .

In the following example we study how the contraction of one redex can duplicate other redexes or create new redexes.

2.1.17. EXAMPLE. (i) Let R be a redex occurrence in a typed term M . Assume

$$M \xrightarrow{\beta} N,$$

i.e. N results from M by contracting R . This contraction can duplicate other redexes. For example (we write $M[P]$, or $M[P, Q]$ to display subterms of M)

$$(\lambda x.M[x, x])_{R_1} \rightarrow_{\beta} M[R_1, R_1]$$

duplicates the other redex R_1 .

(ii) (J.J. Lévy [1978]) Contraction of a β -redex may also create new redexes. For example

$$\begin{aligned} (\lambda x^{A \rightarrow B}.M[x^{A \rightarrow B} P^A]^C)^{(A \rightarrow B) \rightarrow C} (\lambda y^A.Q^B) &\rightarrow_{\beta} M[(\lambda y^A.Q^B)^{A \rightarrow B} P^A]^C; \\ (\lambda x^A.(\lambda y^B.M[x^A, y^B]^C)^{B \rightarrow C})^{A \rightarrow (B \rightarrow C)} P^A Q^B &\rightarrow_{\beta} (\lambda y^B.M[P^A, y^B]^C)^{B \rightarrow C} Q^B; \\ (\lambda x^{A \rightarrow B}.x^{A \rightarrow B})^{(A \rightarrow B) \rightarrow (A \rightarrow B)} (\lambda y^A.P^B)^{A \rightarrow B} Q^A &\rightarrow_{\beta} (\lambda y^A.P^B)^{A \rightarrow B} Q^A. \end{aligned}$$

2.1.18. LEMMA. Assume $M \xrightarrow{\beta} N$ and let R_1 be a created redex in N . Then $\#R > \#R_1$.

PROOF. In Lévy [1978] it is proved that the three ways of creating redexes in example 2.1.17(ii) are the only possibilities. For a proof do exercise 14.5.3 in B[1984]. In each of three cases we can inspect that the statement holds. ■

2.1.19. THEOREM (Weak normalization theorem for λ_{\rightarrow}). If $M \in \Lambda$ is typable in λ_{\rightarrow} , then M is $\beta\eta$ -WN, i.e. has a $\beta\eta$ -nf.

PROOF. By Proposition 1.4.9(ii) it suffices to show this for terms in $\lambda_{\rightarrow}^{\text{Ch}}$. Note η -reductions decreases the length of a term; moreover, for β -normal terms η -contractions do not create β -redexes. Therefore in order to establish $\beta\eta$ -WN it is sufficient to prove that M has a β -nf.

Define the following β -reduction strategy F . If M is in nf, then $F(M) = M$. Otherwise, let R be the *rightmost redex of maximal depth* n in M . Then

$$F(M) = N$$

where $M \xrightarrow{R}_{\beta} N$. Contracting a redex can only duplicate other redexes that are to the right of that redex. Therefore by the choice of R there can only be redexes of M duplicated in $F(M)$ of depth $< n$. By lemma 2.1.18 redexes created in $F(M)$ by the contraction $M \rightarrow_{\beta} F(M)$ are also of depth $< n$. Therefore in case M is not in β -nf we have

$$S_M \rightarrow_S S_{F(M)}.$$

Since \rightarrow_S is SN, it follows that the reduction

$$M \rightarrow_{\beta} F(M) \rightarrow_{\beta} F^2(M) \rightarrow_{\beta} F^3(M) \rightarrow_{\beta} \dots$$

must terminate in a β -nf. ■

For β -reduction this weak normalization theorem was first proved by Turing, see Gandy [1980b]. The proof does not really need SN for \mathcal{S} -reduction. One may also use the simpler result lemma 2.1.13.

It is easy to see that a different reduction strategy does not yield a \mathcal{S} -reduction chain. For example the two terms

$$\begin{aligned} & (\lambda x^A. y^{A \rightarrow A \rightarrow A} x^A x^A)^{A \rightarrow A} ((\lambda x^A. x^A)^{A \rightarrow A} x^A) \rightarrow_{\beta} \\ & y^{A \rightarrow A \rightarrow A} ((\lambda x^A. x^A)^{A \rightarrow A} x^A) ((\lambda x^A. x^A)^{A \rightarrow A} x^A) \end{aligned}$$

give the multisets $\{1, 1\}$ and $\{1, 1\}$. Nevertheless, SN does hold for all systems λ_{\rightarrow} , as will be proved in Section 2.2. It is an open problem whether ordinals can be assigned in a natural and simple way to terms of λ_{\rightarrow} such that

$$M \rightarrow_{\beta} N \Rightarrow \text{ord}(M) > \text{ord}(N).$$

See Howard [1970] and de Vrijer [1987].

Applications of normalization

We will prove that normal terms inhabiting the represented data types (Bool , Nat , Σ^* and T_B) are standard, i.e. correspond to the intended elements. From WN for λ_{\rightarrow} and the subject reduction theorem it then follows that all inhabitants of the mentioned data types are standard.

2.1.20. PROPOSITION. *Let $M \in \Lambda$ be in nf. Then $M \equiv \lambda x_1 \dots x_n. y M_1 \dots M_m$, with $n, m \geq 0$ and the M_1, \dots, M_m again in nf.*

PROOF. By induction on the structure of M . See Barendregt [1984], proposition 8.3.8 for some details if necessary. ■

2.1.21. PROPOSITION. *Let $\text{Bool} \equiv \text{Bool}_\alpha$, with α a type variable. Then for M in nf one has*

$$\vdash M : \text{Bool} \Rightarrow M \in \{\text{true}, \text{false}\}.$$

PROOF. By repeated use of proposition 2.1.20, the free variable lemma 2.1.2 and the generation lemma for $\lambda_{\rightarrow}^{\text{Cu}}$, proposition 2.1.3, one has the following chain of arguments.

$$\begin{aligned} \vdash M : \alpha \rightarrow \alpha \rightarrow \alpha &\Rightarrow M \equiv \lambda x. M_1 \\ &\Rightarrow x:\alpha \vdash M_1 : \alpha \rightarrow \alpha \\ &\Rightarrow M_1 \equiv \lambda y. M_2 \\ &\Rightarrow x:\alpha, y:\alpha \vdash M_2 : \alpha \\ &\Rightarrow M_2 \equiv x \text{ or } M_2 \equiv y. \end{aligned}$$

So $M \equiv \lambda xy. x \equiv \text{true}$ or $M \equiv \lambda xy. y \equiv \text{false}$. ■

2.1.22. PROPOSITION. *Let $\text{Nat} \equiv \text{Nat}_\alpha$. Then for M in nf one has*

$$\vdash M : \text{Nat} \Rightarrow M \in \{\ulcorner n \urcorner \mid n \in \mathbb{N}\}.$$

PROOF. Again we have

$$\begin{aligned} \vdash M : \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha &\Rightarrow M \equiv \lambda x. M_1 \\ &\Rightarrow x:\alpha \vdash M_1 : (\alpha \rightarrow \alpha) \rightarrow \alpha \\ &\Rightarrow M_1 \equiv \lambda f. M_2 \\ &\Rightarrow x:\alpha, f:\alpha \rightarrow \alpha \vdash M_2 : \alpha. \end{aligned}$$

Now we have

$$\begin{aligned} x:\alpha, f:\alpha \rightarrow \alpha \vdash M_2 : \alpha &\Rightarrow [M_2 \equiv x \vee \\ &[M_2 \equiv f M_3 \ \& \ x:\alpha, f:\alpha \rightarrow \alpha \vdash M_3 : \alpha]]. \end{aligned}$$

Therefore by induction on the structure of M_2 it follows that

$$x:\alpha, f:\alpha \rightarrow \alpha \vdash M_2 : \alpha \Rightarrow M_2 \equiv f^n(x),$$

with $n \geq 0$. So $M \equiv \lambda x f. f^n(x) \equiv \ulcorner n \urcorner$. ■

2.1.23. PROPOSITION. *Let $\text{Sigma}^* \equiv \text{Sigma}_\alpha^*$. Then for M in nf one has*

$$\vdash M : \text{Sigma}^* \Rightarrow M \in \{\underline{w} \mid w \in \Sigma^*\}.$$

PROOF. Again we have

$$\begin{aligned}
\vdash M : \alpha \rightarrow (\alpha \rightarrow \alpha)^k \rightarrow \alpha &\Rightarrow M \equiv \lambda x. N \\
&\Rightarrow x : \alpha \vdash N : (\alpha \rightarrow \alpha)^k \rightarrow \alpha \\
&\Rightarrow N \equiv \lambda a_1. N_1 \ \& \ x : \alpha, a_1 : \alpha \rightarrow \alpha \vdash N_1 : (\alpha \rightarrow \alpha)^{k-1} \rightarrow \alpha \\
&\dots \\
&\Rightarrow N \equiv \lambda a_1 \dots a_k. N \ \& \ x : \alpha, a_1, \dots, a_k : \alpha \rightarrow \alpha \vdash N_k : \alpha \\
&\Rightarrow [N_k \equiv x \vee \\
&\quad [N_k \equiv a_{i_j} N'_k \ \& \ x : \alpha, a_1, \dots, a_k : \alpha \rightarrow \alpha \vdash N'_k : \alpha]] \\
&\Rightarrow N_k \equiv a_{i_1} (a_{i_2} (\dots (a_{i_p} x) \dots)) \\
&\Rightarrow M \equiv \lambda x a_1 \dots a_k. a_{i_1} (a_{i_2} (\dots (a_{i_p} x) \dots)) \\
&\equiv \underline{a_{i_1} a_{i_2} \dots a_{i_p}}. \blacksquare
\end{aligned}$$

Before we can prove that inhabitants of $\text{tree}[\beta]$ are standard, we have to introduce an auxiliary notion.

2.1.24. DEFINITION. Given $t \in T[b_1, \dots, b_n]$ define $[t]^{p,l} \in \Lambda$ as follows.

$$\begin{aligned}
[b_i]^{p,l} &= lb_i; \\
[P(t_1, t_2)]^{p,l} &= p[t_1]^{p,l} [t_2]^{p,l}.
\end{aligned}$$

2.1.25. LEMMA. For $t \in T[b_1, \dots, b_n]$ we have

$$[t] =_{\beta} \lambda pl. [t]^{p,l}.$$

PROOF. By induction on the structure of t .

$$\begin{aligned}
[b_i] &\equiv \lambda pl. lb_i \\
&\equiv \lambda pl. [b_i]^{p,l}; \\
[P(t_1, t_2)] &\equiv \lambda pl. p([t_1] pl) ([t_2] pl) \\
&= \lambda pl. p[t_1]^{p,l} [t_2]^{p,l}, & \text{by the IH,} \\
&\equiv \lambda pl. [P(t_1, t_2)]^{p,l}. \blacksquare
\end{aligned}$$

2.1.26. PROPOSITION. Let $\text{tree}[\beta] \equiv \text{tree}_{\alpha}[\beta]$. Then for M in nf one has

$$b_1, \dots, b_n : \beta \vdash M : \text{tree}[\beta] \Rightarrow M \in \{[t] \mid t \in T[b_1, \dots, b_n]\}.$$

PROOF. We have $\vec{b}:\beta \vdash M : (\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\beta \rightarrow \alpha) \rightarrow \alpha \Rightarrow$

$$\begin{aligned}
&\Rightarrow M \equiv \lambda p.M' \\
&\Rightarrow \vec{b}:\beta, p:\alpha \rightarrow \alpha \rightarrow \alpha \vdash M' : (\beta \rightarrow \alpha) \rightarrow \alpha \\
&\Rightarrow M' \equiv \lambda l.M'' \\
&\Rightarrow \vec{b}:\beta, p:(\alpha \rightarrow \alpha \rightarrow \alpha), l:(\beta \rightarrow \alpha) \vdash M'' : \alpha \\
&\Rightarrow M'' \equiv lb_i \vee [M'' \equiv pM_1M_2 \ \& \\
&\quad \vec{b}:\beta, p:(\alpha \rightarrow \alpha \rightarrow \alpha), l:(\beta \rightarrow \alpha) \vdash M_j : \alpha], \quad j=1,2, \\
&\Rightarrow M'' \equiv [t]^{p,l}, \text{ for some } t \in T[\vec{b}], \\
&\Rightarrow M \equiv \lambda pl.[t]^{p,l} =_\beta [t], \quad \text{by lemma 2.1.25. } \blacksquare
\end{aligned}$$

2.2. Proofs of strong normalization

We now will give two proofs showing that λ_{\rightarrow} is strongly normalizing. The first one is the classical proof due to Tait [1967] that needs little technique, but uses set theoretic comprehension. The second proof due to Statman is elementary, but needs results about reduction.

2.2.1. THEOREM (SN for $\lambda_{\rightarrow}^{\text{Ch}}$). *For all $A \in \Pi_{\infty}$, $M \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$ one has $\text{SN}_{\beta\eta}(M)$.*

PROOF. We use an induction loading. First we add to λ_{\rightarrow} constants $d_{\alpha} \in \Lambda_{\rightarrow}^{\text{Ch}}(\alpha)$ for each atom α , obtaining $\lambda_{\rightarrow}^{\text{Ch}}$. Then we prove SN for the extended system. It follows *a fortiori* that the system without the constants is SN.

One first defines for $A \in \Pi_{\infty}$ the following class \mathcal{C}_A of *computable* terms of type A . We write SN for $\text{SN}_{\beta\eta}$.

$$\begin{aligned}
\mathcal{C}_{\alpha} &= \{M \in \Lambda_{\rightarrow}^{\emptyset}(\alpha) \mid \text{SN}(M)\}; \\
\mathcal{C}_{A \rightarrow B} &= \{M \in \Lambda_{\rightarrow}^{\emptyset}(A \rightarrow B) \mid \forall P \in \mathcal{C}_A. MP \in \mathcal{C}_B\}.
\end{aligned}$$

Then one defines the classes \mathcal{C}_A^* of terms that are *computable under substitution*

$$\mathcal{C}_A^* = \{M \in \Lambda_{\rightarrow}^{\emptyset}(A) \mid \forall \vec{Q} \in \mathcal{C}. [M[\vec{x} = \vec{Q}] \in \Lambda_{\rightarrow}^{\emptyset}(A) \Rightarrow M[\vec{x} = \vec{Q}] \in \mathcal{C}_A]\}.$$

Write $\mathcal{C}^{(*)} = \bigcup \{\mathcal{C}_A^{(*)} \mid A \in \Pi(\lambda_{\rightarrow}^{\text{Ch}})\}$. For $A = A_1 \rightarrow \dots \rightarrow A_n \rightarrow \alpha$ define

$$d_A \equiv \lambda x_1:A_1 \dots \lambda x_n:A_n. d_{\alpha}.$$

Then for A one has

$$M \in \mathcal{C}_A \iff \forall \vec{P} \in \mathcal{C}. M\vec{P} \in \text{SN}, \quad (0)$$

$$M \in \mathcal{C}_A^* \iff \forall \vec{P}, \vec{Q} \in \mathcal{C}. M[\vec{x} = \vec{Q}]\vec{P} \in \text{SN}, \quad (1)$$

where the \vec{P}, \vec{Q} should have the right types and $M\vec{P}$ and $M[\vec{x} = \vec{Q}]\vec{P}$ are of type α , respectively. By an easy simultaneous induction on A one can show

$$M \in \mathcal{C}_A \Rightarrow \text{SN}(M); \quad (2)$$

$$d_A \in \mathcal{C}_A. \quad (3)$$

In particular, since $M[\vec{x} := \vec{P}]\vec{Q} \in \text{SN} \Rightarrow M \in \text{SN}$, it follows that

$$M \in \mathcal{C}^* \Rightarrow M \in \text{SN}. \quad (4)$$

Now one shows by induction on M that

$$M \in \Lambda(A) \Rightarrow M \in \mathcal{C}_A^*. \quad (5)$$

We distinguish cases and use (1).

Case $M \equiv x$. Then for $P, \vec{Q} \in \mathcal{C}$ one has $M[x := P]\vec{Q} \equiv P\vec{Q} \in \mathcal{C} \subseteq \text{SN}$, by the definition of \mathcal{C} and (2).

Case $M \equiv NL$ is easy.

Case $M \equiv \lambda x.N$. Now $\lambda x.N \in \mathcal{C}^*$ iff for all $\vec{P}, Q, \vec{R} \in \mathcal{C}$ one has

$$(\lambda x.N[\vec{y} := \vec{P}])Q\vec{R} \in \text{SN}. \quad (6)$$

If $\vec{P}, Q, \vec{R} \in \mathcal{C} \subseteq \text{SN}$, then by the IH one has $N \in \mathcal{C}^* \subseteq \text{SN}$ so

$$N[x := Q, \vec{y} := \vec{P}]\vec{R} \in \text{SN}. \quad (7)$$

Now every maximal reduction path σ starting from the term in (6) passes through a reduct of the term in (7), as reductions within N, \vec{P}, Q, \vec{R} are finite, hence σ is finite. Therefore we have (6).

Finally by (5) and (4), every typable term of λ_{\rightarrow}^+ , hence of λ_{\rightarrow} , is SN. ■

The idea of the proof is that one would have liked to prove by induction on M that it is SN. But this is not directly possible. One needs the *induction loading* that $M\vec{P} \in \text{SN}$. For a typed system with only combinators this is sufficient and this was the original argument of Tait [1967]. For lambda terms one needs the extra induction loading of being computable under substitution. This argument was first presented by Prawitz [1971] and Girard [1971].

2.2.2. COROLLARY (SN for $\lambda_{\rightarrow}^{\text{Cu}}$). $\forall M \in \Lambda_{\rightarrow}^{\text{Cu}}. \text{SN}_{\beta\eta}(M)$.

PROOF. Suppose $M \in \Lambda$ has type A (with respect to Γ) and had an infinite reduction path σ . By repeated use of Proposition 1.4.9(ii) lift M to $M' \in \Lambda_{\rightarrow}^{\text{Ch}}$ with an infinite reduction path (that projects to σ), contradicting the Theorem. ■

An elementary proof of strong normalization

Now the elementary proof of strong normalization of $\lambda_{\rightarrow}^{\text{Cu}}$ due to Statman will be presented. Inspiration came from Nederpelt, Gandy and Klop. The point of this proof is that in this reduction system strong normalizability follows from normalizability by local structure arguments similar to and in many cases identical to those presented for the untyped lambda calculus in Barendregt [1984]. These include analysis of redex creation,

permutability of head with internal reductions, and permutability of η - with β -redexes. In particular, no special proof technique is needed to obtain strong normalization once normalization has been observed. We assume that the reader is familiar with the untyped lambda calculus

2.2.3. DEFINITION. (i) Let $R \equiv (\lambda x.X)Y$ be a β -redex. Then R is

- (1) βI if $x \in \text{FV}(X)$;
- (2) βK if $x \notin \text{FV}(X)$;
- (3) βK^- if R is βK and $x:0$ and $X : 0$;
- (4) βK^+ if R is a βK and is not a βK^- .

(ii) The term X is said to have the λK^- property if every dummy abstraction $\lambda x.X$ has $x:0$ and $X : 0$.

NOTATION. (i) \rightarrow_β is beta reduction.

- (ii) \rightarrow_η is η reduction.
- (iii) $\rightarrow_{\beta I}$ is reduction of λI -redexes.
- (iv) $\rightarrow_{\beta K^+}$ is reduction of λI - or λK^+ -redexes.
- (v) $\rightarrow_{\beta K^-}$ is reduction of λK^- -redexes.

2.2.4. THEOREM. Every $M \in \Lambda_{\rightarrow}^{\text{Cu}}$ is strongly $\beta\eta$ -normalizable.

PROOF. The result is proved in several steps.

(i) Every term is $\beta(\eta)$ -normalizable. For β this is Theorem 2.1.19.

(ii) β -head reduction sequences terminate. By the standardization theorem, B[1984] Theorem 11.4.7, there is a standard reduction to the β -normal form.

(iii) No β -reduction cycles. Consider a shortest term M beginning a cyclic reduction. By the standardization theorem there exists a standard reduction from M to M . By choice of M this standard reduction contains a head reduction. By permutability of head reductions with internal reductions for each natural number m , M begins a head reduction with at least m steps contradicting (ii).

(iv) $M \twoheadrightarrow_{\beta\eta} N \Rightarrow \exists P.M \twoheadrightarrow_\beta P \twoheadrightarrow_\eta N$ (Church). This usually is referred to as η postponement, see B[1984] Corollary 15.1.5, for a proof.

(v) Strong normalizability of \rightarrow_β implies strong normalizability of $\rightarrow_{\beta\eta}$. Take an infinite $\rightarrow_{\beta\eta}$ sequence and apply η postponement at each step. The result yields an infinite \rightarrow_β sequence.

(vi) Let βI be the notion of reduction in which a β -redex $(\lambda x.M)N$ is only allowed to be contracted if $x \in \text{FV}(M)$. Then βI is weakly normalizing. Same argument as for (i).

(vii) βI is strongly normalizing (Church). See B[1984], Theorem 11.3.7.

(viii) $M \twoheadrightarrow_\beta N \Rightarrow \exists P.M \twoheadrightarrow_{\beta K^+} P \twoheadrightarrow_{\beta K^-} N$ (βK^- -redex postponement). Contraction of βK^- -redexes cannot create new redexes or copy old ones so βK^- -redexes can be permuted with βI and βK^+ -redexes. This permutation can cause the βK^- -redex to be copied several times but it cannot turn an I -redex into a K -redex (indeed the reverse can happen) so no new βK^- -redexes are created.

(ix) If M has the λK^- property then M β -reduces to only finitely many N . This follows by (vii) and (viii).

(x) If M has the λK^- property then M is strongly β -normalizable. By (i), (iii) and (ix).

(xi) If M has the λK^- property then M is strongly $\beta\eta$ -normalizable. By (v) and (x).

(xii) For each M there is an N with the λK^- property such that $N \rightarrow_{\beta\eta} M$. First expand M by η expansion so that every subterm of M beginning with a lambda is a lambda prefix followed by a matrix of type 0. Let $a : \alpha$ and $f : 0 \rightarrow (0 \rightarrow 0)$ be new variables. For each type $T = T_1 \rightarrow \dots \rightarrow T_t \rightarrow \alpha$ with $T_i = T_{i,1} \rightarrow \dots \rightarrow T_{i,k_i} \rightarrow \alpha_i$ for $i = 1, \dots, t$ define terms $U_A : T$ recursively by

$$\begin{aligned} U_0 &= a; \\ U_T &= \lambda x_1 \dots \lambda x_t. f(x_1 U_{T_{1,1}} \dots U_{T_{1,k_1}}) \dots \\ &\quad (f(x_{t-1} U_{T_{t-1,1}} \dots U_{T_{t-1,k_{t-1}}})(x_t U_{T_{t,1}} \dots U_{T_{t,k_t}})) \dots \end{aligned}$$

Now recursively replace each dummy λx occurring $\lambda x \lambda y \dots \lambda z. X$ with $x : T$ and $X : 0$ by $\lambda x \lambda y \dots \lambda z. KX(x U_{T_1} \dots U_{T_t})$. Clearly the resulting N satisfies $N \rightarrow_{\beta\eta} M$ and the λK^- property, since all dummy lambdas appear in $K : 1_2$.

(xiii) Every typable term is strongly $\beta\eta$ normalizable. By (xi) and (xii). ■

Still another proof is to be found in de Vrijer [1987] in which for a typed term M a computation is given of the longest reduction path to β -nf.

2.3. Checking and finding types

There are several natural problems concerning type systems.

2.3.1. DEFINITION. (i) The problem of *type checking* consists of determining, given basis Γ , term M and type A whether $\Gamma \vdash M : A$.

(ii) The problem of *typeability* consists of given a term M determining whether M has some type with respect to some Γ .

(iii) The problem of *type reconstruction* ('finding types') consists of finding all possible types A and bases Γ that type a given M .

(iv) The *inhabitation problem* consists of finding out whether a given type A is inhabited by some term M in a given basis Γ .

(v) The *enumeration problem* consists of determining for a given type A and a given context Γ all possible terms M such that $\Gamma \vdash M : A$.

The five problems may be summarized stylistically as follows.

$$\begin{array}{lll} \Gamma \vdash_{\lambda\rightarrow} M : A ? & \text{type checking;} \\ \exists A, \Gamma [\Gamma \vdash_{\lambda\rightarrow} M : A] ? & \text{typeability;} \\ ? \vdash_{\lambda\rightarrow} M : ? & \text{type reconstruction;} \\ \exists M [\Gamma \vdash_{\lambda\rightarrow} M : A] ? & \text{inhabitation;} \\ \Gamma \vdash_{\lambda\rightarrow} ? : A & \text{enumeration.} \end{array}$$

In another notation this is the following.

$$\begin{array}{ll}
M \in \Lambda_{\rightarrow}^{\Gamma}(A)? & \text{type checking;} \\
\exists A, \Gamma \quad M \in \Lambda_{\rightarrow}^{\Gamma}(A)? & \text{typeability;} \\
M \in \Lambda_{\rightarrow}^? (?) & \text{type reconstruction;} \\
\Lambda_{\rightarrow}^{\Gamma}(A) \neq \emptyset? & \text{inhabitation;} \\
? \in \Lambda_{\rightarrow}^{\Gamma}(A) & \text{enumeration.}
\end{array}$$

In this section we will treat the problems of type checking, typeability and type reconstruction for the three versions of λ_{\rightarrow} . It turns out that these problems are decidable for all versions. The solutions are essentially simpler for $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{dB}}$ than for $\lambda_{\rightarrow}^{\text{Cu}}$. The problems of inhabitation and enumeration will be treated in the next section.

One may wonder what is the role of the context Γ in these questions. The problem

$$\exists \Gamma \exists A \Gamma \vdash M : A.$$

can be reduced to one without a context. Indeed, for $\Gamma = \{x_1:A_1, \dots, x_n:A_n\}$

$$\Gamma \vdash M : A \Leftrightarrow \vdash (\lambda x_1(:A_1) \dots \lambda x_n(:A_n).M) : (A_1 \rightarrow \dots \rightarrow A_n \rightarrow A).$$

Therefore

$$\exists \Gamma \exists A [\Gamma \vdash M : A] \Leftrightarrow \exists B [\vdash \lambda \vec{x}.M : B].$$

On the other hand the question

$$\exists \Gamma \exists M [\Gamma \vdash M : A]?$$

is trivial: take $\Gamma = \{x:A\}$ and $M \equiv x$. So we do not consider this question.

The solution of the problems like type checking for a fixed context will have important applications for the treatment of constants.

Checking and finding types for $\lambda_{\rightarrow}^{\text{dB}}$ and $\lambda_{\rightarrow}^{\text{Ch}}$

We will see again that the systems $\lambda_{\rightarrow}^{\text{Ch}}$ and $\lambda_{\rightarrow}^{\text{dB}}$ are essentially equivalent. For these systems the solutions to the problems of type checking, typeability and type reconstruction are easy. All of the solutions are computable with an algorithm of linear complexity.

2.3.2. PROPOSITION (Type checking for $\lambda_{\rightarrow}^{\text{dB}}$). *Let Γ be a basis of $\lambda_{\rightarrow}^{\text{dB}}$. Then there is a computable function $\text{type}_{\Gamma} : \Lambda_{\Pi} \rightarrow \Pi \cup \{\text{error}\}$ such that*

$$M \in \Lambda_{\rightarrow}^{\Gamma}(\text{Ch}(A)) \Leftrightarrow \text{type}_{\Gamma}(M) = A.$$

PROOF. Define

$$\begin{array}{ll}
\text{type}_{\Gamma}(x) &= \Gamma(x); \\
\text{type}_{\Gamma}(MN) &= B, & \text{if } \text{type}_{\Gamma}(M) = \text{type}_{\Gamma}(N) \rightarrow B, \\
&= \text{error}, & \text{else;} \\
\text{type}_{\Gamma}(\lambda x:A.M) &= A \rightarrow \text{type}_{\Gamma \cup \{x:A\}}(M), & \text{if } \text{type}_{\Gamma \cup \{x:A\}}(M) \neq \text{error}, \\
&= \text{error}, & \text{else.}
\end{array}$$

Then the statement follows by induction on the structure of M . ■

2.3.3. COROLLARY. *Typeability and type reconstruction for $\lambda_{\rightarrow}^{\text{dB}}$ are computable. In fact one has the following.*

- (i) $M \in \Lambda_{\rightarrow}^{\Gamma} \text{dB} \iff \text{type}_{\Gamma}(M) \neq \text{error}$.
- (ii) *Each $M \in \Lambda_{\rightarrow}^{\Gamma}(\text{type}_{\Gamma})$ has a unique type; in particular*

$$M \in \Lambda_{\rightarrow}^{\Gamma}(\text{type}_{\Gamma}(M)).$$

PROOF. By the proposition. ■

For $\lambda_{\rightarrow}^{\text{Ch}}$ things are essentially the same, except that there are no bases needed, since variables come with their own types.

2.3.4. PROPOSITION (Type checking for $\lambda_{\rightarrow}^{\text{Ch}}$). *There is a computable function $\text{type} : \Lambda_{\rightarrow}^{\text{Ch}} \rightarrow \mathbb{T} \cup \{\text{error}\}$ such that*

$$M \in \Lambda_{\rightarrow}^{\text{Ch}}(A) \iff \text{type}(M) = A.$$

PROOF. Define

$$\begin{aligned} \text{type}(x^A) &= A; \\ \text{type}(MN) &= B, & \text{if } \text{type}(M) = \text{type}(N) \rightarrow B, \\ &= \text{error}, & \text{else;} \\ \text{type}(\lambda x^A.M) &= A \rightarrow \text{type}(M), & \text{if } \text{type}(M) \neq \text{error}, \\ &= \text{error}, & \text{else.} \end{aligned}$$

Then the statement follows again by induction on the structure of M . ■

2.3.5. COROLLARY. *Typeability and type reconstruction for $\lambda_{\rightarrow}^{\text{Cu}}$ are computable. In fact one has the following.*

- (i) $M \in \Lambda_{\rightarrow}^{\text{Cu}} \iff \text{type}(M) \neq \text{error}$.
- (ii) *Each $M \in \Lambda_{\rightarrow}^{\text{Cu}}$ has a unique type; in particular $M \in \Lambda_{\rightarrow}^{\text{Cu}}(\text{type}(M))$.*

PROOF. By the proposition. ■

Checking and finding types for $\lambda_{\rightarrow}^{\text{Cu}}$

We now will show the computability of the three questions for $\lambda_{\rightarrow}^{\text{Cu}}$. This occupies 2.3.6 - 2.3.16 and in these items \vdash stands for $\vdash_{\lambda_{\rightarrow}^{\text{Cu}}}$.

Let us first make the easy observation that in $\lambda_{\rightarrow}^{\text{Cu}}$ types are not unique. For example $I \equiv \lambda x.x$ has as possible type $\alpha \rightarrow \alpha$, but also $(\beta \rightarrow \beta) \rightarrow (\beta \rightarrow \beta)$ and $(\alpha \rightarrow \beta \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta \rightarrow \beta)$. Of these types $\alpha \rightarrow \alpha$ is the ‘most general’ in the sense that the other ones can be obtained by a substitution in α .

2.3.6. DEFINITION. (i) A substitutor is an operation $*$: $\mathbb{T} \rightarrow \mathbb{T}$ such that

$$*(A \rightarrow B) \equiv *(A) \rightarrow *(B).$$

(ii) We write A^* for $*(A)$.

(iii) Usually a substitution $*$ has a finite support, that is, for all but finitely many type variables α one has $\alpha^* \equiv \alpha$ (the support of $*$ being

$$\text{sup}(*) = \{\alpha \mid \alpha^* \not\equiv \alpha\}.$$

In that case we write

$$*(A) = A[\alpha_1 := \alpha_1^*, \dots, \alpha_n := \alpha_n^*],$$

where $\{\alpha_1, \dots, \alpha_n\} \supseteq \text{sup}(*)$. We also write

$$* = [\alpha_1 := \alpha_1^*, \dots, \alpha_n := \alpha_n^*]$$

and

$$* = []$$

for the identity substitution.

2.3.7. DEFINITION. (i) Let $A, B \in \mathbb{T}$. A *unifier* for A and B is a substitutor $*$ such that $A^* \equiv B^*$.

(ii) The substitutor $*$ is a *most general unifier* for A and B if

- $A^* \equiv B^*$
- $A^{*1} \equiv B^{*1} \Rightarrow \exists *_2 . *_1 \equiv *_2 \circ *$.

(iii) Let $E = \{A_1 = B_1, \dots, A_n = B_n\}$ be a finite set of equations between types. The equations do not need to be valid. A *unifier* for E is a substitutor $*$ such that $A_1^* \equiv B_1^* \& \dots \& A_n^* \equiv B_n^*$. In that case one writes $* \models E$. Similarly one defines the notion of a *most general unifier* for E .

2.3.8. EXAMPLES. The types $\beta \rightarrow (\alpha \rightarrow \beta)$ and $(\gamma \rightarrow \gamma) \rightarrow \delta$ have a unifier. For example $* = [\beta := \gamma \rightarrow \gamma, \delta := \alpha \rightarrow (\gamma \rightarrow \gamma)]$ or $*_1 = [\beta := \gamma \rightarrow \gamma, \alpha := \varepsilon \rightarrow \varepsilon, \delta := \varepsilon \rightarrow \varepsilon \rightarrow (\gamma \rightarrow \gamma)]$. The unifier $*$ is most general, $*_1$ is not.

2.3.9. DEFINITION. A is a *variant* of B if for some $*_1$ and $*_2$ one has

$$A = B^{*1} \text{ and } B = A^{*2}.$$

2.3.10. EXAMPLE. $\alpha \rightarrow \beta \rightarrow \beta$ is a variant of $\gamma \rightarrow \delta \rightarrow \delta$ but not of $\alpha \rightarrow \beta \rightarrow \alpha$.

Note that if $*_1$ and $*_2$ are both most general unifiers of say A and B , then A^{*1} and A^{*2} are variants of each other and similarly for B .

The following result due to Robinson (1965) states that unifiers can be constructed effectively.

2.3.11. THEOREM (Unification theorem). (i) *There is a recursive function U having (after coding) as input a pair of types and as output either a substitutor or **fail** such that*

$$\begin{aligned} A \text{ and } B \text{ have a unifier} &\Rightarrow U(A, B) \text{ is a most general unifier} \\ &\quad \text{for } A \text{ and } B; \\ A \text{ and } B \text{ have no unifier} &\Rightarrow U(A, B) = \mathbf{fail}. \end{aligned}$$

(ii) *There is (after coding) a recursive function U having as input finite sets of equations between types and as output either a substitutor or **fail** such that*

$$\begin{aligned} E \text{ has a unifier} &\Rightarrow U(E) \text{ is a most general unifier for } E; \\ E \text{ has no unifier} &\Rightarrow U(E) = \mathbf{fail}. \end{aligned}$$

PROOF. Note that $A_1 \rightarrow A_2 \equiv B_1 \rightarrow B_2$ holds iff $A_1 \equiv B_1$ and $A_2 \equiv B_2$ hold.

(i) Define $U(A, B)$ by the following recursive loop, using case distinction.

$$\begin{aligned} U(\alpha, B) &= [\alpha := B], & \text{if } \alpha \notin \text{FV}(B), \\ &= [], & \text{if } B = \alpha, \\ &= \mathbf{fail}, & \text{else;} \end{aligned}$$

$$\begin{aligned} U(A_1 \rightarrow A_2, \alpha) &= U(\alpha, A_1 \rightarrow A_2); \\ U(A_1 \rightarrow A_2, B_1 \rightarrow B_2) &= U(A_1^{U(A_2, B_2)}, B_1^{U(A_2, B_2)}) \circ U(A_2, B_2), \end{aligned}$$

where this last expression is considered to be **fail** if one of its parts is. Let $\#_{\text{var}}(A, B)$ = ‘the number of variables in $A \rightarrow B$ ’ and $\#_{\rightarrow}(A, B)$ = ‘the number of arrows in $A \rightarrow B$ ’. By induction on $(\#_{\text{var}}(A, B), \#_{\rightarrow}(A, B))$ ordered lexicographically one can show that $U(A, B)$ is always defined. Moreover U satisfies the specification.

(ii) If $E = \{A_1 = B_1, \dots, A_n = B_n\}$, then define $U(E) = U(A, B)$, where $A = A_1 \rightarrow \dots \rightarrow A_n$ and $B = B_1 \rightarrow \dots \rightarrow B_n$. ■

See [??] for more on unification. The following result due to Parikh [1973] for propositional logic (interpreted by the propositions-as-types interpretation) and Wand [1987] simplifies the proof of the decidability of type checking and typeability for λ_{\rightarrow} .

2.3.12. PROPOSITION. *For every basis Γ , term $M \in \Lambda$ and $A \in \mathbb{T}$ such that $\text{FV}(M) \subseteq \text{dom}(\Gamma)$ there is a finite set of equations $E = E(\Gamma, M, A)$ such that for all substitutors $*$ one has*

$$* \models E(\Gamma, M, A) \Rightarrow \Gamma^* \vdash M : A^*, \tag{1}$$

$$\Gamma^* \vdash M : A^* \Rightarrow *_1 \models E(\Gamma, M, A), \tag{2}$$

*for some $*_1$ such that $*$ and $*_1$ have the same effect on the type variables in Γ and A .*

PROOF. Define $E(\Gamma, M, A)$ by induction on the structure of M :

$$\begin{aligned} E(\Gamma, x, A) &= \{A = \Gamma(x)\}; \\ E(\Gamma, MN, A) &= E(\Gamma, M, \alpha \rightarrow A) \cup E(\Gamma, N, \alpha), \\ &\quad \text{where } \alpha \text{ is a fresh variable;} \\ E(\Gamma, \lambda x. M, A) &= E(\Gamma \cup \{x:\alpha\}, M, \beta) \cup \{\alpha \rightarrow \beta = A\}, \\ &\quad \text{where } \alpha, \beta \text{ are fresh.} \end{aligned}$$

By induction on M one can show (using the generation lemma (2.1.3)) that (1) and (2) hold. ■

2.3.13. DEFINITION. (i) Let $M \in \Lambda$. Then (Γ, A) is a *principal pair* (pp) for M if

- (1) $\Gamma \vdash M : A$.
- (2) $\Gamma' \vdash M : A' \Rightarrow \exists * [\Gamma^* \subseteq \Gamma' \ \& \ A^* \equiv A']$.

Here $\{x_1:A_1, \dots\}^* = \{x_1:A_1^*, \dots\}$.

(ii) Let $M \in \Lambda$ be closed. Then A is a *principal type* (pt) for M if

- (1) $\vdash M : A$
- (2) $\vdash M : A' \Rightarrow \exists * [A^* \equiv A']$.

Note that if (Γ, A) is a *pp* for M , then every variant (Γ', A') of (Γ, A) , in the obvious sense, is also a *pp* for M . Conversely if (Γ, A) and (Γ', A') are *pp*'s for M , then (Γ', A') is a variant of (Γ, A) . Similarly for closed terms and *pt*'s. Moreover, if (Γ, A) is a *pp* for M , then $\text{FV}(M) = \text{dom}(\Gamma)$.

The following result is independently due to Curry (1969), Hindley (1969) and Milner (1978). It shows that for λ_{\rightarrow} the problems of type checking and typeability are decidable.

2.3.14. THEOREM (Principal type theorem for $\lambda_{\rightarrow}^{\text{Cu}}$). (i) *There exists a computable function pp such that one has*

$$\begin{aligned} M \text{ has a type} &\Rightarrow pp(M) = (\Gamma, A), \text{ where } (\Gamma, A) \text{ is a pp for } M; \\ M \text{ has no type} &\Rightarrow pp(M) = \mathbf{fail}. \end{aligned}$$

(ii) *There exists a computable function pt such that for closed terms M one has*

$$\begin{aligned} M \text{ has a type} &\Rightarrow pt(M) = A, \text{ where } A \text{ is a pt for } M; \\ M \text{ has no type} &\Rightarrow pt(M) = \mathbf{fail}. \end{aligned}$$

PROOF. (i) Let $\text{FV}(M) = \{x_1, \dots, x_n\}$ and set $\Gamma_0 = \{x_1:\alpha_1, \dots, x_n:\alpha_n\}$ and $A_0 = \beta$. Note that

$$\begin{aligned} M \text{ has a type} &\Rightarrow \exists \Gamma \exists A \ \Gamma \vdash M : A \\ &\Rightarrow \exists * \ \Gamma_0^* \vdash M : A_0^* \\ &\Rightarrow \exists * \ * \models E(\Gamma_0, M, A_0). \end{aligned}$$

Define

$$\begin{aligned} pp(M) &= (\Gamma_0^*, A_0^*), & \text{if } U(E(\Gamma_0, M, A_0)) = *; \\ &= \mathbf{fail}, & \text{if } U(E(\Gamma_0, M, A_0)) = \mathbf{fail}. \end{aligned}$$

Then $pp(M)$ satisfies the requirements. Indeed, if M has a type, then

$$U(E(\Gamma_0, M, A_0)) = *$$

is defined and $\Gamma_0^* \vdash M : A_0^*$ by (1) in proposition 2.3.12. To show that (Γ_0^*, A_0^*) is a pp, suppose that also $\Gamma' \vdash M : A'$. Let $\tilde{\Gamma} = \Gamma' \upharpoonright \text{FV}(M)$; write $\tilde{\Gamma} = \Gamma_0^{*0}$ and $A' = A_0^{*0}$. Then also $\Gamma_0^{*0} \vdash M : A_0^{*0}$. Hence by (2) in proposition 2.3.12 for some $*_1$ (acting the same as $*_0$ on Γ_0, A_0) one has $*_1 \models E(\Gamma_0, M, A_0)$. Since $*$ is a most general unifier (proposition 2.3.11) one has $*_1 = *_2 \circ *$ for some $*_2$. Now indeed

$$(\Gamma_0^*)^{*_2} = \Gamma_0^{*1} = \Gamma_0^{*0} = \tilde{\Gamma} \subseteq \Gamma'$$

and

$$(A_0^*)^{*_2} = A_0^{*1} = A_0^{*0} = A'.$$

If M has no type, then $\neg \exists * \cdot * \models E(\Gamma_0, M, A_0)$ hence

$$U(\Gamma_0, M, A_0) = \mathbf{fail} = pp(M).$$

(ii) Let M be closed and $pp(M) = (\Gamma, A)$. Then $\Gamma = \emptyset$ and we can put $pt(M) = A$. ■

2.3.15. COROLLARY. *Type checking and typeability for λ_{\rightarrow} are decidable.*

PROOF. As to type checking, let M and A be given. Then

$$\vdash M : A \iff \exists * [A = pt(M)^*].$$

This is decidable (as can be seen using an algorithm—*pattern matching*—similar to the one in Theorem 2.3.11).

As to the question of typeability, let M be given. Then M has a type iff $pt(M) \neq \mathbf{fail}$. ■

The following result is due to Hindley [1969].

2.3.16. THEOREM (Second principal type theorem for $\lambda_{\rightarrow}^{\text{Cu}}$). (i) *For every type $A \in \mathbb{T}$ one has*

$$\vdash M : A \Rightarrow \exists M' [M' \twoheadrightarrow_{\beta\eta} M \ \& \ \mathbf{pt}(M') = A].$$

(ii) *For every type $A \in \mathbb{T}$ there exists a basis Γ and term $M \in \Lambda$ such that (Γ, A) is a pp for M .*

PROOF. (i) We present a proof by examples. We choose three situations in which we have to construct an M' that are representative for the general case. Do exercise ?? for the general proof.

Case $M \equiv \lambda x.x$ and $A \equiv (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$. Then $\mathbf{pt}(M) \equiv \alpha \rightarrow \alpha$. Take $M' \equiv \lambda xy.xy$. The η -expansion of $\lambda x.x$ to $\lambda xy.xy$ makes subtypes of A correspond to unique subterms of M' .

Case $M \equiv \lambda xy.y$ and $A \equiv (\alpha \rightarrow \gamma) \rightarrow \beta \rightarrow \beta$. Then $\mathbf{pt}(M) \equiv \alpha \rightarrow \beta \rightarrow \beta$. Take $M' \equiv \lambda xy.Ky(\lambda z.xz)$. The β -expansion forces x to have a functional type.

Case $M \equiv \lambda xy.x$ and $A \equiv \alpha \rightarrow \alpha \rightarrow \alpha$. Then $\mathbf{pt}(M) \equiv \alpha \rightarrow \beta \rightarrow \alpha$. Take $M' \equiv \lambda xy.Kx(\lambda f.[fx, fy])$. The β -expansion forces x and y to have the same types.

(ii) Let A be given. We know that $\vdash I : A \rightarrow A$. Therefore by (i) there exists an $I' \rightarrow_{\beta\eta} I$ such that $\mathbf{pt}(I') = A \rightarrow A$. Then take $M \equiv I'x$. We have $\mathbf{pp}(I'x) = (\{x:A\}, A)$. ■

Complexity

The space and time complexity of finding a type for a typable term is exponential, see exercise 2.5.18.

In order to decide whether for two typed terms $M, N \in \Lambda_{\rightarrow}(A)$ one has

$$M =_{\beta\eta} N,$$

one can normalize both terms and see whether the results are syntactically equal (up to α -conversion). In exercise 2.5.17 it will be shown that the time and space costs of doing this is at least hyper-exponential (in the size of MN). The reason is that the type-free application of Church numerals

$$\mathbf{c}_n \mathbf{c}_m = \mathbf{c}_{m^n}$$

can be typed, even when applied iteratively

$$\mathbf{c}_{n_1} \mathbf{c}_{n_2} \dots \mathbf{c}_{n_k}.$$

In exercise 2.5.16 it is shown that the costs are also at most hyper-exponential. The reason is that Turing's proof of normalization for terms in λ_{\rightarrow} uses a successive development of redexes of 'highest' type. Now the length of each such development depends exponentially on the length of the term, whereas the length of a term increases at most quadratically at each reduction step. The result even holds for typable terms $M, N \in \Lambda_{\rightarrow\text{Cu}}(A)$, as the cost of finding types only adds a simple exponential to the cost.

One may wonder whether there is not a more efficient way to decide $M =_{\beta\eta} N$, for example by using memory for the reduction of the terms, rather than a pure reduction strategy that only depends on the state of the term reduced so far. The sharpest question is whether there is any Turing computable method, that has a better complexity class. In Statman [1979] it is shown that this is not the case, by showing that every elementary time bounded Turing machine computation can be coded as a convertibility problem for terms of some type in λ_{\rightarrow}^0 . A shorter proof of this result can be found in Mairson [1992].

2.4. Finding inhabitants

In this section we study for λ_{\rightarrow} the problem of finding inhabitants. That is, given a type A , we look for terms M such that in the empty context $\vdash_{\lambda_{\rightarrow}} M : A$. By Corollaries 1.4.8 and 1.4.16 it does not matter whether we work in the system *à la* Curry, Church or de Bruijn. Therefore we will focus on $\lambda_{\rightarrow}^{\text{Cu}}$. Note that by proposition 2.1.2 the term M must be closed.

For example, if $A = \alpha \rightarrow \alpha$, then we can take $M \equiv \lambda x(:\alpha).x$. In fact we will see later that this M is modulo β -conversion the only choice. For $A = \alpha \rightarrow \alpha \rightarrow \alpha$ there are two inhabitants: $M_1 \equiv \lambda x_1 x_2.x_1 \equiv \mathbf{K}$ and $M_2 \equiv \lambda x_1 x_2.x_2 \equiv \mathbf{K}_*$. Again we have exhausted all inhabitants. If $A = \alpha$, then there are no inhabitants, as we will see soon.

Various interpretations will be useful to solve inhabitation problems.

The Boolean model

Type variables can be interpreted as ranging over $\mathbf{B} = \{0, 1\}$ and \rightarrow as the two-ary function on \mathbf{B} defined by

$$x \rightarrow y = 1 - x + xy$$

(classical implication). This makes every type A into a Boolean function. More formally this is done as follows.

2.4.1. DEFINITION. (i) A *Boolean valuation* is a map $\rho : \mathbb{T} \text{ var} \rightarrow \mathbf{B}$.

(ii) Let ρ be a Boolean valuation. The *Boolean interpretation under ρ* of a type $A \in \mathbb{T}(\lambda_{\rightarrow})$, notation $\llbracket A \rrbracket_{\rho}$, is defined inductively as follows.

$$\begin{aligned} \llbracket \alpha \rrbracket_{\rho} &= \rho(\alpha); \\ \llbracket A_1 \rightarrow A_2 \rrbracket_{\rho} &= \llbracket A_1 \rrbracket_{\rho} \rightarrow \llbracket A_2 \rrbracket_{\rho}. \end{aligned}$$

(iii) A Boolean valuation ρ *satisfies a type A* , notation $\rho \models A$, if $\llbracket A \rrbracket_{\rho} = 1$. Let $\Gamma = \{x_1 : A_1, \dots, x_n : A_n\}$, then ρ *satisfies Γ* , notation $\rho \models \Gamma$, if

$$\rho \models A_1 \ \& \ \dots \ \& \ \rho \models A_n.$$

(iv) A type A is *classically valid*, notation $\models A$, iff for all Boolean valuations ρ one has $\rho \models A$.

2.4.2. PROPOSITION. *Let $\Gamma \vdash_{\lambda_{\rightarrow}} M : A$. Then for all Boolean valuations ρ one has*

$$\rho \models \Gamma \Rightarrow \rho \models A.$$

PROOF. By induction on the derivation in λ_{\rightarrow} . ■

From this it follows that inhabited types are classically valid. This in turn implies that the type α is not inhabited.

2.4.3. COROLLARY. (i) *If A is inhabited, then $\models A$.*

(ii) A type variable α is not inhabited.

PROOF. (i) Immediate by proposition 2.4.2, by taking $\Gamma = \emptyset$.

(ii) Immediate by (i), by taking $\rho(\alpha) = 0$. ■

One may wonder whether the converse of 2.4.3(i), i.e.

$$\models A \Rightarrow A \text{ is inhabited} \quad (1)$$

holds. We will see that in λ_{\rightarrow} this is not the case. For the subsystem λ_{\rightarrow}^o (having only one base type o), however, the implication (1) is valid.

2.4.4. PROPOSITION (Statman [1982]). *Let $A = A_1 \rightarrow \dots \rightarrow A_n \rightarrow o$, with $n \geq 1$ be a type of λ_{\rightarrow}^o . Then*

$$A \text{ is inhabited} \iff \text{for some } i \text{ with } 1 \leq i \leq n \text{ the type } A_i \text{ is not inhabited.}$$

PROOF. (\Rightarrow) Assume $\vdash_{\lambda_{\rightarrow}^o} M : A$. Suppose towards a contradiction that all A_i are inhabited, i.e. $\vdash_{\lambda_{\rightarrow}^o} N_i : A_i$. Then $\vdash_{\lambda_{\rightarrow}^o} MN_1 \dots N_n : o$, contradicting 2.4.3(ii).

(\Leftarrow) By induction on the structure of A . Assume that A_i with $1 \leq i \leq n$ is not inhabited.

Case 1. $A_i = o$. Then

$$x_1 : A_1, \dots, x_n : A_n \vdash x_i : o$$

so

$$\vdash (\lambda x_1 \dots x_n. x_i) : A_1 \rightarrow \dots \rightarrow A_n \rightarrow o,$$

i.e. A is inhabited.

Case 2. $A_i = B_1 \rightarrow \dots \rightarrow B_m \rightarrow o$. By (the contrapositive of) the induction hypothesis applied to A_i it follows that all B_j are inhabited, say $\vdash M_j : B_j$. Then

$$\begin{aligned} & x_1 : A_1, \dots, x_n : A_n \vdash x_i : A_i = B_1 \rightarrow \dots \rightarrow B_m \rightarrow o \\ \Rightarrow & x_1 : A_1, \dots, x_n : A_n \vdash x_i M_1 \dots M_m : o \\ \Rightarrow & \vdash \lambda x_1 \dots x_n. x_i M_1 \dots M_m : A_1 \rightarrow \dots \rightarrow A_n \rightarrow o = A. \blacksquare \end{aligned}$$

From the proposition it easily follows that inhabitation of terms in λ_{\rightarrow}^o is decidable with a linear time algorithm.

2.4.5. COROLLARY. *In λ_{\rightarrow}^o one has for all types A*

$$A \text{ is inhabited} \iff \models A.$$

PROOF. (\Rightarrow) By proposition 2.4.3(i). (\Leftarrow) Assume $\models A$ and that A is not inhabited. Then $A = A_1 \rightarrow \dots \rightarrow A_n \rightarrow o$ with each A_i inhabited. But then for $\rho_0(o) = 0$ one has

$$\begin{aligned} 1 &= \llbracket A \rrbracket_{\rho_0} \\ &= \llbracket A_1 \rrbracket_{\rho_0} \rightarrow \dots \rightarrow \llbracket A_n \rrbracket_{\rho_0} \rightarrow o \\ &= 1 \rightarrow \dots \rightarrow 1 \rightarrow 0, \text{ since } \models A_i \text{ for all } i, \\ &= 0, \text{ since } 1 \rightarrow 0 = 0, \end{aligned}$$

contradiction. ■

Corollary 2.4.5 does not hold for $\lambda_{\rightarrow}^{\infty}$. In fact the type $((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha$ (corresponding to Peirce's law) is a valid type that is not inhabited, as we will see soon.

2.4.6. EXERCISE. Each type A of λ_{\rightarrow}^o can be interpreted as an element $\llbracket A \rrbracket \in \mathbf{B}^{\mathbf{B}}$ as follows.

$$\llbracket A \rrbracket(i) = \llbracket A \rrbracket_{\rho_i},$$

where $\rho_i(o) = i$. There are four elements in $\mathbf{B}^{\mathbf{B}}$

$$\{\lambda x \in \mathbf{B}. 0, \lambda x \in \mathbf{B}. 1, \lambda x \in \mathbf{B}. x, \lambda x \in \mathbf{B}. 1 - x\}.$$

Prove that $\llbracket A \rrbracket = \lambda x \in \mathbf{B}. 1$ iff A is inhabited and $\llbracket A \rrbracket = \lambda x \in \mathbf{B}. x$ iff A is not inhabited.

Intuitionistic propositional logic

Although inhabited types correspond to Boolean tautologies, not all such tautologies correspond to inhabited types. Intuitionistic logic provides a precise characterization of inhabited types. The underlying idea, the *propositions-as-types* correspondence will be explained in more detail in §5.2 and then in §30.1.

2.4.7. DEFINITION. (i) The set of formulas of the implicational fragment of propositional logic, notation $\text{form}(\text{PROP})$, is defined by the following abstract syntax. Write $\text{form} = \text{form}(\text{PROP})$.

$\begin{aligned} \text{form} &= \text{var} \mid \text{form} \supset \text{form} \\ \text{var} &= p \mid \text{var}' \end{aligned}$
--

For example $p', p' \supset p, p' \supset (p' \supset p)$ are formulas.

(ii) Let Γ be a set of formulas and let A be a formula. Then A is derivable from Γ , notation $\Gamma \vdash_{\text{PROP}} A$, if $\Gamma \vdash A$ can be produced by the following formal system.

$\begin{aligned} A \in \Gamma &\Rightarrow \Gamma \vdash A \\ \Gamma \vdash A \supset B, \Gamma \vdash A &\Rightarrow \Gamma \vdash B \\ \Gamma, A \vdash B &\Rightarrow \Gamma \vdash A \supset B \end{aligned}$

NOTATION. (i) q, r, s, t, \dots stand for arbitrary propositional variables.

(ii) As usual $\Gamma \vdash A$ stands for $\Gamma \vdash_{PROP} A$ if there is little danger for confusion. Moreover, $\Gamma \vdash A$ stands for $\emptyset \vdash A$.

2.4.8. EXAMPLE. (i) $\vdash A \supset A$;

(ii) $B \vdash A \supset B$;

(iii) $\vdash A \supset (B \supset A)$;

(iv) $A \supset (A \supset B) \vdash A \supset B$.

2.4.9. DEFINITION. Let $A \in \text{form}(\text{PROP})$ and $\Gamma \subseteq \text{form}(\text{PROP})$.

(i) We define $[A] \in \mathbb{T}$ and $\Gamma_A \subseteq \mathbb{T}$ as follows.

A	$[A]$	Γ_A
p	α	\emptyset
$P \supset Q$	$[P] \rightarrow [Q]$	$\Gamma_P \cup \Gamma_Q$

It so happens that $\Gamma_A = \emptyset$ and $A \equiv [A]$ for all A . But the setup will be needed for more complex logics and type theories.

(ii) Moreover, we set $[\Gamma] = \{x_A : A \mid A \in \Gamma\}$.

2.4.10. PROPOSITION. Let $A \in \text{form}(\text{PROP})$ and $\Delta \subseteq \text{form}(\text{PROP})$. Then

$$\Delta \vdash_{\text{PROP}} A \Rightarrow [\Delta] \vdash_{\lambda \rightarrow} M : [A], \text{ for some } M.$$

PROOF. By induction on the generation of $\Delta \vdash A$.

Case 1. $\Delta \vdash A$ because $A \in \Delta$. Then $(x_A : [A]) \in [\Delta]$ and hence $[\Delta] \vdash x_A : [A]$. So we can take $M \equiv x_A$.

Case 2. $\Delta \vdash A$ because $\Delta \vdash B \supset A$ and $\Delta \vdash B$. Then by the induction hypothesis $[\Delta] \vdash P : [B] \rightarrow [A]$ and $[\Delta] \vdash Q : [B]$. Therefore, $[\Delta] \vdash PQ : [A]$.

Case 3. $\Delta \vdash A$ because $A \equiv B \supset C$ and $\Delta, B \vdash C$. By the induction hypothesis $[\Delta], x_B : [B] \vdash M : [C]$. Hence $[\Delta] \vdash (\lambda x_B. M) : [B] \rightarrow [C] \equiv [B \supset C] \equiv [A]$. ■

Conversely we have the following.

2.4.11. PROPOSITION. Let $\Delta, A \subseteq \text{form}(\text{PROP})$. Then

$$[\Delta] \vdash_{\lambda \rightarrow} M : [A] \Rightarrow \Delta \vdash_{\text{PROP}} A.$$

PROOF. By induction on the structure of M .

Case 1. $M \equiv x$. Then by the generation lemma 2.1.3 one has $(x : [A]) \in [\Delta]$ and hence $A \in \Delta$; so $\Delta \vdash_{\text{PROP}} A$.

Case 2. $M \equiv PQ$. By the generation lemma for some $C \in \Lambda$ one has $[\Delta] \vdash P : C \rightarrow [A]$ and $[\Delta] \vdash Q : C$. Clearly, for some $C' \in \text{form}$ one has $C \equiv [C']$. Then $C \rightarrow [A] \equiv [C' \supset A]$. By the induction hypothesis one has $\Delta \vdash C' \rightarrow A$ and $\Delta \vdash C'$. Therefore $\Delta \vdash A$.

Case 3. $M \equiv \lambda x. P$. Then $[\Delta] \vdash \lambda x. P : [A]$. By the generation lemma $[A] \equiv B \rightarrow C$ and $[\Delta], x : B \vdash P : C$, so that $[\Delta], x : [B'] \vdash P : [C']$, with $B' \equiv B, [C'] \equiv C$ (hence $[A] \equiv [B' \supset C']$). By the induction hypothesis it follows that $\Delta, B \vdash C$ and therefore $\Delta \vdash B \rightarrow C \equiv A$. ■

Although intuitionistic logic gives a complete characterization of those types that are inhabited, this does not answer immediately the question whether a type like $((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha$ corresponding to Peirce's law is inhabited.

Kripke models

Remember that a type $A \in \mathbb{T}(\lambda_{\rightarrow})$ is inhabited iff $[A]$, the translation of A into propositional calculus is intuitionistically provable. This explains why

$$A \text{ inhabited} \Rightarrow \models A,$$

but not conversely, since $\models A$ correspond to classical validity. A common tool to prove that types are not inhabited or that formulas are not intuitionistically derivable consist of so called *Kripke models* that we will introduce now.

2.4.12. DEFINITION. (i) A *Kripke model* consists of a tuple $\mathcal{K} = \langle K, \leq, \perp, F \rangle$, such that

- (1) $\langle K, \leq, \perp \rangle$ is a partially ordered set with least element \perp ;
- (2) $F : K \rightarrow \wp(\text{var})$ is a monotonic map from K to the powerset of the set of type-variables: $\forall k, k' \in K [k \leq k' \Rightarrow F(k) \subseteq F(k')]$.

We often just write $\mathcal{K} = \langle K, F \rangle$.

(ii) Let $\mathcal{K} = \langle K, F \rangle$ be a Kripke model. For $k \in K$ we define by induction on the structure of $A \in \mathbb{T}(\lambda_{\rightarrow})$ the notion k forces A , notation $k \Vdash_{\mathcal{K}} A$. We often omit the subscript.

$$\begin{aligned} k \Vdash \alpha &\iff \alpha \in F(k); \\ k \Vdash A_1 \rightarrow A_2 &\iff \forall k' \geq k [k' \Vdash A_1 \Rightarrow k' \Vdash A_2]. \end{aligned}$$

(iii) \mathcal{K} forces A , notation $\mathcal{K} \Vdash A$, is defined as $\perp \Vdash_{\mathcal{K}} A$.

(iv) If $\Gamma = \{x_1:A_1, \dots, x_n:A_n\}$, then we write $\mathcal{K} \Vdash \Gamma$, pronounce \mathcal{K} forces Γ , for $\mathcal{K} \Vdash A_1$ & ... & $\mathcal{K} \Vdash A_n$. Define $\Gamma \Vdash A$, pronounce ' Γ forces A ', iff for all Kripke models \mathcal{K} one has

$$\mathcal{K} \Vdash \Gamma \Rightarrow \mathcal{K} \Vdash A.$$

In particular $\Vdash A$, pronounce 'forced A ', if $\mathcal{K} \Vdash A$ for all Kripke models \mathcal{K} .

2.4.13. LEMMA. Let \mathcal{K} be a Kripke model. Then for all $A \in \mathbb{T}(\lambda_{\rightarrow})$ one has

$$k \leq k' \text{ \& } k \Vdash_{\mathcal{K}} A \Rightarrow k' \Vdash_{\mathcal{K}} A.$$

PROOF. By induction on the structure of A . ■

2.4.14. PROPOSITION. $\Gamma \vdash_{\lambda_{\rightarrow}} M : A \Rightarrow \Gamma \Vdash A$.

PROOF. By induction on the derivation of $M : A$ from Γ . If $M : A$ is $x : A$ and is in Γ , then this is trivial. If $\Gamma \vdash M : A$ is $\Gamma \vdash FP : A$ and is a direct consequence of $\Gamma \vdash F : B \rightarrow A$ and $\Gamma \vdash P : B$, then the conclusion follows from the induction hypothesis and the fact that $k \Vdash B \rightarrow A \ \& \ k \Vdash B \Rightarrow k \Vdash A$. In the case that $\Gamma \vdash M : A$ is $\Gamma \vdash \lambda x.N : A_1 \rightarrow A_2$ and follows directly from $\Gamma, x:A_1 \vdash N : A_2$ we have to do something. By the induction hypothesis we have for all \mathcal{K}

$$\mathcal{K} \Vdash \Gamma, A_1 \Rightarrow \mathcal{K} \Vdash A_2. \quad (2)$$

We must show $\Gamma \Vdash A_1 \rightarrow A_2$, i.e. $\mathcal{K} \Vdash \Gamma \Rightarrow \mathcal{K} \Vdash A_1 \rightarrow A_2$ for all \mathcal{K} .

Given \mathcal{K} and $k \in K$, define

$$\mathcal{K}_k = \langle \{k' \in K \mid k \leq k'\}, \leq, k, K \rangle,$$

(where \leq and F are in fact the appropriate restrictions to the subset $\{k' \in K \mid k \leq k'\}$ of K). Then it is easy to see that also \mathcal{K}_k is a Kripke model and

$$k \Vdash_{\mathcal{K}} A \iff \mathcal{K}_k \Vdash A. \quad (3)$$

Now suppose $\mathcal{K} \Vdash \Gamma$ in order to show $\mathcal{K} \Vdash A_1 \rightarrow A_2$, i.e. for all $k \in K$

$$k \Vdash_{\mathcal{K}} A_1 \Rightarrow k \Vdash_{\mathcal{K}} A_2.$$

Indeed,

$$\begin{aligned} k \Vdash_{\mathcal{K}} A_1 &\Rightarrow \mathcal{K}_k \Vdash A_1, && \text{by (3)} \\ &\Rightarrow \mathcal{K}_k \Vdash A_2, && \text{by (2), since by lemma 2.4.13 also } \mathcal{K}_k \Vdash \Gamma, \\ &\Rightarrow k \Vdash_{\mathcal{K}} A_2. \blacksquare \end{aligned}$$

2.4.15. COROLLARY. Let $A \in \mathbb{T}(\lambda_{\rightarrow})$. Then

$$A \text{ is inhabited} \Rightarrow \Vdash A.$$

PROOF. Take $\Gamma = \emptyset$. ■

Now it can be proved, see exercise 2.4.16, that (the type corresponding to) Peirce's law $P = ((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha$ is not forced in some Kripke model. Since $\nVdash P$ it follows that P is not inhabited, in spite of the fact that $\models P$.

2.4.16. EXERCISE. Show that Peirce's law $P = ((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha$ is not forced in the Kripke model $\mathcal{K} = \langle K, \leq, 0, F \rangle$ with $K = \{0, 1\}$, $0 \leq 1$ and $F(0) = \emptyset, F(1) = \{\alpha\}$.

We also have a converse to corollary 2.4.15 which theoretically answers the inhabitation question for λ_{\rightarrow} , namely the completeness theorem, Kripke [], (usually formulated for provability in intuitionistic logic):

$$A \text{ is inhabited} \iff \Vdash A.$$

This is done by constructing for a type that is not inhabited a Kripke 'counter-model' \mathcal{K} , i.e. $\mathcal{K} \nVdash A$. In [] it is shown that these Kripke countermodels can be taken to be finite. This solves the decision problem for inhabitation in λ_{\rightarrow} . In Statman [1979] the decision problem is shown to be PSPACE complete, so that further analysis of the complexity of the decision problem appears to be very difficult.

Set-theoretic models

Now we will prove using set-theoretic models that there do not exist terms satisfying certain properties.

2.4.17. DEFINITION. An $A \times A \rightarrow A$ pairing is a triple (P, L, R) such that in λ_{\rightarrow} one has

$$\begin{aligned} &\vdash P : A \rightarrow A \rightarrow A; \\ &\vdash L : A \rightarrow A \ \& \ \vdash R : A \rightarrow A; \\ &L(Pxy) =_{\beta\eta} x \ \& \ R(Pxy) =_{\beta\eta} y. \end{aligned}$$

It can be shown that there does not exist a $A \times A \rightarrow A$ pairing for arbitrary types A , see Barendregt [1974].

2.4.18. DEFINITION. Let X be a set. The *full typed structure* (for λ_{\rightarrow}^o types) over X , notation $\mathcal{M}_X = \{X(A)\}_{A \in \Pi_o}$, is defined as follows. For $A \in \Pi_o$ let $X(A)$ be defined inductively as follows.

$$\begin{aligned} X(o) &= X; \\ X(A \rightarrow B) &= X(B)^{X(A)}, \text{ the set of functions from } X(A) \text{ into } X(B). \end{aligned}$$

The reason that λ_{\rightarrow}^o -classic was introduced can be seen now from the way terms of that system can be interpreted easily into \mathcal{M}_X .

2.4.19. DEFINITION. (i) A *valuation* in \mathcal{M}_X is a map ρ from typed variables into $\cup_A X(A)$ such that $\rho(x^A) \in X(A)$ for all $A \in \Pi_o$.

(ii) Let ρ be a valuation in \mathcal{M}_X . The *interpretation under ρ* of a λ_{\rightarrow}^o -classic term into \mathcal{M}_X , notation $\llbracket M \rrbracket_{\rho}$, is defined as follows.

$$\begin{aligned} \llbracket x^A \rrbracket_{\rho} &= \rho(x^A); \\ \llbracket MN \rrbracket_{\rho} &= \llbracket M \rrbracket_{\rho} \llbracket N \rrbracket_{\rho}; \\ \llbracket \lambda x^A. M \rrbracket_{\rho} &= \lambda d \in X(A). \llbracket M \rrbracket_{\rho(x^A := d)}, \end{aligned}$$

where $\rho(x^A := d) = \rho'$ with $\rho'(x^A) = d$ and $\rho'(y^B) = \rho(y^B)$ if $y^B \neq x^A$.¹

(iii) Define

$$\mathcal{M}_X \models M = N \iff \forall \rho \llbracket M \rrbracket_{\rho} = \llbracket N \rrbracket_{\rho}.$$

Before proving properties about the models it is good to do exercises 2.5.8 and 2.5.9.

2.4.20. PROPOSITION. (i) $M \in \mathbf{term}_A \Rightarrow \llbracket M \rrbracket_{\rho} \in X(A)$.

(ii) $M =_{\beta\eta} N \Rightarrow \mathcal{M}_X \models M = N$.

¹Sometimes it is preferred to write $\llbracket \lambda x^A. M \rrbracket_{\rho}$ as $\lambda d \in X(A). \llbracket M[x^A := \underline{d}] \rrbracket_{\rho}$, where \underline{d} is a constant to be interpreted as d . Although this notation is perhaps more intuitive, we will not use it, since it also has technical drawbacks.

PROOF. (i) By induction on the structure of M .

(ii) By induction on the ‘proof’ of $M =_{\beta\eta} N$, using

$$\begin{aligned} \llbracket M[x := N] \rrbracket_\rho &= \llbracket M \rrbracket_{\rho(x := \llbracket N \rrbracket_\rho)}, \text{ for the } \beta\text{-rule;} \\ \rho \upharpoonright \text{FV}(M) = \rho' \upharpoonright \text{FV}(M) &\Rightarrow \llbracket M \rrbracket_\rho = \llbracket M \rrbracket_{\rho'}, \text{ for the } \eta\text{-rule;} \\ [\forall d \in X(A) \llbracket M \rrbracket_{\rho(x := d)} = \llbracket N \rrbracket_{\rho(x := d)}] &\Rightarrow \llbracket \lambda x^A. M \rrbracket_\rho = \llbracket \lambda x^A. N \rrbracket_\rho, \text{ for the} \end{aligned}$$

ξ -rule. ■

Now we will give applications of the notion of typed structure.

2.4.21. PROPOSITION. *Let A be a type in λ_{\rightarrow}^o . Then there does not exist an $A \times A \rightarrow A$ pairing.*

PROOF. Take $X = \{0, 1\}$. Then for every type A the set $X(A)$ is finite. Therefore by a cardinality argument there cannot be an $A \times A \rightarrow A$ pairing, for otherwise f defined by

$$f(x, y) = \llbracket P \rrbracket xy$$

would be an injection from $X(A) \times X(A)$ into $X(A)$, do exercise 2.5.9. ■

2.4.22. PROPOSITION. *There does not exist a term pred such that $\vdash_{\lambda_{\rightarrow}^o} \text{pred} : \text{Nat} \rightarrow \text{Nat}$ and*

$$\begin{aligned} \text{pred} \ulcorner 0 \urcorner &= \ulcorner 0 \urcorner; \\ \text{pred} \ulcorner n + 1 \urcorner &= \ulcorner n \urcorner. \end{aligned}$$

PROOF. As before for $X = \{0, 1\}$ the set $X(\text{Nat})$ is finite. Therefore

$$\mathcal{M}_X \models \ulcorner n \urcorner = \ulcorner m \urcorner$$

for some $n \neq m$. If pred did exist, then it follows easily that $\mathcal{M}_X \models \ulcorner 0 \urcorner = \ulcorner 1 \urcorner$. But this implies that $X(o)$ has cardinality 1, since $\ulcorner 0 \urcorner(Kx)y = y$ but $\ulcorner 1 \urcorner(Kx)y = Kxy = x$, a contradiction. ■

Another application of semantics is that there are no fixed-point operators in λ_{\rightarrow}^o .

2.4.23. DEFINITION. Let A be a type of λ_{\rightarrow}^o . A term Y is a *fixed-point operator of type A* iff $\vdash_{\tau} Y : (A \rightarrow A) \rightarrow A$ and

$$Y =_{\beta\eta} \lambda f : A \rightarrow A. f(Yf).$$

2.4.24. PROPOSITION. *For no type A there exists in λ_{\rightarrow}^o a fixed-point combinator.*

PROOF. Take $X = \{0, 1\}$. Then for every A the set $X(A)$ has at least two elements, say $x, y \in X(A)$ with $x \neq y$. Then there exists an $f \in X(A \rightarrow A)$ without a fixed-point:

$$\begin{aligned} f(z) &= x, & \text{if } z \neq x; \\ f(z) &= y, & \text{else.} \end{aligned}$$

Suppose a fixed-point combinator of type A did exist, then in \mathcal{M}_X one has

$$\llbracket Y \rrbracket f = f(\llbracket Y \rrbracket f),$$

contradicting that f has no fixed-point. ■

The results can easily be generalized to λ_{\rightarrow} , do exercise 2.5.10.

2.5. Exercises

2.5.1. Find out which of the following terms are typeable and determine for those that are the principal type.

$$\begin{aligned} &\lambda xyz.xz(yz); \\ &\lambda xyz.xy(xz); \\ &\lambda xyz.xy(zy). \end{aligned}$$

2.5.2. (i) Let $A = (\alpha \rightarrow \beta) \rightarrow ((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha$. Construct a term M such that $\vdash M : A$. What is the principal type of M ?

(ii) Find an expansion of M such that it has A as principal type.

2.5.3. (Uniqueness of Type Assignments) A \mathbf{K} -redex is an $R \equiv (\lambda x.M)N$ with $x \notin \text{FV}(M)$; if $x \in \text{FV}(M)$, then R is an \mathbf{l} -redex. In the following we consider a Church typing of an untyped term M and all of its subterms so that $M \in \Lambda_{\rightarrow \text{Ch}}(A)$.

(i) Use the principal type theorem to show that if more than one such typing exists then there is one which contains an atomic type not in A .

(ii) Show that if M only has lambda-l redexes, then every atomic type in such a typing must occur in A [Hint: Show this for $\beta\eta$ -nf and show that atomic types are preserved by $\beta\eta$ -reduction of $\lambda\mathbf{l}$ -terms].

(iii) Conclude that if a term $M \in \Lambda_{\rightarrow}^{\text{Cu}}(A)$ has only \mathbf{l} -redexes, then it has exactly one typing: if $M_1, M_2 \in \Lambda_{\rightarrow}^{\text{Ch}}(A)$ such that $|M_1| \equiv |M_2| \equiv M$, then $M_1 \equiv M_2$.

(iv) Give an example of a $\lambda\mathbf{K}$ -term $M \in \Lambda$ of type A with a non-unique Church typing of its subterms.

(v) Explain why an untyped term with only \mathbf{l} -redexes may have more than one type.

2.5.4. (i) Formulate and prove an appropriate version of the Church-Rosser theorem for $\lambda_{\rightarrow}^{\text{Ch}}$. [Hint. One possibility is to redo the proof of this result for untyped terms, as e.g. in B[1984]. An alternative is to use Proposition 1.4.9, Theorem 2.1.8 for $\lambda_{\rightarrow}^{\text{Cu}}$, the normalization theorem for λ_{\rightarrow} 2.1.19 and Exercise 2.5.3.

- (ii) Show that $\lambda_{\rightarrow}^{\text{dB}}$ satisfies the Church-Rosser Theorem. [Hint. Use (i) and translations between $\lambda_{\rightarrow}^{\text{dB}}$ and $\lambda_{\rightarrow}^{\text{Ch}}$.]

- 2.5.5. (Hindley) Show that if $\vdash_{\lambda_{\rightarrow}^{\text{Cu}}} M : A$, then there is a M' such that

$$M' \rightarrow_{\beta\eta} M \ \& \ \text{pt}(M') = A.$$

[Hints. 1. First make an η -expansion of M in order to obtain a term with a principal type having the same tree as A . 2. Show that for any type B with a subtype B_0 there exists a context $C[\]$ such that

$$z:B \vdash C[z] : B_0.$$

3. Use 1,2 and a term like $\lambda fz.z(fP)(fQ)$ to force identification of the types of P and Q . (For example one may want to identify α and γ in $(\alpha \rightarrow \beta) \rightarrow \gamma \rightarrow \delta$.)]

- 2.5.6. Prove that $\Lambda_{\rightarrow}^o(o) = \emptyset$ by applying the normalization and subject reduction theorems.

- 2.5.7. Let X be a set and consider the model \mathcal{M}_X of λ_{\rightarrow}^o . Notice that every permutation $\pi = pi_o$ (bijection) of X can be lifted to all levels $X(A)$ by defining

$$\pi_{A \rightarrow B}(f) = \pi_B \circ f \circ \pi_A^{-1}.$$

Prove that every lambda definable element $f \in X(A)$ in $\mathcal{M}(X)$ is invariant under all lifted permutations; i.e. $\pi_A(f) = f$. [Hint. Use the fundamental theorem for logical relations.]

- 2.5.8. (i) Show that $\mathcal{M}_X \models (\lambda x^A.x^A)y^A = y^A$.
(ii) Show that $\mathcal{M}_X \models (\lambda x^{A \rightarrow A}.x^{A \rightarrow A}) = (\lambda x^{A \rightarrow A} \lambda y^A.x^{A \rightarrow A}y^A)$.
(iii) Show that $\llbracket \ulcorner 2 \urcorner (Kx^o)y^o \rrbracket_{\rho} = \rho(x)$.

- 2.5.9. Let P, L, R be an $A \times B \rightarrow C$ pairing. Show that in every structure \mathcal{M}_X one has

$$\llbracket P \rrbracket xy = \llbracket P \rrbracket x'y' \Rightarrow x = x' \ \& \ y = y',$$

hence $\text{card}(A) \cdot \text{card}(B) \leq \text{card}(C)$.

- 2.5.10. Show that propositions 2.4.21, 2.4.22 and 2.4.24 can be generalized to λ_{\rightarrow} by modifying the notion of typed structure.

- 2.5.11. Prove that $\Lambda_{\rightarrow}^o(o) = \emptyset$ by applying models and the fact shown in the previous exercise that lambda definable elements are invariant under lifted permutations.

- 2.5.12. Let $\sim A \equiv A \rightarrow o$. Show that if o does not occur in A , then $\sim \sim (\sim \sim A \rightarrow A)$ is not inhabited. Why is the condition about o necessary?

- 2.5.13. We say that the structure of the rational numbers can be represented in λ_{\rightarrow} if there is a type $Q \in \mathbb{T}(\lambda_{\rightarrow})$ and closed lambda terms:

$$\begin{aligned} 0, 1 &: Q; \\ +, \cdot &: Q \rightarrow Q \rightarrow Q; \\ -, ^{-1} &: Q \rightarrow Q; \end{aligned}$$

such that $(Q, +, \cdot, -, ^{-1}, 0, 1)$ satisfies the axioms of a field of characteristic 0. Show that the rationals cannot be represented in λ_{\rightarrow} .

2.5.14. Show that there is no closed term

$$P : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}$$

such that P is a bijection in the sense that

$$\forall M : \text{Nat} \exists ! N_1, N_2 : \text{Nat} \ P N_1 N_2 =_{\beta\eta} M.$$

2.5.15. Show that every closed term of type $(0 \rightarrow 0 \rightarrow 0) \rightarrow 0 \rightarrow 0$ is $\beta\eta$ -convertible to $\lambda f^{0 \rightarrow 0 \rightarrow 0}. \lambda x^0. t$ with t given by the grammar

$$t := x \mid ftt.$$

The next two exercises show that the minimal length of a reduction-path of a term to normal form is non-elementary in the length of the term². See Péter [1967] for the definition of the class of (Kalmar) elementary functions. This class is the same as \mathcal{E}_3 in the Grzegorzcz hierarchy. To get some intuition for this class, define the family of functions $2_n : \mathbb{N} \rightarrow \mathbb{N}$ as follows.

$$\begin{aligned} 2_0(x) &= x; \\ 2_{n+1}(x) &= 2^{2_n(x)}. \end{aligned}$$

Then every elementary function f is eventually bounded by some 2_n :

$$\exists n, m \forall x > m \ f(x) \leq 2_n(x).$$

2.5.16. (i) Define the function $\mathbf{gk} : \mathbb{N} \rightarrow \mathbb{N}$ by

$$\begin{aligned} \mathbf{gk}(m) &= \#F_{\mathbf{gk}}(M), & \text{if } m = \#(M) \text{ for some untyped} \\ & & \text{lambda term } M; \\ &= 0, & \text{else.} \end{aligned}$$

Here $\#M$ denotes the Gödelnumber of the term M and $F_{\mathbf{gk}}$ is the Gross-Knuth reduction strategy defined by completely developing all present redexes in M , see B[1984]. Show that \mathbf{gk} is Kalmar elementary.

(ii) For a type $A \in \Pi_{\infty}$ define its *depth* $D(A) \in \mathbb{N}$ by

$$\begin{aligned} D(\alpha) &= 0; \\ D(A \rightarrow B) &= \max(D(A), D(B)) + 1. \end{aligned}$$

For a term $M \in \Lambda_{\rightarrow}^{\text{Ch}}$ define

$$D(M) = \max\{D(A \rightarrow B) \mid (\lambda x^A. P)^{A \rightarrow B} Q \text{ is a redex in } M\}$$

²In Gandy [1980a] this is also proved for arbitrary reduction paths starting from typable terms. In de Vrijer [1987] an exact calculation is given for the longest reduction paths to normal form.

Show that

$$F_{\text{GK}}(|M|) = |N| \Rightarrow D(M) > D(N).$$

[Hint. Use Lévy's analysis of redex creation, see Barendregt [1984], exercise 14.5.3.]

- (iii) If M is a term, then its *length*, notation $\text{length}(M)$, is the number of symbols in M . If a closed term $M \in \Lambda$ has a type, then its principal type $\text{pt}(M)$ has a depth bounded by its length:

$$D(\text{pt}(M)) \leq \text{length}(M).$$

See the proof of theorem 2.3.14.

- (iv) Write $\sigma: M \rightarrow M^{\text{nf}}$ if σ is some reduction path of M to normal form M^{nf} . Let $\$ \sigma$ be the number of reduction steps in σ . Define

$$\$(M) = \min\{\$ \sigma \mid \sigma : M \rightarrow M^{\text{nf}}\}.$$

Show that $\$(M) \leq g(\text{length}(M))$, for some function $g \in \mathcal{E}_4$. [Hint. Take $g(m) = gk^m(m)$.]

- 2.5.17. (i) Define $\mathbf{2}_1 = \lambda f^{o \rightarrow o} x^o. f(fx)$ and $\mathbf{2}_{n+1} = ([o := o \rightarrow o])\mathbf{2}_n \mathbf{2}$. Then the type of $\mathbf{2}_n$ is principal.
(ii) [Church] Show $([o := o \rightarrow o]\mathbf{c}_n)\mathbf{c}_m =_{\beta} \mathbf{c}_{m^n}$.
(iii) Show $\mathbf{2}_n =_{\beta} \mathbf{c}_{2^n(1)}$.
(iv) Let $M, N \in \Lambda$ be untyped terms. Show that if $M \twoheadrightarrow_{\beta} N$, then

$$\text{length}(M) \leq \text{length}(N)^2.$$

- (v) Conclude that the shortest length of a reduction path starting with a typed term M to normal form is in the worst case non-elementary in the length of M .

- 2.5.18. It will be shown that the length of the principal type of a typable term is at least exponential in the length of the term. This means that if $f(m) = \max\{\text{length}(\text{pt}(M)) \mid \text{length}(M) \leq m\}$, then for some real number $c_1 > 1$ one has $c_1^n \leq f(n)$, for sufficiently large n . It will be shown also that the depth of the principal type of a typable term M is linear in the length of M . It follows that the length of the principal type of M is also at most exponential in the length of M .

- (i) Define M_n to be the following term:

$$\lambda x_n \dots x_1. (x_n(x_n(x_{n-1}))(x_{n-1}(x_{n-1}x_{n-2}))) \dots (x_2(x_2x_1)).$$

Show that the principal type of M_n has length $> 2^n$. Conclude that the length of the principle type of a term is at least exponential in the length of the term.

- (ii) Show that there is a constant c_2 such that for typable lambda terms M one has for sufficiently long M

$$\text{depth}(\text{pt}(M)) \leq c_2(\text{length}(M)).$$

[Hint Use exercise 2.5.16(iii).]

- (iii) Conclude that the length of the principle type of a term is also at most exponential.

2.5.19. (Statman) In this exercise $\mathbb{T} = \mathbb{T}_o$. Let $\mathcal{M}_n = \mathcal{M}(\{0, 1, 2, \dots, n\})$. The purpose of this exercise is to show that this structure can be isomorphically embedded into $\mathcal{M}(\mathbb{N})$.

- (i) (Church's δ) We add to the language λ_{\rightarrow} constants $\underline{k} : o$ for $k \leq n$ and a constant $\underline{\delta} : o^4 \rightarrow o$. The intended interpretation of $\underline{\delta}$ is the function δ defined by

$$\begin{aligned} \delta xyuv &= u && \text{if } x = y; \\ &= v && \text{else.} \end{aligned}$$

We define the notion of reduction δ by the contraction rules

$$\begin{aligned} \underline{\delta} \underline{i} \underline{j} \underline{k} \underline{l} &\rightarrow_{\delta} \underline{k} && \text{if } i = j; \\ &\rightarrow_{\delta} \underline{l}, && \text{if } i \neq j. \end{aligned}$$

The resulting language of terms is called Λ_{δ} and on this we consider the notion of reduction $\rightarrow_{\beta\eta\delta}$.

- (ii) Show that every $M \in \Lambda_{\delta}$ satisfies $\text{SN}_{\beta\eta\delta}(M)$.
 (iii) Show that $\rightarrow_{\beta\eta\delta}$ is Church-Rosser.
 (iv) Let $M \in \Lambda_{\delta}^0(o)$ be a closed term of type o . Show that the normal form of M is one of the constants $\underline{0}, \dots, \underline{n}$.
 (v) (Church's theorem) Show that every element $\Phi \in \mathcal{M}_n$ can be defined by a closed term $M_{\Phi} \in \Lambda_{\delta}$. [Hint. For each $A \in \mathbb{T}$ define simultaneously the map $\Phi \mapsto M_{\Phi} : \mathcal{M}_n(A) \rightarrow \Lambda_{\delta}(A)$ and $\delta_A \in \Lambda_{\delta}(A \rightarrow A \rightarrow o \rightarrow o)$ such that for any closed M, N of type A one has

$$\begin{aligned} \delta_A MNuv &=_{\beta\eta\delta} u && \text{if } \llbracket M \rrbracket^{\mathcal{M}_n} = \llbracket N \rrbracket^{\mathcal{M}_n}; \\ &=_{\beta\eta\delta} v && \text{else.} \end{aligned}$$

For $A = o$ take $M_i = \underline{i}$ and $\delta_o = \delta$. For $A = B \rightarrow C$, let $\mathcal{M}_n(B) = \{\Phi_1, \dots, \Phi_t\}$ and let $B = B_1 \rightarrow \dots \rightarrow B_k \rightarrow o$. Now set

$$\begin{aligned} \delta_A &\equiv \lambda xyuv. \delta_B(x M_{\Phi_1}(y M_{\Phi_1})(\dots (\delta_A(x M_{\Phi_t}(y \Phi_t))uv) \dots))v. \\ M_{\Phi} &\equiv \lambda x \vec{y}. \\ &\quad \delta_B x M_{\Phi_1}(M_{\Phi_1} \vec{y}) \\ &\quad (\delta_B x M_{\Phi_2}(M_{\Phi_2} \vec{y}) \\ &\quad (\dots \\ &\quad (\delta_B x M_{\Phi_t}(M_{\Phi_t} \vec{y}) \underline{0}) \dots)). \end{aligned}$$

- (vi) Show that $\Phi \mapsto \llbracket M_\Phi \rrbracket^{\mathcal{M}(\mathbb{N})}$ is the required isomorphic embedding of \mathcal{M}_n into $\mathcal{M}(\mathbb{N})$.
- (vii) (To be used later.) Let $\pi_i^n \equiv (\lambda x_1 \dots x_n. x_i) : (o^n \rightarrow o)$. Define

$$\begin{aligned} \Delta_n &\equiv \lambda a b u v \vec{x}. a \\ &\quad (b(u\vec{x})(v\vec{x})(v\vec{x}) \dots (v\vec{x})) \\ &\quad (b(v\vec{x})(u\vec{x})(v\vec{x}) \dots (v\vec{x})) \\ &\quad \dots \\ &\quad (b(v\vec{x})(v\vec{x}) \dots (v\vec{x})(u\vec{x})). \end{aligned}$$

Then

$$\begin{aligned} \Delta^n \pi_i^n \pi_j^n \pi_k^n \pi_l^n &=_{\beta\eta\delta} \pi_k^n, & \text{if } i = j; \\ &=_{\beta\eta\delta} \pi_l^n, & \text{else.} \end{aligned}$$

Show that for $i \in \{1, \dots, n\}$ one has for all $M : o$

$$\begin{aligned} M &=_{\beta\eta\delta} \underline{i} \Rightarrow \\ M[o : o^n \rightarrow o][\delta : \Delta^n][\underline{1} : \pi_1^n] \dots [\underline{n} : \pi_n^n] &=_{\beta\eta} \pi_i^n. \end{aligned}$$

2.5.20. (Th. Joly)

- (i) Let $M = \langle Q, q_0, F, \delta \rangle$ be a deterministic finite automaton over the finite alphabet $\Sigma = \{a_1, \dots, a_n\}$. That is, Q is the finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta : \Sigma \times Q \rightarrow Q$ is the transition function. Let $L^r(M)$ be the (regular) language consisting of words in Σ^* accepted by M by reading the words from right to left. Let $\mathcal{M} = \mathcal{M}(Q)$ be the model of λ_{\rightarrow}^o over Q . Show that

$$w \in L^r(M) \iff \llbracket \underline{w} \rrbracket^{\mathcal{M}} \delta_{a_1} \dots \delta_{a_n} q_0 \in F,$$

where $\delta_a(q) = \delta(a, q)$ and \underline{w} is defined in 1.3.6.

- (ii) Similarly represent classes of trees (with at the nodes elements of Σ) accepted by a frontier-to-root tree automaton, see Thatcher [1973], by the model \mathcal{M} at the type $\top_n = (0^2 \rightarrow 0)^n \rightarrow 0 \rightarrow 0$.

Chapter 3

Tools

3.1. Typed lambda Models

In this chapter we work with λ_{\rightarrow}^o , having one atomic type o , rather than with λ_{\rightarrow} , having infinitely many atomic types. The reader is encouraged to investigate which results do generalize. We will write $\mathbb{T} = \mathbb{T}(\lambda_{\rightarrow}^o)$.

In this section we will treat models, consistent sets of equations and their term models. We develop the theory for λ_{\rightarrow}^o , the reader being invited to see how much can be generalized to λ_{\rightarrow} .

3.1.1. DEFINITION. Let $\mathcal{M} = \{\mathcal{M}(A)\}_{A \in \mathbb{T}_o}$ be a family of non-empty sets indexed by types.

(i) \mathcal{M} is called a *typed structure* for λ_{\rightarrow}^o if

$$\mathcal{M}(A \rightarrow B) \subseteq \mathcal{M}(B)^{\mathcal{M}(A)}.$$

(ii) Let \mathcal{M} be provided with application operators

$$(\mathcal{M}, \cdot) = (\{\mathcal{M}(A)\}_{A \in \mathbb{T}_o}, \{\cdot_{A,B}\}_{A,B \in \mathbb{T}_o})$$

with $\cdot_{A,B} : \mathcal{M}(A \rightarrow B) \times \mathcal{M}(A) \rightarrow \mathcal{M}(B)$. Such a structure we call an *applicative typed structure* if the following property of *extensionality* holds:

$$\forall f, g \in \mathcal{M}(A \rightarrow B) \quad [[\forall a \in \mathcal{M}(A) \quad f \cdot_{A,B} a = g \cdot_{A,B} a] \Rightarrow f = g].$$

(iii) \mathcal{M} is called *trivial* if $\mathcal{M}(o)$ is a singleton.

3.1.2. NOTATION. For applicative typed structures we use the infix notation $f \cdot_{A,B} x$ for $\cdot_{A,B}(f, x)$. Often we will be even more brief, extensionality becoming

$$\forall f, g \in \mathcal{M}(A \rightarrow B) \quad [[\forall a \in \mathcal{M}_A \quad fa = ga] \Rightarrow f = g]$$

or simply (if A, B are implicitly known)

$$\forall f, g \in \mathcal{M} \quad [[\forall a \quad fa = ga] \Rightarrow f = g].$$

3.1.3. PROPOSITION. *The notions of typed structure and applicative typed structure are equivalent.*

PROOF. In a typed structure \mathcal{M} define $f \cdot a = f(a)$; extensionality is obvious. Conversely, let $\langle \mathcal{M}, \cdot \rangle$ be an applicative typed structure. Define the typed structure \mathcal{M}' and $\Phi_A : \mathcal{M}(A) \rightarrow \mathcal{M}'(A)$ as follows.

$$\begin{aligned} \mathcal{M}'(o) &= \mathcal{M}(o); \\ \Phi_o(a) &= a; \\ \mathcal{M}'(A \rightarrow B) &= \{\Phi_{A \rightarrow B}(f) \in \mathcal{M}'(B)^{\mathcal{M}'(A)} \mid f \in \mathcal{M}(A \rightarrow B)\}; \\ \Phi_{A \rightarrow B}(f)(\Phi_A(a)) &= \Phi_B(f \cdot a). \end{aligned}$$

By definition Φ is surjective. By extensionality of the applicative typed structure it is also injective. Hence $\Phi_{A \rightarrow B}(f)$ is well defined. Clearly one has $\mathcal{M}'(A \rightarrow B) \subseteq \mathcal{M}'(B)^{\mathcal{M}'(A)}$. ■

3.1.4. DEFINITION. Let \mathcal{M}, \mathcal{N} be two applicative typed structures. F is a *morphism* iff $F = \{F_A\}_{A \in \mathbb{T}_o}$ such that for each $A, B \in \mathbb{T}_o$ one has

$$\begin{aligned} F_A : \mathcal{M}(A) &\rightarrow \mathcal{N}(A); \\ F_{A \rightarrow B}(f) \cdot F_A(a) &= F_B(f \cdot a). \end{aligned}$$

From now on we will not make a distinction between the notions of type and applicative typed structure.

3.1.5. PROPOSITION. *Let \mathcal{M} be a typed structure. Then*

$$\mathcal{M} \text{ is trivial} \iff \forall A \in \mathbb{T}_o. \mathcal{M}(A) \text{ is a singleton.}$$

PROOF. (\Leftarrow) By definition. (\Rightarrow) We will show this for $A = 1 = o \rightarrow o$. If $\mathcal{M}(o)$ is a singleton, then for all $f, g \in \mathcal{M}(1)$ one has $\forall x : \mathcal{M}(o).(fx) = (gx)$, hence $f = g$. Therefore $\mathcal{M}(o)$ is a singleton. ■

3.1.6. EXAMPLE. (i) The full typed structure $\mathcal{M}_X = \{X(A)\}_{A \in \mathbb{T}_o}$ over a non-empty set X , see definition 2.4.18, is an applicative typed structure.

(ii) Let (X, \leq) be a non-empty partially ordered set. Let $D(o) = X$ and $D(A \rightarrow B)$ consist of the monotone elements of $D(B)^{D(A)}$, where we order this set pointwise: for $f, g \in D(A \rightarrow B)$ define

$$f \leq g \iff \forall a \in D(A) fa \leq ga.$$

The elements of the applicative typed structure $D_X = \{D(A)\}_{A \in \mathbb{T}_o}$ are called the *hereditarily monotonic functions*, see the chapter by Howard in Troelstra [1973].

(iii) Let \mathcal{M} be an applicative typed structure. A *layered non-empty subfamily* of \mathcal{M} is a family $\Delta = \{\Delta(A)\}_{A \in \mathbb{T}_o}$ of sets, such that the following hold

$$\forall A \in \mathbb{T}_o. \emptyset \neq \Delta(A) \subseteq \mathcal{M}(A)$$

Δ is called *closed under application* if

$$f \in \Delta(A \rightarrow B), g \in \Delta(A) \Rightarrow fg \in \Delta(B).$$

Δ is called *extensional* if

$$\forall A, B \in \mathbb{T}_o \forall f, g \in \Delta(A \rightarrow B). [\forall a \in \Delta(A). fa = ga] \Rightarrow f = g.$$

If Δ satisfies all these conditions, then $\mathcal{M} \upharpoonright \Delta = (\Delta, \cdot \upharpoonright \Delta)$ is an applicative typed structure.

3.1.7. DEFINITION. (i) Let \mathcal{M} be an applicative type structure. Then a (*partial*) *valuation* in \mathcal{M} is a family of (partial) maps $\rho = \{\rho_A\}_{A \in \mathbb{T}_o}$ such that $\rho_A : \mathbf{Var}(A) \rightarrow \mathcal{M}(A)$.

(ii) Given an applicative typed structure \mathcal{M} and a partial valuation ρ in \mathcal{M} one can define for $M \in \Lambda_o(A)$ the partial semantics $\llbracket M \rrbracket_\rho^\mathcal{M} \in \mathcal{M}(A)$ as follows

$$\begin{aligned} \llbracket x^A \rrbracket_\rho^\mathcal{M} &= \rho_A(x); \\ \llbracket PQ \rrbracket_\rho^\mathcal{M} &= \llbracket P \rrbracket_\rho^\mathcal{M} \llbracket Q \rrbracket_\rho^\mathcal{M}; \\ \llbracket \lambda x. P \rrbracket_\rho^\mathcal{M} &= \lambda d. \llbracket P \rrbracket_{\rho[x:=d]}^\mathcal{M}. \end{aligned}$$

We often write $\llbracket M \rrbracket_\rho$ for $\llbracket M \rrbracket_\rho^\mathcal{M}$. The expression $\llbracket M \rrbracket_\rho$ may not always be defined, even if ρ is total. The problem arises with $\llbracket \lambda x. P \rrbracket_\rho$. Although the function $\lambda d \in A. \llbracket P \rrbracket_{\rho[x:=d]} \in \mathcal{M}(B)^{\mathcal{M}(A)}$ is uniquely determined by $\llbracket \lambda x. P \rrbracket_\rho d = \llbracket P \rrbracket_{\rho[x:=d]}$, it may fail to be an element of $\mathcal{M}(A \rightarrow B)$ which is only a subset of $\mathcal{M}(B)^{\mathcal{M}(A)}$. We write $\llbracket M \rrbracket_\rho \downarrow$ if $\llbracket M \rrbracket_\rho \in \mathcal{M}(A)$ is well defined. We will write $\llbracket M \rrbracket_\rho = \llbracket M \rrbracket_\rho^\mathcal{M}$ if there is little danger of confusion.

3.1.8. DEFINITION. (i) A typed structure \mathcal{M} is called a λ_o^- -*model* or simply a (*type*) *model* if for every partial valuation $\rho = \{\rho_A\}_A$ and every $A \in \mathbb{T}_o$ and $M \in \Lambda_o(A)$ such that $\mathbf{FV}(M) \subseteq \mathbf{dom}(\rho)$ one has $\llbracket M \rrbracket_\rho \downarrow$.

(ii) Let \mathcal{M} be a model and ρ a partial evaluation. Then we define

$$\mathcal{M}, \rho \models M = N \iff \llbracket M \rrbracket_\rho^\mathcal{M} = \llbracket N \rrbracket_\rho^\mathcal{M}.$$

Here and later we assume implicitly that M and N have the same type.

(iii) Let \mathcal{M} be a model. Then we say that \mathcal{M} satisfies $M = N$, notation

$$\mathcal{M} \models M = N$$

iff for all partial ρ with $\mathbf{FV}(MN) \subseteq \mathbf{dom}(\rho)$ one has $\mathcal{M}, \rho \models M = N$.

(iv) Let \mathcal{M} be a model. The *theory* of \mathcal{M} is

$$\mathbf{Th}(\mathcal{M}) = \{M = N \mid M, N \in \Lambda_o^\emptyset \text{ \& } \mathcal{M} \models M = N\}.$$

3.1.9. NOTATION. Let E_1, E_2 be partial (i.e. possibly undefined) expressions.

(i) Write $E_1 \rightrightarrows E_2$ iff $E_1 \downarrow \Rightarrow [E_2 \downarrow \text{ \& } E_1 = E_2]$.

(ii) Write $E_1 \simeq E_2$ iff $E_1 \rightrightarrows E_2 \text{ \& } E_2 \rightrightarrows E_1$.

3.1.10. LEMMA. (i) Let $M \in \Lambda_o(A)$ and N be a subterm of M . Then

$$\llbracket M \rrbracket_\rho \downarrow \Rightarrow \llbracket N \rrbracket_\rho \downarrow.$$

(ii) Let $M \in \Lambda_o(A)$. Then

$$\llbracket M \rrbracket_\rho \simeq \llbracket M \rrbracket_{\rho \upharpoonright \text{FV}(M)}.$$

(iii) Let $M \in \Lambda_o(A)$ and ρ_1, ρ_2 be such that $\rho_1 \upharpoonright \text{FV}(M) = \rho_2 \upharpoonright \text{FV}(M)$. Then

$$\llbracket M \rrbracket_{\rho_1} \simeq \llbracket M \rrbracket_{\rho_2}.$$

PROOF. (i) By induction on the structure of M .

(ii) Similarly.

(iii) By (ii). ■

3.1.11. LEMMA. Let \mathcal{M} be an applicative structure. Then

(i) For $M \in \Lambda_o(A)$, $x, N \in \Lambda_o(B)$ one has

$$\llbracket M[x:=N] \rrbracket_\rho^\mathcal{M} \simeq \llbracket M \rrbracket_{\rho[x:=\llbracket N \rrbracket_\rho^\mathcal{M}]}^\mathcal{M}.$$

(ii) For $M, N \in \Lambda_o(A)$ one has

$$M \twoheadrightarrow_{\beta\eta} N \Rightarrow \llbracket M \rrbracket_\rho^\mathcal{M} \rightrightarrows \llbracket N \rrbracket_\rho^\mathcal{M}.$$

PROOF. (i) By induction on the structure of M . Write $M^\bullet \equiv M[x := N]$. We only treat the case $M \equiv \lambda y.P$. By the variable convention we may assume that $y \notin \text{FV}(N)$. We have

$$\begin{aligned} \llbracket (\lambda y.P)^\bullet \rrbracket_\rho &\simeq \llbracket \lambda y.P^\bullet \rrbracket_\rho \\ &\simeq \lambda d. \llbracket P^\bullet \rrbracket_{\rho[y:=d]} \\ &\simeq \lambda d. \llbracket P \rrbracket_{\rho[y:=d][x:=\llbracket N \rrbracket_{\rho[y:=d]}]}, && \text{by the IH,} \\ &\simeq \lambda d. \llbracket P \rrbracket_{\rho[y:=d][x:=\llbracket N \rrbracket_\rho]}, && \text{by Lemma 3.1.10,} \\ &\simeq \lambda d. \llbracket P \rrbracket_{\rho[x:=\llbracket N \rrbracket_\rho][y:=d]} \\ &\simeq \llbracket \lambda y.P \rrbracket_{\rho[x:=\llbracket N \rrbracket_\rho]}. \end{aligned}$$

(ii) By induction on the generation of $M \twoheadrightarrow_{\beta\eta} N$.

Case $M \equiv (\lambda x.P)Q$ and $N \equiv P[x := Q]$. Then

$$\begin{aligned} \llbracket (\lambda x.P)Q \rrbracket_\rho &\rightrightarrows (\lambda d. \llbracket P \rrbracket_{\rho[x:=d]})(\llbracket Q \rrbracket_\rho) \\ &\rightrightarrows \llbracket P \rrbracket_{\rho[x:=\llbracket Q \rrbracket_\rho]} \\ &\simeq \llbracket P[x := Q] \rrbracket_\rho, && \text{by (i).} \end{aligned}$$

Case $M \equiv \lambda x.Nx$, with $x \notin \text{FV}(N)$. Then

$$\begin{aligned} \llbracket \lambda x.Nx \rrbracket_\rho &\rightrightarrows \lambda d. \llbracket N \rrbracket_\rho(d) \\ &\simeq \llbracket N \rrbracket_\rho. \end{aligned}$$

Cases $M \twoheadrightarrow_{\beta\eta} N$ is $PZ \twoheadrightarrow_{\beta\eta} QZ$, $ZP \twoheadrightarrow_{\beta\eta} ZQ$ or $\lambda x.P \twoheadrightarrow_{\beta\eta} \lambda x.Q$, and follows directly from $P \twoheadrightarrow_{\beta\eta} Q$, then the result follows from the IH.

The cases where $M \twoheadrightarrow_{\beta\eta} N$ follows via reflexivity or transitivity are easy to treat. ■

3.1.12. DEFINITION. Let \mathcal{M}, \mathcal{N} be typed lambda models and let $A \in \mathbb{T}_o$.

(i) \mathcal{M} and \mathcal{N} are *elementary equivalent at A*, notation $\mathcal{M} \equiv_A \mathcal{N}$, iff

$$\forall M, N \in \Lambda_o^\emptyset(A). [\mathcal{M} \models M = N \iff \mathcal{N} \models M = N].$$

(ii) \mathcal{M} and \mathcal{N} are *elementary equivalent*, notation $\mathcal{M} \equiv \mathcal{N}$, iff

$$\forall A \in \mathbb{T}_o. \mathcal{M} \equiv_A \mathcal{N}.$$

3.1.13. PROPOSITION. Let \mathcal{M} be typed lambda model. Then

$$\mathcal{M} \text{ is non-trivial} \iff \forall A \in \mathbb{T}_o. \mathcal{M}(A) \text{ is not a singleton.}$$

PROOF. (\Leftarrow) By definition. (\Rightarrow) We will show this for $A = 1 = o \rightarrow o$. Let c_1, c_2 be distinct elements of $\mathcal{M}(o)$. Consider $M \equiv \lambda x^o.y^o \in \Lambda_o^\emptyset(1)$. Let ρ_i be the partial valuation with $\rho_i(y^o) = c_i$. Then $\llbracket M \rrbracket_{\rho_i} \downarrow$ and $\llbracket M \rrbracket_{\rho_1} c_1 = c_1, \llbracket M \rrbracket_{\rho_2} c_1 = c_2$. Therefore $\llbracket M \rrbracket_{\rho_1}, \llbracket M \rrbracket_{\rho_2}$ are different elements of $\mathcal{M}(1)$. ■

3.1.14. PROPOSITION. Let \mathcal{M}, \mathcal{N} be models and $F: \mathcal{M} \rightarrow \mathcal{N}$ a surjective morphism. Then the following hold.

(i) $F(\llbracket M \rrbracket_\rho^\mathcal{M}) = \llbracket M \rrbracket_{F \circ \rho}^\mathcal{N}$, for all $M \in \Lambda_o(A)$.

(ii) $F(\llbracket M \rrbracket^\mathcal{M}) = \llbracket M \rrbracket^\mathcal{N}$, for all $M \in \Lambda_o^\emptyset(A)$.

PROOF. (i) By induction on the structure of M .

(ii) By (i). ■

3.1.15. PROPOSITION. Let \mathcal{M} be a typed lambda model.

(i) $\mathcal{M} \models (\lambda x.M)N = M[x := N]$.

(ii) $\mathcal{M} \models \lambda x.Mx = M$, if $x \notin \text{FV}(\mathcal{M})$.

PROOF. (i) $\begin{aligned} \llbracket (\lambda x.M)N \rrbracket_\rho &= \llbracket \lambda x.M \rrbracket_\rho \llbracket N \rrbracket_\rho \\ &= \llbracket M \rrbracket_{\rho[x := \llbracket N \rrbracket_\rho]}, & \text{by Lemma 3.1.11,} \\ &= \llbracket M[x := N] \rrbracket_\rho. \end{aligned}$

(ii) $\begin{aligned} \llbracket \lambda x.Mx \rrbracket_\rho d &= \llbracket Mx \rrbracket_{\rho[x := d]} \\ &= \llbracket M \rrbracket_{\rho[x := d]} d \\ &= \llbracket M \rrbracket_\rho d, & \text{as } x \notin \text{FV}(M). \end{aligned}$

Therefore by extensionality $\llbracket \lambda x.Mx \rrbracket_\rho = \llbracket M \rrbracket_\rho$. ■

3.1.16. LEMMA. *Let \mathcal{M} be a typed lambda model. Then*

$$\mathcal{M} \models M = N \iff \mathcal{M} \models \lambda x.M = \lambda x.N.$$

$$\begin{aligned} \text{PROOF. } \mathcal{M} \models M = N &\iff \forall \rho. \quad \llbracket M \rrbracket_\rho = \llbracket N \rrbracket_\rho \quad \blacksquare \\ &\iff \forall \rho, d. \quad \llbracket M \rrbracket_{\rho[x:=d]} = \llbracket N \rrbracket_{\rho[x:=d]} \\ &\iff \forall \rho, d. \quad \llbracket \lambda x.M \rrbracket_\rho d = \llbracket \lambda x.N \rrbracket_\rho d \\ &\iff \forall \rho. \quad \llbracket \lambda x.M \rrbracket_\rho = \llbracket \lambda x.N \rrbracket_\rho \\ &\iff \mathcal{M} \models \lambda x.M = \lambda x.N. \end{aligned}$$

3.1.17. PROPOSITION. (i) *For every non-empty set X the typed structure \mathcal{M}_X is a λ_{\rightarrow}^o -model.*

(ii) *Let X be a poset. Then \mathcal{D}_X is a λ_{\rightarrow}^o -model.*

(iii) *Let \mathcal{M} be an applicative typed structure. Assume that $\llbracket K_{A,B} \rrbracket^{\mathcal{M}} \downarrow$ and $\llbracket S_{A,B,C} \rrbracket^{\mathcal{M}} \downarrow$. Then \mathcal{M} is a λ_{\rightarrow}^o -model.*

(iv) *Let Δ be a layered non-empty subfamily of an applicative structure \mathcal{M} that is extensional and closed under application. Suppose $\llbracket K_{A,B} \rrbracket, \llbracket S_{A,B,C} \rrbracket$ are defined and in Δ . Then $\mathcal{M} \upharpoonright \Delta$, see Example 3.1.6(iii), is a λ_{\rightarrow}^o -model.*

PROOF. (i) Since \mathcal{M}_X is the full typed structure, $\llbracket M \rrbracket_\rho$ always exists.

(ii) By induction on M one can show that $\lambda d. \llbracket M \rrbracket_{\rho(x:=d)}$ is monotonic. It then follows by induction on M that $\llbracket M \rrbracket_\rho \in \mathcal{D}_X$.

(iii) For every λ -term M there exists a typed applicative expression P consisting only of K s and S s such that $P \rightarrow_{\beta\eta} M$. Now apply Lemma 3.1.11.

(iv) By (iii). \blacksquare

Operations on models

Now we will introduce two operations on models: $\mathcal{M}, \mathcal{N} \mapsto \mathcal{M} \times \mathcal{N}$, the cartesian product, and $\mathcal{M} \mapsto \mathcal{M}^*$, the polynomial model. The relationship between \mathcal{M} and \mathcal{M}^* is similar to that of a ring R and its ring of multivariate polynomials $R[\vec{x}]$.

Cartesian products

3.1.18. DEFINITION. If \mathcal{M}, \mathcal{N} are applicative structures, then $\mathcal{M} \times \mathcal{N}$ is the structure defined by

$$\begin{aligned} (\mathcal{M} \times \mathcal{N})(A) &= \mathcal{M}(A) \times \mathcal{N}(A) \\ (M_1, M_2)(N_1, N_2) &= (M_1 N_1, M_2 N_2). \end{aligned}$$

3.1.19. PROPOSITION. *Let \mathcal{M}, \mathcal{N} be models. For a partial valuation ρ in $\mathcal{M} \times \mathcal{N}$ write $\rho(x) = (\rho_1(x), \rho_2(x))$.*

- (i) $\llbracket M \rrbracket_\rho^{\mathcal{M} \times \mathcal{N}} = (\llbracket M \rrbracket_{\rho_1}^{\mathcal{M}}, \llbracket M \rrbracket_{\rho_2}^{\mathcal{N}})$.
- (ii) $\mathcal{M} \times \mathcal{N}$ is a model.

(iii) $\text{Th}(\mathcal{M} \times \mathcal{N}) = \text{Th}(\mathcal{M}) \cap \text{Th}(\mathcal{N})$.

PROOF. (i) By induction on M .

(ii) By (i).

$$\begin{aligned}
 \text{(iii)} \quad \mathcal{M} \times \mathcal{N}, \rho \models M = N &\iff \llbracket M \rrbracket_\rho = \llbracket N \rrbracket_\rho \\
 &\iff (\llbracket M \rrbracket_{\rho_1}^{\mathcal{M}}, \llbracket M \rrbracket_{\rho_2}^{\mathcal{N}}) = (\llbracket N \rrbracket_{\rho_1}^{\mathcal{M}}, \llbracket N \rrbracket_{\rho_2}^{\mathcal{N}}) \\
 &\iff \llbracket M \rrbracket_{\rho_1}^{\mathcal{M}} = \llbracket N \rrbracket_{\rho_1}^{\mathcal{M}} \ \& \ \llbracket M \rrbracket_{\rho_2}^{\mathcal{M}} = \llbracket N \rrbracket_{\rho_2}^{\mathcal{M}} \\
 &\iff \mathcal{M}, \rho \models M = N \ \& \ \mathcal{N}, \rho \models M = N,
 \end{aligned}$$

by (i). Hence for closed terms M, N

$$\mathcal{M} \times \mathcal{N} \models M = N \iff \mathcal{M} \models M = N \ \& \ \mathcal{N} \models M = N. \blacksquare$$

Polynomial models

3.1.20. DEFINITION. (i) We introduce a new constant c_m for each $m \in \mathcal{M}$ and we let $\underline{\mathcal{M}}$ be the set of all typed applicative combinations of variables and constants c_m .

(ii) For each valuation $\rho : \text{Var} \rightarrow \mathcal{M}$ define the interpretation $((-))_\rho : \underline{\mathcal{M}} \rightarrow \mathcal{M}$ by

$$\begin{aligned}
 ((x))_\rho &= \rho(x); \\
 ((c_m))_\rho &= m; \\
 ((PQ))_\rho &= ((P))_\rho ((Q))_\rho.
 \end{aligned}$$

(iii) Write

$$P \sim_{\mathcal{M}} Q \iff \forall \rho ((P))_\rho = ((Q))_\rho,$$

where ρ ranges over valuations in \mathcal{M} .

3.1.21. LEMMA. (i) $\sim_{\mathcal{M}}$ is an equivalence relation satisfying $c_d \cdot c_e \sim_{\mathcal{M}} c_{de}$.

(ii) For all $P, Q \in \underline{\mathcal{M}}$ one has

$$P_1 \sim_{\mathcal{M}} P_2 \iff \forall Q_1, Q_2 \in \underline{\mathcal{M}} [Q_1 \sim_{\mathcal{M}} Q_2 \Rightarrow P_1 Q_1 \sim_{\mathcal{M}} P_2 Q_2].$$

PROOF. Easy. \blacksquare

3.1.22. DEFINITION. Let \mathcal{M} be an applicative structure. The *polynomial structure* over \mathcal{M} is $\mathcal{M}^* = (|\mathcal{M}^*|, \text{app})$ defined by

$$\begin{aligned}
 |\mathcal{M}^*| &= \underline{\mathcal{M}} / \sim_{\mathcal{M}} = \{[P]_{\sim_{\mathcal{M}}} \mid P \in \underline{\mathcal{M}}\}, \\
 \text{app } [P]_{\sim_{\mathcal{M}}} [Q]_{\sim_{\mathcal{M}}} &= [PQ]_{\sim_{\mathcal{M}}}.
 \end{aligned}$$

By lemma 3.1.21(ii) this is well defined.

3.1.23. PROPOSITION. (i) $\mathcal{M} \subseteq \mathcal{M}^*$ by the embedding morphism $d \mapsto c_d$.

(ii) $\mathcal{M}^* \cong \mathcal{M}^{**}$.

PROOF. (i) Define $i(d^A) = [c_d^A]$. Then $i : \mathcal{M}(A) \rightarrow \mathcal{M}^*(A)$ and

$$\begin{aligned} i(d \cdot_{\mathcal{M}} e) &= [c_d \cdot_{\mathcal{M}} e] \\ &= [c_d \cdot_{\underline{\mathcal{M}}} c_e], && \text{by definition of } \sim_{\mathcal{M}}, \\ &= [c_d] \cdot_{\mathcal{M}^*} [c_e], && \text{by definition of } \mathbf{app} \\ &= i(d) \cdot_{\mathcal{M}^*} i(e). \end{aligned}$$

We will not always be so explicit about where application is taken.

(ii) Adding twice an infinite set of variables is the same as adding one infinite set, since $2 \cdot \aleph_0 = \aleph_0$. ■

Working with \mathcal{M}^* it is often convenient to use as elements those of $\underline{\mathcal{M}}$ and reason about them modulo $\sim_{\mathcal{M}}$.

3.1.24. DEFINITION. Let $P \in \underline{\mathcal{M}}$ and let x be a variable. We say that

$$P \text{ does not depend on } x$$

if whenever ρ_1, ρ_2 satisfy $\rho_1(y) = \rho_2(y)$ for $y \neq x$, we have $\llbracket P \rrbracket_{\rho_1} = \llbracket P \rrbracket_{\rho_2}$.

3.1.25. LEMMA. If P does not depend on x , then $P \sim_{\mathcal{M}} P[x := Q]$ for all $Q \in \underline{\mathcal{M}}$.

PROOF. First show that $((P[x := Q]))_{\rho} = \llbracket P \rrbracket_{\rho[x := (Q)_{\rho}]}$, in analogy to Lemma 3.1.11(i). Now suppose P does not depend on x . Then

$$\begin{aligned} ((P[x := Q]))_{\rho} &= \llbracket P \rrbracket_{\rho[x := (Q)_{\rho}]} \\ &= ((P))_{\rho}, && \text{as } P \text{ does not depend on } x. \blacksquare \end{aligned}$$

3.1.26. PROPOSITION. Let \mathcal{M} be an applicative structure. Then

- (i) \mathcal{M} is a model \iff for each $P \in \mathcal{M}^*$ and variable x there exists an $F \in \mathcal{M}^*$ not depending on x such that $Fx = P$.
- (ii) \mathcal{M} is a model $\Rightarrow \mathcal{M}^*$ is a model.

PROOF. (i) It suffices to show that

$$\mathcal{M} \text{ is a model} \iff \text{for each } P \in \underline{\mathcal{M}} \text{ and variable } x \text{ there exists an } F \in \underline{\mathcal{M}} \text{ not depending on } x \text{ such that } Fx \sim_{\mathcal{M}} P.$$

(\Rightarrow) Let \mathcal{M} be a model and let P be given. We treat an illustrative example, e.g. $P \equiv c_f x^o y^o$, with $f \in \mathcal{M}(1_2)$. We take $F \equiv c_{[\lambda y z_f. z_f x y]} y c_f$. Then

$$((Fx))_{\rho} = \llbracket \lambda y z_f. z_f x y \rrbracket_{\rho}(y) f \rho(x) = f \rho(x) \rho(y) = ((c_f x y))_{\rho},$$

hence indeed $Fx \sim_{\mathcal{M}} c_f x y$. In general for each constant c_d in P we take a variable z_d and define $F \equiv \llbracket \lambda \vec{y} \vec{z}_d x. P \rrbracket \vec{y} \vec{c}_f$.

(\Leftarrow) We show by induction on M that $\forall M \in \Lambda_o \exists P_M \in \underline{\mathcal{M}} \forall \rho. \llbracket M \rrbracket_{\rho} = ((P_M))_{\rho}$. For M being a variable, constant or application this is trivial. For $M = \lambda x. N$, we know by the

induction hypothesis that $\llbracket N \rrbracket_\rho = ((P_N))_\rho$ for all ρ . By assumption there is an F not depending on x such that $Fx \sim_{\mathcal{M}} P_N$. Then

$$((F))_\rho d = ((Fx))_{\rho[x:=d]} = ((P_N))_{\rho[x:=d]} \stackrel{\text{IH}}{=} \llbracket N \rrbracket_{\rho[x:=d]}.$$

Hence $\llbracket \lambda x.N \rrbracket_{\rho} \downarrow = ((F))_\rho$. It follows that \mathcal{M} is a model.

(ii) By (i) \mathcal{M}^* is a model if a certain property holds for \mathcal{M}^{**} . But $\mathcal{M}^{**} \cong \mathcal{M}^*$ and the property does hold here, since \mathcal{M} is a model. [To make matters concrete, one has to show for example that for all $M \in \mathcal{M}^{**}$ there is an N not depending on y such that $Ny \sim_{\mathcal{M}^*} M$. Writing $M \equiv M[x_1, x_2][y]$ one can obtain N by rewriting the y in M obtaining $M' \equiv M[x_1, x_2][x] \in \mathcal{M}^*$ and using the fact that \mathcal{M} is a model: $M' = Nx$, so $Ny = M$]. ■

3.1.27. PROPOSITION. *If \mathcal{M} is a model, then $\text{Th}(\mathcal{M}^*) = \text{Th}(\mathcal{M})$.*

PROOF. Do exercise 3.6.2. ■

3.1.28. REMARK. In general for typed structures $\mathcal{M}^* \times \mathcal{N}^* \not\cong (\mathcal{M} \times \mathcal{N})^*$, but the isomorphism holds in case \mathcal{M}, \mathcal{N} are typed lambda models.

3.2. Lambda Theories and Term Models

3.2.1. DEFINITION. (i) A *constant* (of type A) is a variable (of the same type) that we promise not to bind by a λ . Rather than x, y, z, \dots we write constants as c, d, e, \dots . The letters $\mathcal{C}, \mathcal{D}, \dots$ range over sets of constants (of varying types).

(ii) Let $\mathcal{D} = \{c_1^{A_1}, \dots, c_n^{A_n}\}$ be a set of constants with types in \mathbb{T}_o . Write $\Lambda_o[\mathcal{D}](A)$ for the set of open terms of type A , possibly containing the constants in \mathcal{D} . Moreover $\Lambda_o[\mathcal{D}] = \cup_{A \in \mathbb{T}} \Lambda_o[\mathcal{D}](A)$.

(iii) Similarly $\Lambda_o^\emptyset[\mathcal{D}](A)$ and $\Lambda_o^\emptyset[\mathcal{D}]$ consist of closed terms possibly containing the constants in \mathcal{D} .

(iv) An equation over \mathcal{D} is of the form $M = N$ with $M, N \in \Lambda_o^\emptyset[\mathcal{D}]$ of the same type.

In this subsection we will consider sets of equations over \mathcal{D} . When writing $M = N$, we implicitly assume that M, N have the same type.

3.2.2. DEFINITION. Let \mathcal{E} be a set of equations over the constants. \mathcal{D} .

(i) $P = Q$ is derivable from \mathcal{E} , notation $\mathcal{E} \vdash P = Q$ if $P = Q$ can be proved in the

equational theory axiomatized as follows

$(\lambda x.M)N = M[x := N]$	(β)
$(\lambda x.Mx = M), x \notin \text{FV}(M)$	(η)
$\{M = N \mid (M = N) \in \mathcal{E}\}$	
$M = M$	(reflexivity)
$M = N$	(symmetry)
$N = M$	
$M = N \quad N = L$	(transitivity)
$M = L$	
$M = N$	(R-congruence)
$MZ = NZ$	
$M = N$	(L-congruence)
$ZM = ZN$	
$M = N$	(ξ -rule)
$\lambda x.M = \lambda x.N$	

We write $M =_{\mathcal{E}} N$ for $\mathcal{E} \vdash M = N$.

- (ii) \mathcal{E} is *consistent*, if not all equations are derivable from it.
- (iii) \mathcal{E} is a *typed lambda theory* iff \mathcal{E} is consistent and closed under derivability.

3.2.3. NOTATION. (i) $\mathcal{E}^+ = \{M = N \mid \mathcal{E} \vdash M = N\}$.

- (ii) For $A \in \mathbb{T}_o$ write $\mathcal{E}(A) = \{M = N \mid (M = N) \in \mathcal{E} \text{ \& } M, N \text{ are of type } A\}$.

- (iii) $\mathcal{E}_{\beta\eta} = \emptyset^+$.

3.2.4. PROPOSITION. *If $Mx =_{\mathcal{E}} Nx$, with $x \notin \text{FV}(M) \cup \text{FV}(N)$, then $M =_{\mathcal{E}} N$.*

PROOF. Use (ξ) and (η). ■

3.2.5. DEFINITION. Let \mathcal{M} be a type model and \mathcal{E} a set of equations.

- (i) We say that \mathcal{M} *satisfies* (or is a *model* of) \mathcal{E} , notation $\mathcal{M} \models \mathcal{E}$, iff

$$\forall (M=N) \in \mathcal{E}. \mathcal{M} \models M = N.$$

- (ii) We say that \mathcal{E} *satisfies* $M = N$, notation $\mathcal{E} \models M = N$, iff

$$\forall \mathcal{M}. [\mathcal{M} \models \mathcal{E} \Rightarrow \mathcal{M} \models M = N].$$

3.2.6. PROPOSITION. (*Soundness*) $\mathcal{E} \vdash M = N \Rightarrow \mathcal{E} \models M = N$.

PROOF. By induction on the derivation of $\mathcal{E} \vdash M = N$. Assume that $\mathcal{M} \models \mathcal{E}$ for a model \mathcal{M} towards $\mathcal{M} \models M = N$. If $M = N \in \mathcal{E}$, then the conclusion follows from the assumption. The cases that $M = N$ falls under the axioms β or η follow from Proposition 3.1.15. The rules reflexivity, symmetry, transitivity and L,R-congruence are trivial to treat. The case falling under the ξ -rule follows from Lemma 3.1.16. ■

From non-trivial models one can obtain typed lambda theories.

3.2.7. PROPOSITION. *Let \mathcal{M} be a non-trivial model.*

- (i) $\mathcal{M} \models \mathcal{E} \Rightarrow \mathcal{E}$ is consistent.
- (ii) $\text{Th}(\mathcal{M})$ is a lambda theory.

PROOF. (i) Suppose $\mathcal{E} \vdash \lambda xy.x = \lambda xy.y$. Then $\mathcal{M} \models \lambda xy.x = \lambda xy.y$. It follows that $d = (\lambda xy.x)de = (\lambda xy.y)de$ for arbitrary d, e . Hence \mathcal{M} is trivial.

(ii) Clearly $\mathcal{M} \models \text{Th}(\mathcal{M})$. Hence by (i) $\text{Th}(\mathcal{M})$ is consistent. If $\text{Th}(\mathcal{M}) \vdash M = N$, then by soundness $\mathcal{M} \models M = N$, and therefore $(M = N) \in \text{Th}(\mathcal{M})$. ■

The full type model over a finite set yields an interesting λ -theory.

3.2.8. EXERCISE. Let $\mathcal{M}_n = \mathcal{M}_{\{1, \dots, n\}}$. Write $c_i = \lambda fx.f^i x$ for $i \in \mathbb{N}$, the Church numerals of type $1 \rightarrow o \rightarrow o$.

- (i) Show that for $i, j \in \mathbb{N}$ one has

$$\mathcal{M}_n \models c_i = c_j \iff i = j \vee [i, j \geq n-1 \ \& \ \forall k_{1 \leq k \leq n}. i \equiv j \pmod{k}].$$

[Hint. For $a \in \mathcal{M}_n(o)$, $f \in \mathcal{M}_n(1)$ define the *trace of a under f* as $\{f^i(a) \mid i \in \mathbb{N}\}$, directed by $G_f = \{(a, b) \mid f(a) = b\}$, which by the pigeonhole principle is ‘lasso-shaped’. Consider the traces of 1 under the functions f_n, g_m with $1 \leq m \leq n$

$$\begin{aligned} f_n(k) &= k+1, & \text{if } k < n, & \text{ and } g_m(k) = k+1, & \text{if } k < m, & \text{ Conclude that e.g. } \mathcal{M}_5 \models \\ &= n, & \text{if } k = n, & & = 1, & \text{if } k = m, \\ & & & & = k, & \text{else.} \end{aligned}$$

$$c_4 = c_{64}, \mathcal{M}_6 \not\models c_4 = c_{64} \text{ and } \mathcal{M}_6 \models c_5 = c_{65}.$$

- (ii) Conclude that $\mathcal{M}_n \equiv_{1 \rightarrow o \rightarrow o} \mathcal{M}_m \iff n = m$.

- (iii) Show that $\bigcap_n \text{Th}(\mathcal{M}_n)(1) = \mathcal{E}_{\beta\eta}(1)$.

It will be shown in the section 3.4 that $\bigcap_n \text{Th}(\mathcal{M}_n) = \mathcal{E}_{\beta\eta}$ and by contrast $\text{Th}(\mathcal{M}_{\mathbb{N}}) = \mathcal{E}_{\beta\eta}$.

Term Models

3.2.9. DEFINITION. Let \mathcal{D} be a set of constants and let \mathcal{E} be a set of closed equations between closed terms with constants from \mathcal{D} . Define the applicative structure $\mathcal{M}_{\mathcal{E}}$ by

$$\mathcal{M}_{\mathcal{E}}(A) = \{[M]_{\mathcal{E}} \mid M \in \Lambda_o[\mathcal{D}](A)\},$$

where $[M]_{\mathcal{E}}$ is the equivalence class modulo the congruence relation $=_{\mathcal{E}}$, with

$$[M]_{\mathcal{E}}[N]_{\mathcal{E}} = [MN]_{\mathcal{E}}$$

as application operator. This is well-defined, because $=_{\mathcal{E}}$ is a congruence.

3.2.10. PROPOSITION. (i) $\mathcal{M}_{\mathcal{E}}$ is an applicative type structure.

(ii) The semantic interpretation of M in $\mathcal{M}_{\mathcal{E}}$ is determined by

$$\llbracket M \rrbracket_{\rho} = [M[\vec{x} := \vec{N}]]_{\mathcal{E}},$$

where the \vec{N} are determined by $\rho(x_i) = [N_i]_{\mathcal{E}}$.

(iii) $\mathcal{M}_{\mathcal{E}}$ is a type model, called the open term model.

PROOF. (i) We need to verify extensionality.

$$\begin{aligned} \forall d \in \mathcal{M}_{\mathcal{E}}. [M]d = [N]d &\Rightarrow [M][x] = [N][x], && \text{for a fresh } x, \\ &\Rightarrow [Mx] = [Nx] \\ &\Rightarrow Mx =_{\mathcal{E}} Nx \\ &\Rightarrow M =_{\mathcal{E}} N \\ &\Rightarrow [M] = [N]. \end{aligned}$$

(ii) We show that $\llbracket M \rrbracket_{\rho}$ satisfies the conditions in definition 3.1.8(ii).

$$\begin{aligned} \llbracket x \rrbracket_{\rho} &= [x[x := N]]_{\mathcal{E}}, && \text{with } \rho(x) = [N]_{\mathcal{E}}, \\ &= [N]_{\mathcal{E}} \\ &= \rho(x); \\ \llbracket PQ \rrbracket_{\rho} &= [(PQ)[\vec{x} := \vec{N}]]_{\mathcal{E}} \\ &= [P[\vec{x} := \vec{N}]Q[\vec{x} := \vec{N}]]_{\mathcal{E}} \\ &= [P[\vec{x} := \vec{N}]]_{\mathcal{E}}[[Q[\vec{x} := \vec{N}]]_{\mathcal{E}}] \\ &= \llbracket P \rrbracket_{\rho} \llbracket Q \rrbracket_{\rho}; \\ \llbracket \lambda y. P \rrbracket_{\rho} \llbracket Q \rrbracket_{\mathcal{E}} &= [(\lambda y. P)[\vec{x} := \vec{N}]]_{\mathcal{E}}[Q]_{\mathcal{E}} \\ &= [\lambda y. P[\vec{x} := \vec{N}]]_{\mathcal{E}}[Q]_{\mathcal{E}} \\ &= [P[\vec{x} := \vec{N}][y := Q]]_{\mathcal{E}} \\ &= [P[\vec{x}, y := \vec{N}, Q]]_{\mathcal{E}}, && \text{because } y \notin \text{FV}(\vec{N}) \text{ by the} \\ &&& \text{variable convention,} \\ &= \llbracket P \rrbracket_{\rho(y := [Q]_{\mathcal{E}})} \\ &= \llbracket P \rrbracket_{\rho(y := [Q]_{\mathcal{E}})}. \end{aligned}$$

(iii) As $\llbracket M \rrbracket_{\rho}$ is always defined by (ii). ■

3.2.11. COROLLARY. (i) $\mathcal{M}_{\mathcal{E}} \models M = N \iff M =_{\mathcal{E}} N$.

(ii) $\mathcal{M}_{\mathcal{E}} \models \mathcal{E}$.

PROOF. (i) (\Rightarrow) Suppose $\mathcal{M}_{\mathcal{E}} \models M = N$. Then $\llbracket M \rrbracket_{\rho} = \llbracket N \rrbracket_{\rho}$ for all ρ . Choosing $\rho(x) = [x]_{\mathcal{E}}$ one obtains $\llbracket M \rrbracket_{\rho} = [M[\vec{x} := \vec{x}]]_{\mathcal{E}} = [M]_{\mathcal{E}}$, and similarly for N , hence $[M]_{\mathcal{E}} = [N]_{\mathcal{E}}$ and therefore $M =_{\mathcal{E}} N$.

- $$\begin{aligned}
(\Leftarrow) \quad M =_{\mathcal{E}} N &\Rightarrow M[\vec{x} := \vec{P}] =_{\mathcal{E}} N[\vec{x} := \vec{P}] \\
&\Rightarrow [M[\vec{x} := \vec{P}]]_{\mathcal{E}} = [N[\vec{x} := \vec{P}]]_{\mathcal{E}} \\
&\Rightarrow \llbracket M \rrbracket_{\rho} = \llbracket N \rrbracket_{\rho} \\
&\Rightarrow \mathcal{M}_{\mathcal{E}} \models M = N.
\end{aligned}$$
- (ii) If $M = N \in \mathcal{E}$, then $M =_{\mathcal{E}} N$, hence $\mathcal{M} \models M = N$, by (i). ■

Using this Corollary we obtain completeness in a simple way.

3.2.12. THEOREM (Completeness). $\mathcal{E} \vdash M = N \iff \mathcal{E} \models M = N$.

PROOF. (\Rightarrow) By soundness, Proposition 3.2.6.

- $$\begin{aligned}
(\Leftarrow) \quad \mathcal{E} \models M = N &\Rightarrow \mathcal{M}_{\mathcal{E}} \models M = N, \quad \text{as } \mathcal{M}_{\mathcal{E}} \models \mathcal{E}, \\
&\Rightarrow M =_{\mathcal{E}} N \\
&\Rightarrow \mathcal{E} \vdash M = N. \quad \blacksquare
\end{aligned}$$

3.2.13. COROLLARY. Let \mathcal{E} be a set of equations. Then

$$\mathcal{E} \text{ has a non-trivial model} \iff \mathcal{E} \text{ is consistent.}$$

PROOF. (\Rightarrow) By Proposition 3.2.7. (\Leftarrow) Suppose that $\mathcal{E} \not\models x^o = y^o$. Then by the Theorem one has $\mathcal{E} \not\models x^o = y^o$. Then for some model \mathcal{M} one has $\mathcal{M} \models \mathcal{E}$ and $\mathcal{M} \not\models x = y$. It follows that \mathcal{M} is non-trivial. ■

If \mathcal{D} contains enough constants, then one can similarly define the applicative structure $\mathcal{M}_{\mathcal{E}[\mathcal{D}]}$ by restricting $\mathcal{M}_{\mathcal{E}}$ to closed terms. See section 3.3.

Constructing Theories

The following result is due to Jacopini [1975].

3.2.14. PROPOSITION. Let \mathcal{E} be a set of equations between closed terms in $\Lambda_o^{\emptyset}[\mathcal{D}]$. Then $\mathcal{E} \vdash M = N$ iff if for some $n \in \mathbb{N}$ and terms F_1, \dots, F_n and equations $P_1 = Q_1, \dots, P_n = Q_n \in \mathcal{E}$ one has

$$\left. \begin{array}{lcl}
M & =_{\beta\eta} & F_1 P_1 Q_1 \\
F_1 Q_1 P_1 & =_{\beta\eta} & F_2 P_2 Q_2 \\
& \dots & \\
F_{n-1} Q_{n-1} P_{n-1} & =_{\beta\eta} & F_n P_n Q_n \\
F_n Q_n P_n & =_{\beta\eta} & N.
\end{array} \right\} \quad (1)$$

This scheme (1) is called a Jacopini tableau and the sequence F_1, \dots, F_n is called the list of witnesses.

PROOF. (\Leftarrow) Obvious, since clearly $\mathcal{E} \vdash FPQ = FQP$ if $P = Q \in \mathcal{E}$.

(\Rightarrow) By induction on the derivation of $M = N$ from the axioms. If $M = N$ is a $\beta\eta$ -axiom or the axiom of reflexivity, then we can take as witnesses the empty list. If $M = N$

is an axiom in \mathcal{E} , then we can take the singleton list K . If $M = N$ follows from $M = L$ and $L = N$, then we can concatenate the lists that exist by the induction hypothesis. If $M = N$ is $PZ = QZ$ (respectively $ZP = ZQ$) and follows from $P = Q$ with list F_1, \dots, F_n , then the list for $M = N$ is F_1', \dots, F_n' with $F_i' \equiv \lambda ab.F_i abZ$ (respectively $F_i' \equiv \lambda ab.Z(F_i ab)$). If $M = N$ follows from $N = M$, then we have to reverse the list. If $M = N$ is $\lambda x.P = \lambda x.Q$ and follows from $P = Q$ with list F_1, \dots, F_n , then the new list is F_1', \dots, F_n' with $F_i' \equiv \lambda pqx.F_i pq$. Here we use that the equations in \mathcal{E} are between closed terms. ■

Remember that $\text{true} \equiv \lambda xy.x$, $\text{false} \equiv \lambda xy.y$ both having type $1_2 = o \rightarrow o \rightarrow o$.

3.2.15. LEMMA. *Let \mathcal{E} be a set of equations over \mathcal{D} . Then*

$$\mathcal{E} \text{ is consistent} \iff \mathcal{E} \not\vdash \text{true} = \text{false}.$$

PROOF. (\Leftarrow) By definition. (\Rightarrow) Suppose $\mathcal{E} \vdash \lambda xy.x = \lambda xy.y$. Then $\mathcal{E} \vdash P = Q$ for arbitrary $P, Q \in \Lambda_o(o)$. But then for arbitrary terms M, N of the same type $A = A_1 \rightarrow \dots \rightarrow A_n \rightarrow o$ one has $\mathcal{E} \vdash M\vec{z} = N\vec{z}$ for fresh $\vec{z} = z_1, \dots, z_n$ of the right type, hence $\mathcal{E} \vdash M = N$, by Proposition 3.2.4. ■

3.2.16. DEFINITION. Let $M, N \in \Lambda_o^\emptyset[\mathcal{D}](A)$ be closed terms of type A .

(i) M is *inconsistent with* N , notation $M \# N$, if

$$\{M = N\} \vdash \text{true} = \text{false}.$$

(ii) M is *separable from* N , notation $M \perp N$, iff for some $F \in \Lambda_o^\emptyset[\mathcal{D}](A \rightarrow 1_2)$

$$FM = \text{true} \ \& \ FN = \text{false}.$$

The following result is not true for the untyped lambda calculus: the equation $K = YK$ is inconsistent, but K and YK are not separable, as follows from the genericity lemma, see Barendregt [1984].

3.2.17. PROPOSITION. *Let $M, N \in \Lambda_o^\emptyset[\mathcal{D}](A)$ be closed terms of type A . Then*

$$M \# N \iff M \perp N.$$

PROOF. (\Leftarrow) Trivially separability implies inconsistency.

(\Rightarrow) Suppose $\{M = N\} \vdash \text{true} = \text{false}$. Then also $\{M = N\} \vdash x = y$. Hence by proposition 3.2.14 one has

$$\begin{aligned} x &=_{\beta\eta} F_1 MN \\ F_1 NM &=_{\beta\eta} F_2 MN \\ &\dots \\ F_n NM &=_{\beta\eta} y. \end{aligned}$$

Let n be minimal for which this is possible. We can assume that $\text{FV}(F_i) \subseteq \{x, y\}$. The normal form of F_1NM must be either x or y . Hence by the minimality of n it must be y , otherwise there is a shorter list of witnesses. Now consider F_1MM . Its normal form must be either x or y .

Case 1: $F_1MM =_{\beta\eta} x$. Then set $F \equiv \lambda axy.F_1aM$ and we have $FM =_{\beta\eta} \text{true}$ and $FN =_{\beta\eta} \text{false}$.

Case 2: $F_1MM =_{\beta\eta} y$. Then set $F \equiv \lambda axy.F_1Ma$ and we have $FM =_{\beta\eta} \text{false}$ and $FN =_{\beta\eta} \text{true}$. ■

3.2.18. COROLLARY. *Let \mathcal{E} be a set of equations over \mathcal{D} . If \mathcal{E} is inconsistent, then for some equation $M=N \in \mathcal{E}$ the terms M and N are separable.*

PROOF. By the same reasoning. ■

3.2.19. THEOREM. *Let*

$$\mathcal{E}_{\max} = \{M=N \mid M, N \in \Lambda_o^\emptyset[\mathcal{D}] \text{ and } M, N \text{ are not separable}\}.$$

Then this is the unique maximally consistent set of equations.

PROOF. By the corollary this set is consistent. By Proposition 3.2.17 it contains all consistent equations. Therefore the set is maximally consistent. Moreover it is the unique such set. ■

It will be shown in Chapter 4 that \mathcal{E}_{\max} is decidable.

3.3. Syntactic and Semantic logical relations

In this section we will introduce the well-known method of *logical relations* in two ways: one on the terms and one on elements of a model. Applications of the method will be given and it will be shown how the two methods are related.

Syntactic Logical Relations

3.3.1. DEFINITION. Let n be a fixed natural number and let $\vec{\mathcal{D}} = \mathcal{D}_1, \dots, \mathcal{D}_n$ be sets of constants.

(i) R is called an (n -ary) *family of relations* (or sometimes just a *relation*) on $\Lambda_o[\vec{\mathcal{D}}]$, if $R = \{R_A\}_{A \in \mathbb{T}}$ and for $A \in \mathbb{T}$

$$R_A \subseteq \Lambda_o[\mathcal{D}_1](A) \times \dots \times \Lambda_o[\mathcal{D}_n](A).$$

If we want to make the sets of constants explicit, we say that R is a relation *on terms from $\mathcal{D}_1, \dots, \mathcal{D}_n$* .

(ii) Such an R is called a *logical relation* if for all $A, B \in \mathbb{T}$ and for all $M_1 \in \Lambda_o[\mathcal{D}_1](A \rightarrow B), \dots, M_n \in \Lambda_o[\mathcal{D}_n](A)$ one has

$$R_{A \rightarrow B}(M_1, \dots, M_n) \iff \forall N_1 \in \Lambda_o[\mathcal{D}_1](A) \dots N_n \in \Lambda_o[\mathcal{D}_n](A) \\ [R_A(N_1, \dots, N_n) \Rightarrow R_B(M_1 N_1, \dots, M_n N_n)].$$

(iii) R is called empty if $R(o) = \emptyset$.

Obviously, a logical family $\{R_A\}$ is completely determined by R_o ; the higher R_A do depend on the choice of the \mathcal{D}_i .

3.3.2. LEMMA. *If R is a non-empty logical relation, then $\forall A \in \mathbb{T}_o. R_A \neq \emptyset$.*

PROOF. (For R unary.) By induction on A . Case $A = o$. By assumption. Case $A = B \rightarrow C$. Then $R_{B \rightarrow C}(M) \iff \forall P \in \Lambda_o(B). [R_B(P) \Rightarrow R_C(MP)]$. By the induction hypothesis one has $R_C(N)$. Then $M \equiv \lambda p. N \in \Lambda_o(B \rightarrow C)$ is in R_A . ■

Even the empty logical relation is interesting.

3.3.3. PROPOSITION. *Let $n = 1, \mathcal{D}_1 = \emptyset$ and R let be the logical relation determined by $R_o = \emptyset$. Then*

$$R_A = \Lambda_o(A) \quad \text{if } \Lambda_o^\emptyset(A) \neq \emptyset; \\ = \emptyset \quad \text{else.}$$

PROOF. By induction on A . If $A = o$, then we are done, as $R_o = \emptyset$ and $\Lambda_o^\emptyset(o) = \emptyset$. If $A = A_1, \dots, A_n \rightarrow o$, then

$$R_A(M) \iff \forall P_i \in R_{A_i}. R_o(M \vec{P}) \\ \iff \forall P_i \in R_{A_i}. \perp,$$

seeing R both as a relation and as a set. This last statement either is always the case, namely iff

$$\exists i. R_{A_i} = \emptyset \iff \exists i. \Lambda_o^\emptyset(A_i) = \emptyset \quad \text{by the induction hypothesis,} \\ \iff \Lambda_o^\emptyset(A) \neq \emptyset, \quad \text{by proposition 2.4.4.}$$

Or else, namely iff $\Lambda_o^\emptyset(A) = \emptyset$, it is never the case, by the same reasoning. ■

3.3.4. EXAMPLE. Let $n = 2$ and set $R_o(M, N) \iff M =_{\beta\eta} N$. Let R be the logical relation determined by R_o . The it is easily seen that for all A and $M, N \in \Lambda_o[\mathcal{D}](A)$ one has $R_A(M, N) \iff M =_{\beta\eta} N$.

3.3.5. DEFINITION. (i) Let M, N be lambda terms. Then M is a *weak head expansion* of N , notation $M \rightarrow_{wh} N$, if $M \equiv (\lambda x. P)Q\vec{R}$ and $N \equiv P[x = Q]\vec{R}$.

(ii) A family R on $\Lambda_o[\mathcal{D}]$ is called *expansive* if R_o is closed under coordinatewise weak head expansion, i.e. if $M_i' \rightarrow_{wh} M_i$ for $1 \leq i \leq n$, then

$$R_o(M_1, \dots, M_n) \Rightarrow R_o(M_1', \dots, M_n').$$

3.3.6. LEMMA. *If R is logical and expansive, then each R_A is closed under coordinatewise weak head expansion.*

PROOF. Immediate by induction on the type A and the fact that

$$M' \rightarrow_{wh} M \Rightarrow M'N \rightarrow_{wh} MN. \blacksquare$$

3.3.7. EXAMPLE. (i) Let M be a term. We say that $\beta\eta$ confluence holds from M , notation $\downarrow_{\beta\eta} M$, if whenever $N_1 \beta\eta \leftarrow M \rightarrow_{\beta\eta} N_2$, then there exists a term L such that $N_1 \rightarrow_{\beta\eta} L \beta\eta \leftarrow N_2$. Define R_o by

$$R_o(M) \iff \beta\eta \text{ confluence holds from } M.$$

Then R_o determines a logical R which is expansive by the permutability of head contractions with internal ones.

(ii) Let R be the logical relation generated from

$$R_o(M) \iff \downarrow_{\beta\eta} M.$$

Then for arbitrary type $A \in \mathbb{T}$ one has

$$R_A(M) \Rightarrow \downarrow_{\beta\eta} M.$$

[Hint. Write $M \downarrow_{\beta\eta} N$ if $\exists Z [M \rightarrow_{\beta\eta} Z \beta\eta \leftarrow N]$. First show that for arbitrary variable of some type B one has $R_B(x)$. Show also that if x is fresh, then by distinguishing cases whether x gets eaten or not

$$N_1 x \downarrow_{\beta\eta} N_2 x \Rightarrow N_1 \downarrow_{\beta\eta} N_2.$$

Then use induction on A .]

3.3.8. DEFINITION. Let R be n -ary and $*_1, \dots, *_n$ be substitutors satisfying $*_i : \mathbf{Var}(A) \rightarrow \Lambda_o[\mathcal{D}_i](A)$.

- (i) Write $R(*_1, \dots, *_n)$ if $R_A(x^{*_1}, \dots, x^{*_n})$ for each variable x of type A .
- (ii) Define R^* by

$$R_A^*(M_1, \dots, M_n) \iff \forall *_1 \dots *_n [R(*_1, \dots, *_n) \Rightarrow R_A(M_1^{*_1}, \dots, M_n^{*_1})].$$

- (iii) R is called *substitutive* if $R = R^*$.

3.3.9. LEMMA. (i) Let R be logical and $M_1 \in \Lambda_o^\emptyset[\mathcal{D}_1], \dots, M_n \in \Lambda_o^\emptyset[\mathcal{D}_n]$ be arbitrary closed terms. Then one has

$$R(M_1, \dots, M_n) \iff R^*(M_1, \dots, M_n).$$

(ii) For a substitutive R one has for arbitrary open $M_1, \dots, M_n, N_1, \dots, N_n$

$$R(M_1, \dots, M_n) \& R(N_1, \dots, N_n) \Rightarrow R(M_1[x:=N_1], \dots, M_n[x:=N_n]).$$

PROOF. (i) Clearly one has $R(M_1, \dots, M_n) \Rightarrow R^*(M_1, \dots, M_n)$. For the converse, note that $R^*(M_1, \dots, M_n) \Rightarrow R(M_1, \dots, M_n)$ in case $R_o \neq \emptyset$. But for $R_o = \emptyset$ the result follows from example 3.3.4.

(ii) Since R is substitutive we have $R^*(M_1, \dots, M_n)$. Let $*_i = [x:=N_i]$. Then $R(*_1, \dots, *_n)$ and hence $R(M_1[x:=N_1], \dots, M_n[x:=N_n])$. ■

3.3.10. EXERCISE. Let R be the logical relation generated by $R_o(M)$ iff $\downarrow_{\beta\eta} M$. Show by induction on M that $R^*(M)$ for all M . [Hint. Use that R is expansive.] Conclude that for closed M one has $R(M)$ and hence $\downarrow_{\beta\eta} M$. The same holds for arbitrary open terms N : let $\{\vec{x}\} = \text{FV}(M)$, then

$$\begin{aligned} \lambda\vec{x}.N \text{ is closed} &\Rightarrow R(\lambda\vec{x}.N) \\ &\Rightarrow R((\lambda\vec{x}.N)\vec{x}), && \text{since } R(x_i), \\ &\Rightarrow R(N), && \text{since } R \text{ is closed under } \rightarrow_{\beta}, \\ &\Rightarrow \downarrow_{\beta\eta} N. \end{aligned}$$

Thus the Church-Rosser property holds for $\rightarrow_{\beta\eta}$.

3.3.11. PROPOSITION. Let R be an arbitrary n -ary family on $\Lambda_o[\mathcal{D}]$. Then

- (i) $R^*(x, \dots, x)$ for all variables.
- (ii) If R is logical, then so is R^* .
- (iii) If R is expansive, then so is R^* .
- (iv) $R^{**} = R^*$, so R^* is substitutive.
- (v) If R is logical and expansive, then

$$R^*(M_1, \dots, M_n) \Rightarrow R^*(\lambda x.M_1, \dots, \lambda x.M_n).$$

PROOF. For notational simplicity we assume $n = 1$.

- (i) If $R^*(x)$, then by definition $R(x)$. Therefore $R^*(x)$.
- (ii) We have to prove

$$R^*(M) \iff \forall N \in \Lambda_o[\mathcal{D}] [R^*(M) \Rightarrow R^*(MN)].$$

(\Rightarrow) Assume $R^*(M) \& R^*(N)$ in order to show $R^*(MN)$. Let $*$ be a substitutor such that $R(*)$. Then

$$\begin{aligned} R^*(M) \& R^*(N) &\Rightarrow R(M^*) \& R(N^*) \\ &\Rightarrow R(M^*N^*) \equiv R((MN)^*) \\ &\Rightarrow R^*(MN). \end{aligned}$$

(\Leftarrow) By the assumption and (i) we have

$$R^*(Mx), \tag{1}$$

where we choose x to be fresh. In order to prove $R^*(M)$ we have to show $R(M^*)$, whenever $R(*)$. In order to do this it suffices to assume $R(N)$ and show $R(M^*N)$. Choose $*' = *(x:=N)$, then also $R(*')$. Hence by (1) and the freshness of x we have $R((Mx)^{*'}) \equiv R(M^*N)$ and we are done.

(iii) First observe that weak head reductions permute with substitution:

$$((\lambda x.P)Q\vec{R})^* \equiv (P[x:=Q]\vec{R})^*.$$

Now let $M \rightarrow_{wh} M^w$ be a weak head reduction step. Then

$$\begin{aligned} R^*(M^w) &\Rightarrow R(M^{w*}) \equiv R(M^{**w}) \\ &\Rightarrow R(M^*) \\ &\Rightarrow R^*(M). \end{aligned}$$

(iv) For substitutors $*_1, *_2$ write $*_1*_2$ for $*_2 \circ *_1$. This is convenient since

$$M^{*1*_2} \equiv M^{*2 \circ *1} \equiv (M^{*1})^{*2}.$$

Assume $R^{**}(M)$. Let $*_1(x) = x$ for all x . Then $R^*(*_1)$, by (i), and hence $R^*(M^{*1}) \equiv R^*(M)$. Conversely, assume $R^*(M)$, i.e.

$$\forall * [R(*) \Rightarrow R(M^*)], \tag{2}$$

in order to show $\forall *_1 [R^*(*_1) \Rightarrow R(M^{*1})]$. Now

$$\begin{aligned} R^*(*_1) &\iff \forall *_2 [R(*_2) \Rightarrow R(*_1*_2)], \\ R^*(M^{*1}) &\iff \forall *_2 [R(*_2) \Rightarrow R(M^{*1*_2})]. \end{aligned}$$

Therefore by (2) applied to $*_1*_2$ we are done.

(v) Let R be logical and expansive. Assume $R^*(M)$. Then

$$\begin{aligned} R^*(N) &\Rightarrow R^*(M[x:=N]), && \text{since } R^* \text{ is substitutive,} \\ &\Rightarrow R^*((\lambda x.M)N), && \text{since } R^* \text{ is expansive.} \end{aligned}$$

Therefore $R^*(\lambda x.M)$ since R^* is logical. ■

3.3.12. THEOREM (Fundamental theorem for syntactic logical relations). *Let R be logical, expansive and substitutive. Then for all $A \in \mathbb{T}$ and all pure terms $M \in \Lambda_o(A)$ one has*

$$R_A(M, \dots, M).$$

PROOF. By induction on M we show that $R_A(M, \dots, M)$.

Case $M \equiv x$. Then the statement follows from the assumption $R = R^*$ (substitutivity) and proposition 3.3.11 (i).

Case $M \equiv PQ$. By the induction hypothesis and the assumption that R is logical.

Case $M \equiv \lambda x.P$. By the induction hypothesis and proposition 3.3.11 (v). ■

3.3.13. COROLLARY. *Let R be an n -ary expansive logical relation. Then for all closed $M \in \Lambda_o^\emptyset[\mathcal{D}]$ one has $R(M, \dots, M)$.*

PROOF. By the theorem applied to R^* and lemma 3.3.9. ■

The proof in exercise 3.3.10 was in fact an application of this corollary.

3.3.14. EXAMPLE. Let R be the logical relation determined by

$$R_o(M) \iff M \text{ is normalizable.}$$

Then R is expansive. Note that if $R_A(M)$, then M is normalizable. [Hint. Use $R_B(x)$ for arbitrary B and x and the fact that if $M\vec{x}$ is normalizable, then so is M .] It follows from corollary 3.3.13 that each closed term is normalizable. By taking closures it follows that all terms are normalizable. This is the proof of weak normalization in Prawitz [1965]. For strong normalization a similar proof brakes down. The corresponding R is not expansive.

3.3.15. EXAMPLE. A family of relations $S_A \subseteq \Lambda_o^\emptyset[\mathcal{D}_1](A) \times \dots \times \Lambda_o^\emptyset[\mathcal{D}_n](A)$ which satisfies

$$\begin{aligned} S_{A \rightarrow B}(M_1, \dots, M_n) &\iff \forall N_1 \in \Lambda_o^\emptyset[\mathcal{D}_1](A) \dots N_n \in \Lambda_o^\emptyset[\mathcal{D}_n](A) \\ &\quad [S_A(N_1, \dots, N_n) \Rightarrow S_B(M_1 N_1, \dots, M_n N_n)] \end{aligned}$$

can be lifted to a substitutive logical relation S^* on $\Lambda_o[\mathcal{D}_1] \times \dots \times \Lambda_o[\mathcal{D}_n]$ as follows. Define for substitutors $*_i : \mathbf{Var}(A) \rightarrow \Lambda_o^\emptyset[\mathcal{D}_i](A)$

$$S_A(*_1, \dots, *_n) \iff \forall x:A \ S_A(x^{*_1}, \dots, x^{*_n}).$$

Now define S^* as follows: for $M_i \in \Lambda_o[\mathcal{D}_i](A)$

$$S_A^*(M_1, \dots, M_n) \iff \forall *_1 \dots *_n [S_A(*_1, \dots, *_n) \Rightarrow S_A(M_1^{*_1}, \dots, M_n^{*_n})].$$

Show that if S is closed under coordinatewise weak head expansions, then S^* is expansive.

The following definition is needed on order to relate the syntactic and the semantic notions of logical relation.

3.3.16. DEFINITION. Let R be an $n + 1$ -ary family. The *projection* of R , notation $\exists R$, is the n -ary family defined by

$$\exists R(M_1, \dots, M_n) \iff \exists M_{n+1} \in \Lambda_o[\mathcal{D}_{n+1}] R(M_1, \dots, M_{n+1}).$$

3.3.17. PROPOSITION. (i) *The universal n -ary relation is defined by*

$$R_A = \Lambda_o[\mathcal{D}_1](A) \times \dots \times \Lambda_o[\mathcal{D}_n](A).$$

This relation is logical, expansive and substitutive.

(ii) *Let $R \subseteq \mathcal{M}_1 \times \dots \times \mathcal{M}_n$ and $S \subseteq \mathcal{N}_1 \times \dots \times \mathcal{N}_m$ be non-empty logical relations. Then $R \times S \subseteq \mathcal{M}_1, \dots, \mathcal{M}_n, \mathcal{N}_1, \dots, \mathcal{N}_m$ is a non-empty logical relation. If moreover both R and S are substitutive, then so is $R \times S$.*

(iii) *If R is an n -ary family and π is a permutation of $\{1, \dots, n\}$, then R^π defined by*

$$R^\pi(M_1, \dots, M_n) \iff R(M_{\pi(1)}, \dots, M_{\pi(n)})$$

is logical if R is logical, is expansive if R is expansive and is substitutive if R is substitutive.

(iv) *Let R be an n -ary substitutive logical relation on terms from $\mathcal{D}_1, \dots, \mathcal{D}_n$ and let $\mathcal{D} \subseteq \cap_i \mathcal{D}_i$. Then the diagonal of R , notation R^Δ , defined by*

$$R^\Delta(M) \iff R(M, \dots, M)$$

is a substitutive logical (unary) relation on terms from \mathcal{D} , which is expansive if R is expansive.

(v) *If \mathcal{R} is a class of n -ary substitutive logical relations, then $\cap \mathcal{R}$ is an n -ary substitutive logical relation, which is expansive if each member of \mathcal{R} is expansive.*

(vi) *If R is an n -ary substitutive, expansive and logical relation, then $\exists R$ is a substitutive, expansive and logical relation.*

PROOF. (i) Trivial.

(ii) Suppose that R, S are logical. We show for $n = m = 1$ that $R \times S$ is logical.

$$\begin{aligned} (R \times S)_{A \rightarrow B}(M, N) &\iff R_{A \rightarrow B}(M) \& S_{A \rightarrow B}(N) \\ &\iff [\forall P. R_A(P) \Rightarrow R_B(MP)] \& \\ &\quad [\forall Q. R_A(Q) \Rightarrow R_B(NQ)] \\ &\iff \forall (P, Q). (R \times S)_A(P, Q) \Rightarrow (R \times S)_B(MP, NQ). \end{aligned}$$

For the last (\Leftarrow) one needs that the R, S are non-empty, and Lemma 3.3.2. If both R, S are expansive, then trivially so is $R \times S$.

(iii) Trivial.

(iv) We have

$$\begin{aligned} R^\Delta(M) &\iff R(M, M) \\ &\iff \forall N_1, N_2. R(N_1, N_2) \Rightarrow R(MN_1, MN_2) \end{aligned}$$

and must show that the chain of equivalences continuous

$$\iff \forall N. R(N, N) \Rightarrow R(MN, MN). \quad (1)$$

Now (\Rightarrow) is trivial. As to (\Leftarrow) , suppose (1) and $R(N_1, N_2)$, in order to show $R(MN_1, MN_2)$. By Lemma 3.3.11(i) one has $R(x, x)$. Hence $R(Mx, Mx)$ by (1). Therefore $R^*(Mx, Mx)$, as R is substitutive. Now taking $*_i = [x := N_i]$, one obtains $R(MN_1, MN_2)$.

(v) Trivial.

(vi) Like in (iv) it suffices to show that

$$\forall P. [\exists R(P) \Rightarrow \exists R(MP)] \quad (2)$$

implies $\exists N \forall P, Q. [R(P, Q) \Rightarrow R(MP, NQ)]$. Again we have $R(x, x)$. Therefore $\exists N_1. R^*(Mx, N_1)$ by (2). Writing $N \equiv \lambda x. N_1$, we get $R^*(Mx, Nx)$. Then $R(P, Q)$ implies as in (iv) that $R(MP, NQ)$. ■

The following property R states that an M essentially does not contain the constants from \mathcal{D} . A term $M \in \Lambda_o[\mathcal{D}]$ is called *pure* iff $M \in \Lambda_o$. The property $R(M)$ states that M is convertible to a pure term.

3.3.18. PROPOSITION. Define for $M \in \Lambda_o[\mathcal{D}](A)$

$$R_A(M) \iff \exists N \in \Lambda_o(A) M =_{\beta\eta} N.$$

Then

- (i) R is logical.
- (ii) R is expansive.
- (iii) R is substitutive.

PROOF. (i) If $R(M)$ and $R(N)$, then clearly $R(MN)$. Conversely, suppose $\forall N [R(N) \Rightarrow R(MN)]$. Since obviously $R(x)$ it follows that $R(Mx)$ for fresh x . Hence there exists a pure $L =_{\beta\eta} Mx$. But then $\lambda x. L =_{\beta\eta} M$, hence $R(M)$.

(ii) Trivial as $P \rightarrow_{\text{wh}} Q \Rightarrow P =_{\beta\eta} Q$.

(iii) We must show $R = R^*$. Suppose $R(M)$ and $R(*)$. Then $M = N$, with N pure and hence $M^* = N^*$ is pure, so $R^*(M)$. Conversely, suppose $R^*(M)$. Then for $*$ with $x^* = x$ one has $R(*)$. Hence $R(M^*)$. But this is $R(M)$. ■

3.3.19. PROPOSITION. Let S be an n -ary logical, expansive and substitutive relation on terms from $\mathcal{D}_1, \dots, \mathcal{D}_n$. Define the restriction to pure terms $S \upharpoonright \Lambda$, again a relation on terms from $\mathcal{D}_1, \dots, \mathcal{D}_n$, by

$$(S \upharpoonright \Lambda)_A(M_1, \dots, M_n) \iff R(M_1) \& \dots \& R(M_n) \& S_A(M_1, \dots, M_n),$$

where R is as in proposition 3.3.18. Then $S \upharpoonright \Lambda$ is logical, expansive and substitutive.

PROOF. Intersection of relations preserves the notion logical, expansive and substitutive. ■

3.3.20. PROPOSITION. Given \mathcal{E} , define $R = R_{\mathcal{E}}$ by

$$R(M, N) \iff \mathcal{E} \vdash M = N.$$

Then

- (i) R is logical.
- (ii) R is expansive.
- (iii) R is substitutive.
- (iv) R is a congruence relation.

PROOF. (i) This is proved in several steps.

(1) If $R(M, N)$ with witness list F_1, \dots, F_n , then also $F_1, \dots, F_n, K(KN)$ is a witness list of this fact.

(2) If $R(M_1, M_2)$ with witness list F_1, \dots, F_n and $R(N_1, N_2)$ with list G_1, \dots, G_m , then we may suppose by (1) that $n = m$. But then $R(M_1N_1, M_2N_2)$ with list $\lambda xy.F_1xy(G_1xy), \dots, F_nxy(G_nxy)$.

(3) $R(x, x)$ holds with witness list $K(Kx)$.

(4) Suppose that whenever we have $R(N_1, N_2)$ we also have $R(M_1N_1, M_2N_2)$. Then for fresh z by (3) we have $R(M_1z, M_2z)$ with witnesses list, say, F_1, \dots, F_n . Then $R(M_1, M_2)$ with list $\lambda xyz.F_1xy, \dots, \lambda xyz.F_nxy$.

(ii) Obvious, since provability from \mathcal{E} is closed under β -conversion, hence *a fortiori* under weak head expansion.

(iii) Assume that $R(M, N)$ in order to show $R^*(M, N)$. So suppose $R(x^{*1}, x^{*2})$. We must show $R(M^{*1}, N^{*2})$. Now going back to the definition of R this means that we have $\mathcal{E} \vdash M = N$ and $\mathcal{E} \vdash x^{*1} = x^{*2}$ and we must show $\mathcal{E} \vdash M^{*1} = N^{*2}$. Now if $FV(MN) \subseteq \{\vec{x}\}$, then

$$\begin{aligned} M^{*1} &=_{\beta} (\lambda \vec{x}.M)\vec{x}^{*1} \\ &=_{\mathcal{E}} (\lambda \vec{x}.N)\vec{x}^{*2} \\ &=_{\beta} N^{*2}. \end{aligned}$$

(iv) Obvious. ■

Semantic Logical Relations

3.3.21. DEFINITION. Let $\mathcal{M}_1, \dots, \mathcal{M}_n$ be applicative typed structures.

(i) R is an n -ary family of relations or just a relation on $\mathcal{M}_1 \times \dots \times \mathcal{M}_n$ iff $R = \{R_A\}_{A \in \mathbb{T}}$ and for all A

$$R_A \subseteq \mathcal{M}_1(A) \times \dots \times \mathcal{M}_n(A).$$

(ii) R is a logical relation iff

$$R_{A \rightarrow B}(d_1, \dots, d_n) \iff \forall e_1 \in \mathcal{M}_1(A) \dots e_n \in \mathcal{M}_n(A) [R_A(e_1, \dots, e_n) \Rightarrow R_B(d_1e_1, \dots, d_ne_n)].$$

for all A, B and all $d_1 \in \mathcal{M}_1(A \rightarrow B), \dots, d_n \in \mathcal{M}_n(A \rightarrow B)$.

Note that R is an n -ary relation on $\mathcal{M}_1 \times \dots \times \mathcal{M}_n$ iff R is a unary relation on the single structure $\mathcal{M}_1 \times \dots \times \mathcal{M}_n$.

3.3.22. EXAMPLE. Define R on $\mathcal{M} \times \mathcal{M}$ by $R(d_1, d_2) \iff d_1 = d_2$. Then R is logical.

3.3.23. EXAMPLE. Let \mathcal{M} be a model and let $\pi = \pi_o$ be a permutation of $\mathcal{M}(o)$ which happens to be an element of $\mathcal{M}(o \rightarrow o)$. Then π can be lifted to higher types by defining

$$\pi_{A \rightarrow B}(d) = \lambda e \in \mathcal{M}(A). \pi_B(d(\pi_A^{-1}(e))).$$

Now define R_π (the graph of π)

$$R_\pi(d_1, d_2) \iff \pi(d_1) = d_2.$$

Then R is logical.

3.3.24. EXAMPLE. (Friedman [1975]) Let \mathcal{M}, \mathcal{N} be typed structures. A *partial surjective homomorphism* is a family $h = \{h_A\}_{A \in \mathcal{A}}$ of surjections

$$h_A : \mathcal{M}(A) \rightarrow \mathcal{N}(A)$$

such that h_o is a surjection and

$$h_{A \rightarrow B}(d) = e \iff \begin{array}{l} e \in \mathcal{N}(A \rightarrow B) \text{ is the unique element (if it exists)} \\ \text{such that } \forall f \in \text{dom}(h_A) \ e(h_A(f)) = h_B(d \cdot f). \end{array}$$

This implies that, if all elements involved exist, then

$$h_{A \rightarrow B}(d)h_A(f) = h_B(df).$$

Note that $h(d)$ can fail to be defined if one of the following conditions hold

1. for some $f \in \text{dom}(h_A)$ one has $df \notin \text{dom}(h_B)$;
2. the correspondence $h_A(f) \mapsto h_B(df)$ fails to be single valued;
3. the map $h_A(f) \mapsto h_B(df)$ fails to be in $\mathcal{N}_{A \rightarrow B}$.

Of course, 3 is the basic reason for partiality, whereas 1 and 2 are derived reasons. A partial surjective homomorphism h is completely determined by its h_o . If we take $\mathcal{M} = \mathcal{M}_X$ and h_o is any surjection $X \rightarrow \mathcal{N}_o$, then h_A is, although partial, indeed surjective for all A . Define $R_A(d, e) \iff h_A(d) = e$, the graph of h . Then R is logical. Conversely, if R_o is the graph of a partial surjective map $h_o : \mathcal{M}(o) \rightarrow \mathcal{N}(o)$, and the logical relation R induced by this R_o satisfies

$$\forall e \in \mathcal{N}(B) \exists d \in \mathcal{M}(A) \ R_B(d, e),$$

then R is the graph of a partial homomorphism from \mathcal{M} to \mathcal{N} .

3.3.25. DEFINITION. Let $\mathcal{M}_1, \dots, \mathcal{M}_n$ be typed structures.

(i) Let R be an n -ary relation on $\mathcal{M}_1, \dots, \mathcal{M}_n$. For valuations ρ_1, \dots, ρ_n with $\rho_i : \text{Var} \rightarrow \mathcal{M}_i$ we define

$$R(\rho_1, \dots, \rho_n) \iff R(\rho_1(x), \dots, \rho_n(x)), \text{ for all variables } x.$$

(ii) Let R be an n -ary relation on $\mathcal{M}_1 \times \dots \times \mathcal{M}_n$. The *lifting* of R to $\mathcal{M}_1^*, \dots, \mathcal{M}_n^*$, notation R^* , is defined by

$$R^*(d_1, \dots, d_n) \iff \forall \rho_1, \dots, \rho_n [R(\rho_1, \dots, \rho_n) \Rightarrow R(\llbracket d_1 \rrbracket_{\rho_1}, \dots, \llbracket d_n \rrbracket_{\rho_n})].$$

Here the ρ_i range over valuations in \mathcal{M}_i .

(iii) Let now R be an n -ary relation on $\mathcal{M}_1^*, \dots, \mathcal{M}_n^*$. We say that R is *substitutive* iff $R = R^*$.

3.3.26. EXAMPLE. (i) Let R be the equality relation on $\mathcal{M} \times \mathcal{M}$. Then R^* is the equality relation on $\mathcal{M}^* \times \mathcal{M}^*$.

(ii) If R is the graph of a surjective homomorphism, then R^* is the graph of a partial surjective homomorphism whose restriction to \mathcal{M} is R and which fixes each indeterminate x . [[Restriction in the literal sense, not the analogue of 3.3.19.]]

3.3.27. THEOREM. (*Fundamental Theorem for semantic logical relations*) Let $\mathcal{M}_1, \dots, \mathcal{M}_n$ be models and let R be a logical relation on $\mathcal{M}_1, \dots, \mathcal{M}_n$. Then for each closed and pure term $M \in \Lambda_o^\emptyset$ one has

$$R(\llbracket M \rrbracket^{\mathcal{M}_1}, \dots, \llbracket M \rrbracket^{\mathcal{M}_n}).$$

PROOF. We treat the case $n = 1$. Let $R \subseteq \mathcal{M}$ be logical. We claim that for all $M \in \Lambda_o$ and all partial valuations ρ such that $\text{dom}(\rho) \subseteq \text{FV}(M)$ one has

$$R(\rho) \Rightarrow R(\llbracket M \rrbracket_\rho).$$

This follows by an easy induction on M . In case $M \equiv \lambda x. N$ one should show $R(\llbracket \lambda x. N \rrbracket_\rho)$, assuming $R(\rho)$. This means that for all d of the right type with $R(d)$ one has $R(\llbracket \lambda x. N \rrbracket_\rho d)$. This is the same as $R(\llbracket N \rrbracket_{\rho[x:=d]})$, which holds by the induction hypothesis.

The statement now follows immediately from the claim, by taking as ρ the empty function. ■

Relating syntactic and semantic logical relations

One may wonder whether the Fundamental Theorem for semantic logical relations follows from the syntactic version. This indeed is the case. The notion \mathcal{M}^* for a model \mathcal{M} will be useful for expressing the relation between syntactic and semantic logical relations.

3.3.28. PROPOSITION. (i) The universal relation $R = \mathcal{M}_1^* \times \dots \times \mathcal{M}_n^*$ is substitutive and logical on $\mathcal{M}_1^*, \dots, \mathcal{M}_n^*$.

(ii) If R and S are respectively an n -ary relation on $\mathcal{M}_1, \dots, \mathcal{M}_n$ and an m -ary relation on $\mathcal{N}_1, \dots, \mathcal{N}_m$ and are both non-empty logical, then $R \times S$ is a $n + m$ -ary non-empty logical relation on $\mathcal{M}_1, \dots, \mathcal{M}_n, \mathcal{N}_1, \dots, \mathcal{N}_m$. If moreover both R and S are substitutive, then so is $R \times S$.

(iii) If R is an n -ary logical relation on \mathcal{M}^n and π is a permutation of $\{1, \dots, n\}$, then R^π defined by

$$R^\pi(d_1, \dots, d_n) \iff R(d_{\pi(1)}, \dots, d_{\pi(n)})$$

is a logical relation. If moreover R is substitutive, then so is R^π .

(iv) If R is an n -ary substitutive logical relation on $\mathcal{M}^* \times \dots \times \mathcal{M}^*$, then the diagonal R^Δ defined by

$$R^\Delta(d) \iff R(d, \dots, d)$$

is a unary substitutive logical relation on \mathcal{M}^* .

(v) If \mathcal{R} is a class of n -ary substitutive logical relations, then $\cap \mathcal{R}$ is a substitutive logical relation.

(vi) If R is an $(n + 1)$ -ary substitutive logical relation on $\mathcal{M}_1^*, \dots, \mathcal{M}_{n+1}^*$ and \mathcal{M}_{n+1}^* is a model, then $\exists R$ defined by

$$\exists R(d_1, \dots, d_n) \iff \exists d_{n+1} R(d_1, \dots, d_{n+1})$$

is an n -ary substitutive logical relation.

(vii) If R is an n -ary substitutive logical relation on $\mathcal{M}_1, \dots, \mathcal{M}_n$, then R^* is an n -ary substitutive logical relation on $\mathcal{M}^* \times \dots \times \mathcal{M}^*$.

(viii) If R is an n -ary substitutive logical relation on $\mathcal{M}_1, \dots, \mathcal{M}_n$ and each \mathcal{M}_i is a model, then

$$R^*(d_1, \dots, d_n) \iff R(\lambda \vec{x}.d_1, \dots, \lambda \vec{x}.d_n),$$

where the variables on which the \vec{d} depend are included in the list \vec{x} .

PROOF. All items are easy. As an example we do the last one for $n = 1$.

$$\begin{aligned} R^*(d) &\iff \forall \rho. [R(\rho) \Rightarrow R(\llbracket d \rrbracket_\rho)] \\ &\iff \forall \rho. [R(\rho) \Rightarrow R(\llbracket (\lambda \vec{x}.d) \vec{x} \rrbracket_\rho)] \\ &\iff \forall \vec{e}. [R(e_1) \Rightarrow \dots \Rightarrow R(e_n) \Rightarrow R((\lambda \vec{x}.d) e_1 \dots e_n)] \\ &\iff R(\lambda \vec{x}.d). \blacksquare \end{aligned}$$

3.3.29. EXAMPLE. Consider $\mathcal{M}_{\mathbb{N}}$ and define

$$R_o(n, m) \iff n \leq m,$$

where \leq is the usual ordering on \mathbb{N} . Then $R^* \cap =^*$ is [[related to]] the set of hereditarily monotone functionals. Moreover $\exists(R^*)$ is [[related to]] the set of hereditarily majorizable functionals, see the section by Howard in Troelstra [1973].

Now we want to link the notions of syntactical and semantical logical relation.

3.3.30. NOTATION. Let \mathcal{M} be an applicative structure. Write

$$\Lambda_o[\mathcal{M}] = \Lambda_o[\{\mathbf{c}_d \mid d \in \mathcal{M}\}].$$

3.3.31. DEFINITION. Define the binary relation $D \subseteq \mathcal{M}^\Lambda \times \mathcal{M}$ by

$$D(M, d) \iff \forall \rho \llbracket M^{\beta\eta} \rrbracket_\rho = d,$$

where $M^{\beta\eta}$ is the $\beta\eta$ -nf of M .

3.3.32. PROPOSITION. D is a substitutive logical relation.

PROOF. Substitutivity is easy. Moreover, we have

$$\begin{aligned} \llbracket M^{\beta\eta} \rrbracket = d &\Rightarrow \forall N, e. \llbracket N^{\beta\eta} \rrbracket = e \Rightarrow \llbracket (MN)^{\beta\eta} \rrbracket = \llbracket M^{\beta\eta} \rrbracket \llbracket N^{\beta\eta} \rrbracket = de \\ &\Rightarrow \llbracket (Mx)^{\beta\eta} \rrbracket = dx, \text{ with } x \text{ fresh,} \\ &\Rightarrow \llbracket M^{\beta\eta} \rrbracket = d. \end{aligned}$$

Therefore

$$\begin{aligned} D_{A \rightarrow B}(M, d) &\iff \llbracket M^b e \rrbracket = d \\ &\iff \forall N, e. \llbracket N^{\beta\eta} \rrbracket = e \Rightarrow \llbracket (MN)^{\beta\eta} \rrbracket = de \\ &\iff \forall N, e. D_A(N, e) \Rightarrow D_B(MN, de). \blacksquare \end{aligned}$$

D is the connection between the two notions of logical relation.

3.3.33. DEFINITION. Let $\mathcal{M}_1, \dots, \mathcal{M}_n$ be typed structures.

(i) Let R be a logical relation on $\mathcal{M}_1, \dots, \mathcal{M}_n$. Let R^\wedge be the relation on $\Lambda[\mathcal{M}_1], \dots, \Lambda[\mathcal{M}_n]$ defined by

$$\begin{aligned} R^\wedge(M_1, \dots, M_n) &\iff \exists d_1 \in \mathcal{M}_1 \dots d_n \in \mathcal{M}_n \\ &\quad [D^*(M_1, d_1) \ \& \ \dots \ D^*(M_n, d_n) \ \& \ R^*(d_1, \dots, d_n)]. \end{aligned}$$

(ii) Let S be a logical relation on $\Lambda[\mathcal{M}_1], \dots, \Lambda[\mathcal{M}_n]$. Let S^\vee be the relation on $\mathcal{M}_1, \dots, \mathcal{M}_n$ defined by

$$\begin{aligned} S^\vee(d_1, \dots, d_n) &\iff \exists M_1 \in \Lambda[\mathcal{M}_1] \dots M_n \in \Lambda[\mathcal{M}_n] \\ &\quad [D^*(M_1, d_1) \ \& \ \dots \ D^*(M_n, d_n) \ \& \ S^*(M_1, \dots, M_n)]. \end{aligned}$$

3.3.34. LEMMA. (i) If R on $\mathcal{M}_1, \dots, \mathcal{M}_n$ is logical, then on $\Lambda(\mathcal{M}_1), \dots, \Lambda(\mathcal{M}_n)$ the relation R^\wedge is expansive and logical.

(ii) If S on $\Lambda(\mathcal{M}_1), \dots, \Lambda(\mathcal{M}_n)$ is expansive and logical, then S^\vee is logical.

PROOF. (i) First we show that R^\wedge is logical. For notational simplicity we take $n = 1$. We must show

$$\begin{aligned} R^\wedge(M) &\iff \forall N. [R^\wedge(N) \Rightarrow R^\wedge(MN)] \\ \text{i.e.} \quad \forall \rho. R(\llbracket M^{\beta\eta} \rrbracket_\rho) &\iff \forall N. [(\forall \rho. R(\llbracket N^{\beta\eta} \rrbracket_\rho)) \Rightarrow (\forall \rho. R(\llbracket (MN)^{\beta\eta} \rrbracket_\rho))]. \end{aligned}$$

Now (\Rightarrow) is trivial. As to (\Leftarrow) , suppose that $\llbracket M \rrbracket_\rho = d$ and that for all e with $R(e)$ one has $R(de)$ in order to show $R(d)$. Take $N = \underline{e}$. Then $R(\llbracket M\underline{e} \rrbracket_\rho)$, hence $R(de)$. Therefore $R(d)$, as R is logical. This shows that R^\wedge is logical. This relation is expansive by the fact that in the definition of D one takes the $\beta\eta$ -nf. [[Make notation c_d vs \underline{d} consistent.]] ■

3.3.35. PROPOSITION. *Let $\mathcal{M}_1, \dots, \mathcal{M}_n$ be models. Let R on $\mathcal{M}_1, \dots, \mathcal{M}_n$ be logical and let S on $\Lambda[\mathcal{M}]_1, \dots, \Lambda[\mathcal{M}]_n$ be expansive and logical. Then*

- (i) R^\wedge is a substitutive logical relation.
- (ii) $R^{\wedge\vee} = R^*$.
- (iii) $S^{\vee\wedge} = S^*$.

PROOF. (i) In order to show that R^\wedge is substitutive, assume $R^\wedge(M)$, $R^\wedge(\vec{N})$ towards $R^\wedge(M[\vec{x} := \vec{N}])$. Now $R^\wedge(M)$ means that $R(\llbracket M^{\beta\eta} \rrbracket_\rho)$, where $\llbracket M^{\beta\eta} \rrbracket_\rho$ is independent of ρ . Since the \vec{M} are models we can leave out the $\beta\eta$. Therefore

$$\llbracket M[\vec{x} := \vec{N}] \rrbracket_\rho = \llbracket M \rrbracket_{\rho[\vec{x} := [\vec{N}]]} = \llbracket M \rrbracket,$$

hence $R^\wedge(M[\vec{x} := \vec{N}])$.

- (ii) Write $T = R^\wedge$. Then (taking $n = 1$)

$$\begin{aligned} T^\vee(d) &\iff \exists M \in \Lambda[\mathcal{M}]. D^*(M, d) \ \& \ T(M), \\ &\iff \exists M \in \Lambda[\mathcal{M}]. D^*(M, d) \ \& \ \exists d'. D^*(M, d') \ \& \ R^*(d'), \\ &\iff \exists M, d'. \llbracket M \rrbracket = d \ \& \ \llbracket M \rrbracket = d' \ \& \ R^*(d'), \\ &\iff R^*(d). \end{aligned}$$

- (iii) Similarly. ■

Using this result, the Fundamental Theorem for syntactical logical relations can be derived from the syntactic version.

We give two applications.

3.3.36. EXAMPLE. Let R be the graph of a partial surjective homomorphism $h : \mathcal{M} \rightarrow \mathcal{N}$. The fundamental theorem just shown implies that for closed pure terms one has $h(M) = M$, which is lemma 15 of Friedman [1975]. From this it is derived in this paper that for infinite X one has

$$\mathcal{M}_X \models M = N \iff M =_{\beta\eta} N.$$

We have derived this in another way.

3.3.37. **EXAMPLE.** Let \mathcal{M} be a typed structure. Let $\Delta \subseteq \mathcal{M}$. Write $\Delta(A) = \Delta \cap \mathcal{M}(A)$. Assume that $\Delta(A) \neq \emptyset$ for all $A \in$ and

$$d \in \Delta(A \rightarrow B), e \in \Delta(A) \Rightarrow de \in \Delta(B).$$

Then Δ may fail to be a typed structure because it is not extensional. Equality as binary relation E_o on $\Delta(o) \times \Delta(o)$ induces a binary logical relation E on $\Delta \times \Delta$. Let $\Delta^E = \{d \in \Delta \mid E(d, d)\}$. Then the restriction of E to Δ^E is an applicative congruence and the equivalence classes form a structure. In particular, if \mathcal{M} is a model, then write

$$\Delta^+ = \{d \in \mathcal{M} \mid \exists M \Lambda_o^\emptyset \exists d_1 \dots d_n \llbracket M \rrbracket d_1 \dots d_n = d\}$$

for the applicative closure of Δ . The *Gandy hull* of Δ in \mathcal{M} is the set Δ^{+E} . From the fundamental theorem for logical relations it can be derived that

$$\mathcal{M}_\Delta = \Delta^{+E}/E$$

is a model. This model will be also called the Gandy hull of Δ in \mathcal{M} .

3.4. Type reducibility

Remember that a type A is reducible to type B , notation $A \leq_{\beta\eta} B$ if for some closed term $\Phi:A \rightarrow B$ one has for all closed $M_1, M_2:A$

$$M_1 =_{\beta\eta} M_2 \iff \Phi M_1 =_{\beta\eta} \Phi M_2.$$

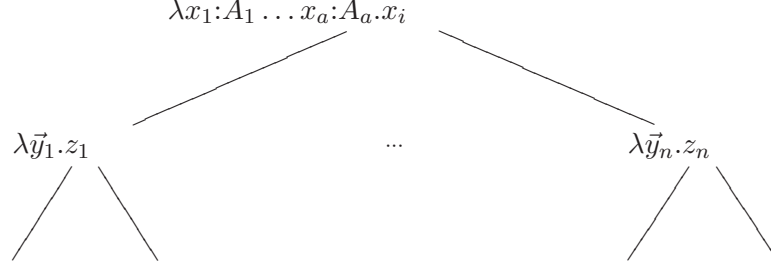
3.4.1. **DEFINITION.** Write $A \sim_{\beta\eta} B$ iff $A \leq_{\beta\eta} B$ & $B \leq_{\beta\eta} A$.

The reducibility theorem, Statman [1980a], states that there is one type to which all types of $\Pi(\lambda_{\rightarrow})$ can be reduced. At first this may seem impossible. Indeed, in a full typed structure \mathcal{M} the cardinality of the sets of higher type increase arbitrarily. So one cannot always have an injection $\mathcal{M}_A \rightarrow \mathcal{M}_B$. But reducibility means that one restricts oneself to definable elements (modulo $=_{\beta\eta}$) and then the injections are possible. The proof will occupy 3.4.2-3.4.7. There are four main steps. In order to show that $\Phi M_1 =_{\beta\eta} \Phi M_2 \Rightarrow M_1 =_{\beta\eta} M_2$ in all cases a (pseudo) inverse Φ^{-1} is used. Pseudo means that sometimes the inverse is not lambda definable, but this is no problem for the implication. Sometimes Φ^{-1} is definable, but the property $\Phi^{-1}(\Phi M) = M$ only holds in an extension of the theory; because the extension will be conservative over $=_{\beta\eta}$ the reducibility follows. Next the type hierarchy theorem, also due to Statman [1980a], will be given. Rather unexpectedly it turns out that under $\leq_{\beta\eta}$ types form a well-ordering of length $\omega + 3$. Finally some consequences of the reducibility theorem will be given, including the 1-section and finite completeness theorems.

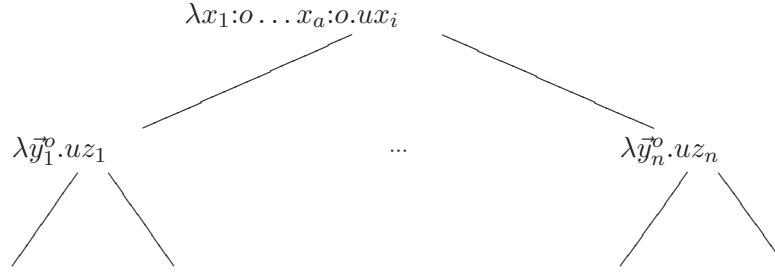
In the first step towards the reducibility theorem it will be shown that every type is reducible to one of rank ≤ 3 . The proof is rather syntactic. In order to show that the definable function Φ is 1-1, a non-definable inverse is needed. A warmup exercise for this is 3.6.4.

3.4.2. PROPOSITION. *For every type A there is a type B such that $\mathbf{rank}(B) \leq 3$ and $A \leq_{\beta\eta} B$.*

PROOF. [The intuition behind the construction of the the term Φ responsible for the reducibility is as follows. If M is a term with Böhmtree (see Barendregt [1984])



Now let UM be a term with “Böhmtree” of the form



where all the typed variables are pushed down to type o and the variables u (each occurrence possibly different) takes care that the new term remains typable. From this description it is clear that the u can be chosen in such way that the result has $\mathbf{rank} \leq 1$. Also that M can be reconstructed from UM so that U is injective. ΦM is just UM with the auxiliary variables bound. This makes it of type with $\mathbf{rank} \leq 3$. What is less clear is that U and hence Φ are lambda-definable.]

Define inductively for any type A the types A^\sharp and A^\flat .

$$\begin{aligned} o^\sharp &= o; \\ o^\flat &= o; \\ (A_1 \rightarrow \dots \rightarrow A_a \rightarrow o)^\sharp &= o \rightarrow A_1^\flat \rightarrow \dots \rightarrow A_a^\flat \rightarrow o; \\ (A_1 \rightarrow \dots \rightarrow A_a \rightarrow o)^\flat &= (o^a \rightarrow o). \end{aligned}$$

Notice that $\mathbf{rank}(A^\sharp) \leq 2$.

In the potentially infinite context

$$\{u_A : A^\sharp \mid A \in \mathbb{T}\}$$

define inductively for any type A terms $V_A : o \rightarrow A, U_A : A \rightarrow A^\flat$.

$$\begin{aligned} U_o &= \lambda x:o.x; \\ V_o &= \lambda x:o.x; \\ U_{A_1 \rightarrow \dots \rightarrow A_a \rightarrow o} &= \lambda z:A \lambda x_1, \dots, x_a:o. z(V_{A_1} x_1) \dots (V_{A_a} x_a); \\ V_{A_1 \rightarrow \dots \rightarrow A_a \rightarrow o} &= \lambda x:o \lambda y_1:A_1 \dots y_a:A_a. u_A x(U_{A_1} y_1) \dots (U_{A_a} y_a), \end{aligned}$$

where $A = A_1 \rightarrow \dots \rightarrow A_a \rightarrow o$.

Notice that for a closed term M of type $A = A_1 \rightarrow \dots \rightarrow A_a \rightarrow o$ one can write

$$M = \lambda y_1:A_1 \dots y_a:A_a. y_i(M_1 y_1 \dots y_a) \dots (M_n y_1 \dots y_a).$$

Now verify that

$$\begin{aligned} U_A M &= \lambda x_1, \dots, x_a:o. M(V_{A_1} x_1) \dots (V_{A_a} x_a) \\ &= \lambda \vec{x}. (V_{A_i} x_i)(M_1(V_{A_1} x_1) \dots (V_{A_a} x_a)) \dots (M_n(V_{A_1} x_1) \dots (V_{A_a} x_a)) \\ &= \lambda \vec{x}. u_{A_i} x_i(U_{A_1}(M_1(V_{A_1} x_1) \dots (V_{A_a} x_a))) \dots \\ &= \lambda \vec{x}. u_{A_i} x_i(U_{A_1 \rightarrow \dots \rightarrow A_a \rightarrow B_1} M_1 \vec{x}) \dots, \end{aligned}$$

where B_j is the type of M_j . Hence we have that if $U_A M =_{\beta\eta} U_A N$, then for $1 \leq j \leq n$

$$U_{A_1 \rightarrow \dots \rightarrow A_a \rightarrow B_j} M_j =_{\beta\eta} U_{A_1 \rightarrow \dots \rightarrow A_a \rightarrow B_j} N_j.$$

Therefore it follows by induction on the complexity of M that if $U_A M =_{\beta\eta} U_A N$, then $M =_{\beta\eta} N$.

Now take as term for the reducibility $\Phi \equiv \lambda m:A \lambda u_{B_1} \dots u_{B_k}. U_A m$, where the \vec{u} are all the ones occurring in the construction of U_A . It follows that

$$A \leq_{\beta\eta} B_1^\sharp \rightarrow \dots \rightarrow B_k^\sharp \rightarrow A^\sharp.$$

Since $\text{rank}(B_1^\sharp \rightarrow \dots \rightarrow B_k^\sharp \rightarrow A^\sharp) \leq 3$, we are done. ■

For an alternative proof, see Exercise 3.6.9.

In the following proposition it will be proved that we can further reduce types to one particular type of rank 3. First do exercise 3.6.5 to get some intuition. We need the following notation.

3.4.3. NOTATION. (i) For $k \geq 0$ write

$$1_k = o^k \rightarrow o,$$

where in general $A^0 \rightarrow o = o$ and $A^{k+1} \rightarrow o = A \rightarrow (A^k \rightarrow o)$.

(ii) For $k_1, \dots, k_n \geq 0$ write

$$(k_1, \dots, k_n) = 1_{k_1} \rightarrow \dots \rightarrow 1_{k_n} \rightarrow o.$$

(iii) For $k_{11}, \dots, k_{1n_1}, \dots, k_{m1}, \dots, k_{mn_m} \geq 0$ write

$$\begin{pmatrix} k_{11} & \dots & k_{1n_1} \\ \vdots & & \vdots \\ k_{m1} & \dots & k_{mn_m} \end{pmatrix} = (k_{11}, \dots, k_{1n_1}) \rightarrow \dots \rightarrow (k_{m1}, \dots, k_{mn_m}) \rightarrow o.$$

Note the “matrix” has a dented right side (the n_i are unequal in general).

3.4.4. PROPOSITION. *Every type A of rank ≤ 3 is reducible to*

$$1_2 \rightarrow 1 \rightarrow 1 \rightarrow 2 \rightarrow o.$$

PROOF. Let A be a type of rank ≤ 3 . It is not difficult to see that A is of the form

$$A = \begin{pmatrix} k_{11} & \dots & k_{1n_1} \\ \vdots & & \vdots \\ k_{m1} & \dots & k_{mn_m} \end{pmatrix}$$

We will first reduce A to type $3 = 2 \rightarrow o$ using a term Φ containing free variables of type $1_2, 1, 1$ respectively acting as a ‘pairing’. Consider the context

$$\{p:1_2, p_1:1, p_2:1\}.$$

Consider the notion of reduction p defined by the contraction rules

$$p_i(pM_1M_2) \rightarrow_p M_1.$$

[There now is a choice how to proceed: if you like syntax, then proceed; if you prefer models omit paragraphs starting with ♣ and jump to those starting with ♠.]

♣ This notion of reduction satisfies the subject reduction property. Moreover $\beta\eta p$ is Church-Rosser, see Pottinger [1981]. This can be used later in the proof. [Extending the notion of reduction by adding

$$p(p_1M)(p_2M) \rightarrow_s M$$

preserves the CR property. In the untyped calculus this is not the case, see Klop [1980] or Barendregt [1984], ch. 14.] Goto ♠.

♠ Given the pairing p, p_1, p_2 one can extend it as follows. Write

$$\begin{aligned} p^1 &= \lambda x:o.x; \\ p^{k+1} &= \lambda x_1 \dots x_n x_{n+1}:o.p(p^k x_1 \dots x_n) x_{n+1}; \\ p_1^1 &= \lambda x:o.x; \\ p_{k+1}^{k+1} &= p_2; \\ p_i^{k+1} &= \lambda z:o.p_i^k(p_1 z), & \text{for } i \leq k; \\ P^k &= \lambda f_1 \dots f_k:1 \lambda z:o.p^k(f_1 z) \dots (f_k z); \\ P_i^k &= \lambda g:1 \lambda z:o.p_i^k(g z), & \text{for } i \leq k. \end{aligned}$$

We have that p^k acts as a coding for k -tuples of elements of type o with projections p_i^k . The P^k, P_i^k do the same for type 1. In context containing $\{f:1_k, g:1\}$ write

$$\begin{aligned} f^{k \rightarrow 1} &= \lambda z:o.f(p_1^k z) \dots (p_k^k z); \\ g^{1 \rightarrow k} &= \lambda z_1 \dots z_k:o.f(p^k z_1 \dots z_k). \end{aligned}$$

Then $f^{k \rightarrow 1}$ is f moved to type 1 and $g^{1 \rightarrow k}$ is g moved to type 1_k .

Using $\beta\eta p$ -convertibility one can show

$$\begin{aligned} p_i^k(p^k z_1 \dots z_k) &= z_i; \\ P_i^k(P^k f_1 \dots f_k) &= f_i; \\ f^{k \rightarrow 1, 1 \rightarrow k} &= f. \end{aligned}$$

For $g^{1 \rightarrow k, k \rightarrow 1} = g$ one needs s , the surjectivity of the pairing.

In order to define the term required for the reducibility start with the term $\Psi:A \rightarrow 3$ (containing p, p_1, p_2 as only free variables). We need an auxiliary term Ψ^{-1} , acting as an inverse for Ψ in the presence of a “true pairing”.

$$\begin{aligned} \Psi &\equiv \lambda M \lambda F:2.M \\ &\quad [\lambda f_{11}:1_{k_{11}} \dots f_{1n_1}:1_{k_{1n_1}}.p_1(F(P^{n_1} f_{11}^{k_{11} \rightarrow 1} \dots f_{1n_1}^{k_{1n_1} \rightarrow 1}))] \dots \\ &\quad [\lambda f_{m1}:1_{k_{m1}} \dots f_{mn_m}:1_{k_{mn_m}}.p_m(F(P^{n_m} f_{m1}^{k_{m1} \rightarrow 1} \dots f_{mn_m}^{k_{mn_m} \rightarrow 1}))]; \\ \Psi^{-1} &\equiv \lambda N:(2 \rightarrow o) \lambda K_1:(k_{11}, \dots, k_{1n_1}) \dots \lambda K_m:(k_{m1}, \dots, k_{mn_m}). \\ &\quad N(\lambda f:1.p^m[K_1(P_1^{n_1} f)^{1 \rightarrow k_{11}} \dots (P_{n_1}^{n_1} f)^{1 \rightarrow k_{1n_1}}] \dots \\ &\quad [K_m(P_1^{n_m} f)^{1 \rightarrow k_{m1}} \dots (P_{n_m}^{n_m} f)^{1 \rightarrow k_{1n_m}}]). \end{aligned}$$

Claim. For closed terms M_1, M_2 of type A we have

$$M_1 =_{\beta\eta} M_2 \iff \Psi M_1 =_{\beta\eta} \Psi M_2.$$

It then follows that for the reduction $A \leq_{\beta\eta} 1_2 \rightarrow 1 \rightarrow 1 \rightarrow 3$ we can take

$$\Phi = \lambda M:A.\lambda p:1_2 \lambda p_1, p_2:1.\Psi M.$$

It remains to show the claim. The only interesting direction is (\Leftarrow) . This follows in two ways. We first show that

$$\Psi^{-1}(\Psi M) =_{\beta\eta p} M. \tag{1}$$

We will write down the computation for the “matrix”

$$\begin{pmatrix} k_{11} & \\ k_{21} & k_{22} \end{pmatrix}$$

which is perfectly general.

$$\begin{aligned}
\Psi M &=_{\beta} \lambda F:2.M[\lambda f_{11}:1_{k_{11}}.p_1(F(P^1 f_{11}^{k_{11} \rightarrow 1})) \\
&\quad [\lambda f_{21}:1_{k_{21}} \lambda f_{22}:1_{k_{22}}.p_2(F(P^2 f_{21}^{k_{21} \rightarrow 1} f_{22}^{k_{22} \rightarrow 1}))]]; \\
\Psi^{-1}(\Psi M) &=_{\beta} \lambda K_1:(k_{11}) \lambda K_2:(k_{21}, k_{22}). \\
&\quad \Psi M(\lambda f:1.p^1[K_1(P_1^1 f)^{1 \rightarrow k_{11}}[K_2(P_2^1 f)^{1 \rightarrow k_{21}}(P_2^2 f)^{1 \rightarrow k_{22}}]]) \\
&\equiv \lambda K_1:(k_{11}) \lambda K_2:(k_{21}, k_{22}). \Psi M H, \text{ say,} \\
&=_{\beta} \lambda K_1 K_2.M[\lambda f_{11}.p_1(H(P^1 f_{11}^{k_{11} \rightarrow 1})) \\
&\quad [\lambda f_{21} \lambda f_{22}.p_2(H(P^2 f_{21}^{k_{21} \rightarrow 1} f_{22}^{k_{22} \rightarrow 1}))]]; \\
&=_{\beta p} \lambda K_1 K_2.M[\lambda f_{11}.p_1(p^2[K_1 f_{11}][\dots \text{'junk'} \dots])] \\
&\quad [\lambda f_{21} \lambda f_{22}.p_2(p^2[\dots \text{'junk'} \dots][K_2 f_{21} f_{22}])]; \\
&=_p \lambda K_1 K_2.M(\lambda f_{11}.K_1 f_{11})(\lambda f_{21} f_{22}.K_2 f_{21} f_{22}) \\
&=_{\eta} \lambda K_1 K_2.M K_1 K_2 \\
&=_{\eta} M,
\end{aligned}$$

since

$$\begin{aligned}
H(P^1 f_{11}) &=_{\beta p} p^2[K_1 f_{11}][\dots \text{'junk'} \dots] \\
H(P^2 f_{21}^{k_{21} \rightarrow 1} f_{22}^{k_{22} \rightarrow 1}) &=_{\beta p} p^2[\dots \text{'junk'} \dots][K_2 f_{21} f_{22}].
\end{aligned}$$

The argument now can be finished in a model theoretic or syntactic way.

♣ If $\Psi M_1 =_{\beta \eta} \Psi M_2$, then $\Psi^{-1}(\Psi M_1) =_{\beta \eta} \Psi^{-1}(\Psi M_2)$. But then by (1) $M_1 =_{\beta \eta p} M_2$. It follows from the Church-Rosser theorem for $\beta \eta p$ that $M_1 =_{\beta \eta} M_2$, since these terms do not contain p . Goto ■.

♠ If $\Psi M_1 =_{\beta \eta} \Psi M_2$, then

$$\lambda p:1_2 \lambda p_1 p_2:1. \Psi^{-1}(\Psi M_1) =_{\beta \eta} \lambda p:1_2 \lambda p_1 p_2:1. \Psi^{-1}(\Psi M_2).$$

Hence

$$\mathcal{M}(\omega) \models \lambda p:1_2 \lambda p_1 p_2:1. \Psi^{-1} \Psi(M_1) = \lambda p:1_2 \lambda p_1 p_2:1. \Psi^{-1}(\Psi M_2).$$

Let \mathbf{q} be an actual pairing on ω with projections $\mathbf{q}_1, \mathbf{q}_2$. Then in $\mathcal{M}(\omega)$

$$(\lambda p:1_2 \lambda p_1 p_2:1. \Psi^{-1}(\Psi M_1)) \mathbf{q} \mathbf{q}_1 \mathbf{q}_2 = \lambda p:1_2 \lambda p_1 p_2:1. \Psi^{-1}(\Psi M_2) \mathbf{q} \mathbf{q}_1 \mathbf{q}_2.$$

Since $(\mathcal{M}(\omega), \mathbf{q}, \mathbf{q}_1, \mathbf{q}_2)$ is a model of $\beta \eta p$ conversion it follows from (1) that

$$\mathcal{M}(\omega) \models M_1 = M_2.$$

But then $M_1 =_{\beta \eta} M_2$, by a result of Friedman [1975]. ■

We will see below, corollary 3.4.23 (i), that Friedman's result will follow from the reducibility theorem. Therefore the syntactic approach is preferable.

The proof of the next proposition is again syntactic. A warmup is exercise 3.6.7.

3.4.5. PROPOSITION. *Let A be a type of rank ≤ 2 . Then*

$$2 \rightarrow A \leq_{\beta \eta} 1 \rightarrow 1 \rightarrow o \rightarrow A.$$

PROOF. Let $A \equiv (1^{k_1}, \dots, 1^{k_n}) = 1_{k_1} \rightarrow \dots 1_{k_n} \rightarrow o$. The term that will perform the reduction is relatively simple

$$\Phi \equiv \lambda M:2\lambda b_1:1_{k_1} \dots \lambda b_n:1_{k_n} . [\lambda f, g:1\lambda z:o. M(\lambda h:1. f(h(g(hz))))].$$

In order to show that for all $M_1, M_2:2 \rightarrow A$ one has

$$\Phi M_1 =_{\beta\eta} \Phi M_2 \Rightarrow M_1 =_{\beta\eta} M_2,$$

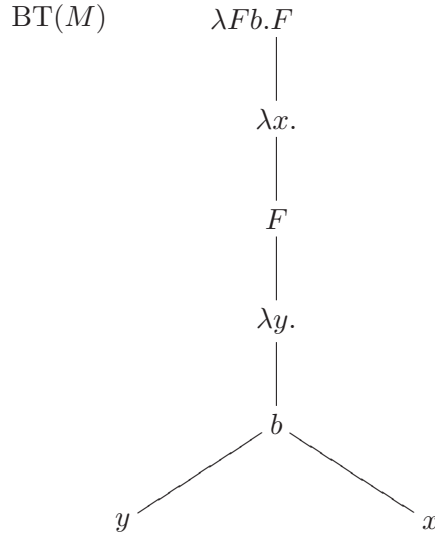
we may assume w.l.o.g. that $A = 1_2 \rightarrow o$. A typical element of $2 \rightarrow 1_2 \rightarrow o$ is

$$M \equiv \lambda F:2\lambda b:1_2. F(\lambda x. F(\lambda y. byx)).$$

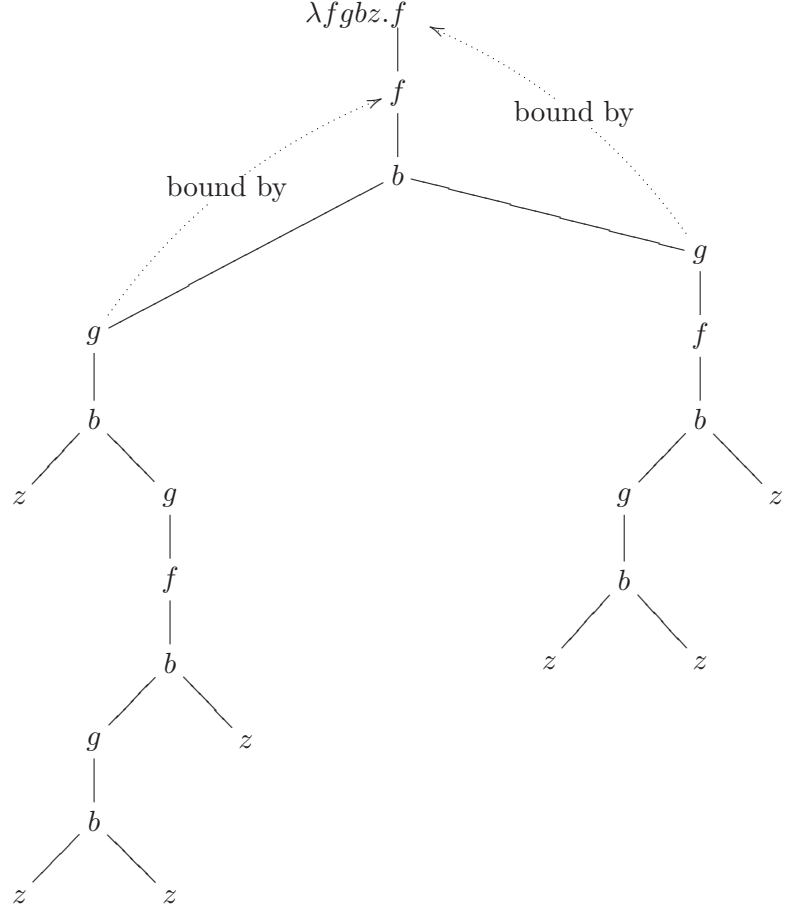
Note that its translation has the following long $\beta\eta$ -nf

$$\begin{aligned} \Phi M &= \lambda b:1_2 \lambda f, g:1\lambda z:o. f(N_x[x: = g(N_x[x: = z])]), \\ &\quad \text{where } N_x \equiv f(b(g(bzx))x), \\ &\equiv \lambda b:1_2 \lambda f, g:1\lambda z:o. f(f(b(g(bz[g(f(b(g(bzz))z])))[g(f(b(g(bzz))z))])). \end{aligned}$$

This term M and its translation have the following trees.



and

BT(ΦM)

Note that if we can ‘read back’ M from its translation ΦM , then we are done. Let $\text{Cut}_{g \rightarrow z}$ be a syntactic operation on terms that replaces maximal subterms of the form gP by z . For example (omitting the abstraction prefix)

$$\text{Cut}_{g \rightarrow z}(\Phi M) = f(f(bzz)).$$

Note that this gives us back the ‘skeleton’ of the term M , by reading f as $F(\lambda\odot$. The remaining problem is how to reconstruct the binding effect of each occurrence of the $\lambda\odot$. Using the idea of counting upwards lambda’s, see de Bruijn [1972], this is accomplished by a realizing that the occurrence z coming from $g(P)$ should be bound at the position f just above where $\text{Cut}_{g \rightarrow z}(P)$ matches in $\text{Cut}_{g \rightarrow z}(\Phi M)$ above that z . For a precise inductive argument for this fact, see Statman [1980a], Lemma 5, or do exercise 3.6.10. ■

The following simple proposition brings almost to an end the chain of reducibility of types.

3.4.6. PROPOSITION.

$$1^4 \rightarrow 1_2 \rightarrow o \rightarrow o \leq_{\beta\eta} 1_2 \rightarrow o \rightarrow o.$$

PROOF. As it is equally simple, let us prove instead

$$1 \rightarrow 1_2 \rightarrow o \rightarrow o \leq_{\beta\eta} 1_2 \rightarrow o \rightarrow o.$$

Define $\Phi : (1 \rightarrow 1_2 \rightarrow o \rightarrow o) \rightarrow 1_2 \rightarrow o \rightarrow o$ by

$$\Phi \equiv \lambda M : (1 \rightarrow 1_2 \rightarrow o \rightarrow o) \lambda b : 1_2 \lambda c : o. \lambda f : 1 \lambda b : 1_2 \lambda c : o. M(f^+)(b^+)c,$$

where

$$\begin{aligned} f^+ &= \lambda t : o. b(\#f)t; \\ b^+ &= \lambda t_1, t_2 : o. b(\#b)(bt_1 t_2); \\ \#f &= bcc; \\ \#b &= bc(bcc). \end{aligned}$$

The terms $\#f, \#b$ serve as recognizers (“Gödel numbers”). Notice that M of type $1 \rightarrow 1_2 \rightarrow o \rightarrow o$ has a closed long $\beta\eta$ -nf of the form

$$M^{\text{nf}} \equiv \lambda f : 1 \lambda b : 1_2 \lambda c : o. t$$

with t an element of the set T generated by the grammar

$$T :: = c \mid fT \mid bT T.$$

Then for such M one has $\Phi M =_{\beta\eta} \Phi(M^{\text{nf}}) \equiv M^+$ with

$$M^+ \equiv \lambda f : 1 \lambda b : 1_2 \lambda c : o. t^+,$$

where t^+ is inductively defined by

$$\begin{aligned} c^+ &= c; \\ (ft)^+ &= b(\#f)t^+; \\ (bt_1 t_2)^+ &= b(\#b)(bt_1^+ t_2^+). \end{aligned}$$

It is clear that M^{nf} can be constructed back from M^+ . Therefore

$$\begin{aligned} \Phi M_1 =_{\beta\eta} \Phi M_2 &\Rightarrow M_1^+ =_{\beta\eta} M_2^+ \\ &\Rightarrow M_1^+ \equiv M_2^+ \\ &\Rightarrow M_1^{\text{nf}} \equiv M_2^{\text{nf}} \\ &\Rightarrow M_1 =_{\beta\eta} M_2. \blacksquare \end{aligned}$$

By the same method one can show that any type of rank ≤ 2 is reducible to \top , do exercise 3.6.12

Combining propositions 3.4.2-3.4.6 we can complete the proof of the reducibility theorem.

3.4.7. THEOREM (Reducibility theorem, Statman [1980a]). *Let*

$$\top = 1_2 \rightarrow o \rightarrow o.$$

Then

$$\forall A \in \mathbb{T} \ A \leq_{\beta\eta} \top.$$

PROOF. Let A be any type. Harvesting the results we obtain

$$\begin{array}{ll} A \leq_{\beta\eta} B, & \text{with } \text{rank}(B) \leq 3, \text{ by 3.4.2,} \\ \leq_{\beta\eta} 1_2 \rightarrow 1^2 \rightarrow 2 \rightarrow o, & \text{by 3.4.4,} \\ \leq_{\beta\eta} 2 \rightarrow 1_2 \rightarrow 1^2 \rightarrow o, & \text{by simply permuting arguments,} \\ \leq_{\beta\eta} 1^2 \rightarrow o \rightarrow 1_2 \rightarrow 1^2 \rightarrow o, & \text{by 3.4.5,} \\ \leq_{\beta\eta} 1_2 \rightarrow o \rightarrow o, & \text{by an other permutation and 3.4.6 } \blacksquare \end{array}$$

Now we turn attention to the type hierarchy, Statman [1980a].

3.4.8. DEFINITION. For the ordinals $\alpha \leq \omega + 3$ define the type $A_\alpha \in \mathbb{T}(\lambda_{\rightarrow}^o)$ as follows.

$$\begin{aligned} A_0 &= o; \\ A_1 &= o \rightarrow o; \\ &\dots \\ A_k &= o^k \rightarrow o; \\ &\dots \\ A_\omega &= 1 \rightarrow o \rightarrow o; \\ A_{\omega+1} &= 1 \rightarrow 1 \rightarrow o \rightarrow o; \\ A_{\omega+2} &= 3 \rightarrow o \rightarrow o; \\ A_{\omega+3} &= 1_2 \rightarrow o \rightarrow o. \end{aligned}$$

3.4.9. PROPOSITION. *For $\alpha, \beta \leq \omega + 3$ one has*

$$\alpha \leq \beta \Rightarrow A_\alpha \leq_{\beta\eta} A_\beta.$$

PROOF. For all finite k one has $A_k \leq_{\beta\eta} A_{k+1}$ via the map

$$\Phi_{k,k+1} \equiv \lambda m:A_k \lambda z x_1 \dots x_k : o.m x_1 \dots x_k =_{\beta\eta} \lambda m:A_k. \mathbb{K}m.$$

Moreover, $A_k \leq_{\beta\eta} A_\omega$ via

$$\Phi_{k,\omega} \equiv \lambda m:A_k \lambda f:1 \lambda x:o.m(\mathbf{c}_1 f x) \dots (\mathbf{c}_k f x).$$

Then $A_\omega \leq_{\beta\eta} A_{\omega+1}$ via

$$\Phi_{\omega,\omega+1} \equiv \lambda m:A_\omega \lambda f, g:1 \lambda x:o.m f x.$$

Now $A_{\omega+1} \leq_{\beta\eta} A_{\omega+2}$ via

$$\Phi_{\omega+1, \omega+2} \equiv \lambda m : A_{\omega+1} \lambda H : 3 \lambda x : o . H(\lambda f : 1 . H(\lambda g : 1 . m f g x)).$$

Finally, $A_{\omega+2} \leq_{\beta\eta} A_{\omega+3} = \top$ because of the reducibility theorem 3.4.7. See also exercise 4.1.9 for a concrete term $\Phi_{\omega+2, \omega+3}$. ■

3.4.10. PROPOSITION. For $\alpha, \beta \leq \omega + 3$ one has

$$\alpha \leq \beta \Leftarrow A_\alpha \leq_{\beta\eta} A_\beta.$$

PROOF. This will be proved in 3.5.32. ■

3.4.11. COROLLARY. For $\alpha, \beta \leq \omega + 3$ one has

$$A_\alpha \leq_{\beta\eta} A_\beta \iff \alpha \leq \beta.$$

For a proof that these types $\{A_\alpha\}_{\alpha \leq \omega+3}$ are a good representation of the reducibility classes we need some syntactic notions.

3.4.12. DEFINITION. A type $A \in \mathbb{T}(\lambda_{\rightarrow}^o)$ is called *large* if it has a negative subterm occurrence of the form $B_1 \rightarrow \dots \rightarrow B_n \rightarrow o$, with $n \geq 2$; A is *small* otherwise.

3.4.13. EXAMPLE. $1_2 \rightarrow o \rightarrow o$ is large; $(1_2 \rightarrow o) \rightarrow o$ and $3 \rightarrow o \rightarrow o$ are small.

Now we will partition the types $\mathbb{T} = \mathbb{T}(\lambda_{\rightarrow}^o)$ in the following classes.

3.4.14. DEFINITION. Define the following sets of types.

$$\begin{aligned} \mathbb{T}_{-1} &= \{A \mid A \text{ has no closed inhabitant}\}; \\ \mathbb{T}_0 &= \{o \rightarrow o\}; \\ \mathbb{T}_1 &= \{o^k \rightarrow o \mid k > 1\}; \\ \mathbb{T}_2 &= \{1 \rightarrow o^q \rightarrow o \mid q > 0\} \cup \{(1_p \rightarrow o) \rightarrow o^q \rightarrow o \mid p > 0, q \geq 0\}; \\ \mathbb{T}_3 &= \{A \mid A \text{ is small, } \text{rank}(A) \in \{2, 3\} \text{ and } A \notin \mathbb{T}_2\}; \\ \mathbb{T}_4 &= \{A \mid A \text{ is small and } \text{rank}(A) > 3\}; \\ \mathbb{T}_5 &= \{A \mid A \text{ is large}\}. \end{aligned}$$

It is clear that the \mathbb{T}_i form a partition of \mathbb{T} . A typical element of \mathbb{T}_{-1} is o . This class we will not consider much.

3.4.15. THEOREM (Hierarchy theorem, Statman [1980a]).

$$\begin{aligned} A \in \mathbb{T}_5 &\iff A \sim_{\beta\eta} 1_2 \rightarrow o \rightarrow o; \\ A \in \mathbb{T}_4 &\iff A \sim_{\beta\eta} 3 \rightarrow o \rightarrow o; \\ A \in \mathbb{T}_3 &\iff A \sim_{\beta\eta} 1 \rightarrow 1 \rightarrow o \rightarrow o; \\ A \in \mathbb{T}_2 &\iff A \sim_{\beta\eta} 1 \rightarrow o \rightarrow o; \\ A \in \mathbb{T}_1 &\iff A \sim_{\beta\eta} o^k \rightarrow o, & \text{for some } k > 1; \\ A \in \mathbb{T}_0 &\iff A \sim_{\beta\eta} o \rightarrow o; \\ A \in \mathbb{T}_{-1} &\iff A \sim_{\beta\eta} o. \end{aligned}$$

PROOF. Since the Π_i form a partition, it is sufficient to show just the \Rightarrow 's.

As to Π_5 , it is enough to show that $1_2 \rightarrow o \rightarrow o \leq_{\beta\eta} A$, for every large type A , since we know already the converse. For this see Statman [1980a], lemma 7. As a warmup exercise do 3.6.20.

As to Π_4 , it is shown in Statman [1980a], proposition 2, that if A is small, then $A \leq_{\beta\eta} 3 \rightarrow o \rightarrow o$. It remains to show that for any small type A of rank > 3 one has $3 \rightarrow o \rightarrow o \leq_{\beta\eta} A$. Do exercise 3.6.27.

As to Π_3 , the implication is shown in Statman [1980a], lemma 12. The condition about the type in that lemma is equivalent to belonging to Π_3 .

As to Π_2 , do exercise 3.6.22(ii).

As to Π_i , with $i = 1, 0, -1$, notice that $\Lambda^\emptyset(o^k \rightarrow o)$ contains exactly k closed terms for $k \geq 0$. This is sufficient. ■

For an application in the next section we need a refinement of the hierarchy theorem.

3.4.16. DEFINITION. Let A, B be types.

(i) A is *head-reducible* to B , notation $A \leq_h B$, iff for some closed term $\Phi \in \Lambda^\emptyset(A \rightarrow B)$ one has

$$\forall M_1, M_2 : A \ [M_1 =_{\beta\eta} M_2 \iff \Phi M_1 =_{\beta\eta} \Phi M_2],$$

and moreover Φ is of the form

$$\Phi = \lambda m : A \lambda x_1 \dots x_b : B . m P_1 \dots P_a, \quad (+)$$

with $m \notin \text{FV}(P_1, \dots, P_a)$.

(ii) A is *multi head-reducible* to B , notation $A \leq_{h+} B$, iff there are closed terms $\Phi_1, \dots, \Phi_m \in \Lambda^\emptyset(A \rightarrow B)$ each of the form (+) such that

$$\forall M_1, M_2 : A \ [M_1 =_{\beta\eta} M_2 \iff \Phi_1 M_1 =_{\beta\eta} \Phi_1 M_2 \ \& \dots \ \& \ \Phi_m M_1 =_{\beta\eta} \Phi_m M_2].$$

(iii) Write $A \sim_h B$ iff $A \leq_h B \leq_h A$ and similarly $A \sim_{h+} B$ iff $A \leq_{h+} B \leq_{h+} A$.

3.4.17. PROPOSITION. (i) $A \leq_h B \Rightarrow A \leq_{\beta\eta} B$.

(ii) Let $A, B \in \Pi_i$, with $i \neq 2$. Then $A \sim_h B$.

(iii) Let $A, B \in \Pi_2$. Then $A \sim_{h+} B$.

(iv) $A \leq_{\beta\eta} B \Rightarrow A \leq_{h+} B$.

PROOF. (i) Trivial.

(ii) Suppose $A \leq_{\beta\eta} B$. By inspection of the proof of the hierarchy theorem in all cases except for $A \in \Pi_2$ one obtains $A \leq_h B$. Do exercise 3.6.24.

(iii) In the exceptional case one obtains $A \leq_{h+} B$, see exercise 3.6.23. ■

(iv) By (ii) and (iii), using the hierarchy theorem.

3.4.18. COROLLARY (Hierarchy theorem (revisited), Statman [1980b]).

$$\begin{aligned}
A \in \Pi_5 &\iff A \sim_h 1_2 \rightarrow o \rightarrow o; \\
A \in \Pi_4 &\iff A \sim_h 3 \rightarrow o \rightarrow o; \\
A \in \Pi_3 &\iff A \sim_h 1 \rightarrow 1 \rightarrow o \rightarrow o; \\
A \in \Pi_2 &\iff A \sim_{h+} 1 \rightarrow o \rightarrow o; \\
A \in \Pi_1 &\iff A \sim_{h+} o^2 \rightarrow o; \\
A \in \Pi_0 &\iff A \sim_h o \rightarrow o; \\
A \in \Pi_{-1} &\iff A \sim_h o.
\end{aligned}$$

PROOF. The only extra fact to verify is that $o^k \rightarrow o \leq_{h+} o^2 \rightarrow o$. ■

Applications of the reducibility theorem

The reducibility theorem has several consequences.

3.4.19. DEFINITION. Let \mathcal{C} be a class of $\lambda \rightarrow$ models. \mathcal{C} is called *complete* iff

$$\forall M, N \in \Lambda^\emptyset[\mathcal{C} \models M = N \iff M =_{\beta\eta} N].$$

3.4.20. DEFINITION. (i) $\mathcal{T} = \mathcal{T}_{b,c}$ is the algebraic structure of trees inductively defined as follows.

$$\mathcal{T} = c \mid b \mathcal{T} \mathcal{T}$$

(ii) For a $\lambda \rightarrow$ model \mathcal{M} we say that \mathcal{T} can be embedded into \mathcal{M} , notation $\mathcal{T} \hookrightarrow \mathcal{M}$, iff there exist $b_0 \in \mathcal{M}(o \rightarrow o \rightarrow o)$, $c_0 \in \mathcal{M}(o)$ such that

$$\forall t, s \in \mathcal{T}[t \neq s \Rightarrow \mathcal{M} \models t^{\text{cl}} b_0 c_0 \neq s^{\text{cl}} b_0 c_0],$$

where $u^{\text{cl}} = \lambda b:o \rightarrow o \rightarrow o \lambda c:o.u$, is the closure of $u \in \mathcal{T}$.

The elements of \mathcal{T} are binary trees with c on the leaves and b on the connecting nodes. Typical examples are $c, bcc, bc(bcc)$ and $b(bcc)c$. The existence of an embedding using b_0, c_0 implies for example that $b_0 c_0(b_0 c_0 c_0), b_0 c_0 c_0$ and c_0 are mutually different in \mathcal{M} .

Note that $\mathcal{T} \not\hookrightarrow \mathcal{M}(2)$. To see this, write $gx = bxx$. One has $g^2(c) \neq g^4(c)$, but $\mathcal{M}(2) \models \forall g:o \rightarrow o \forall c:o. g^2(c) = g^4(c)$, do exercise 3.6.13.

3.4.21. LEMMA. (i) $\Pi_{i \in I} \mathcal{M}_i \models M = N \iff \forall i \in I. \mathcal{M}_i \models M = N$.

(ii) $M \in \Lambda^\emptyset(\top) \iff \exists s \in \mathcal{T}. M =_{\beta\eta} s^{\text{cl}}$.

PROOF. (i) Since $\llbracket M \rrbracket^{\Pi_{i \in I} \mathcal{M}_i} = \lambda i \in I. \llbracket M \rrbracket^{\mathcal{M}_i}$.

(ii) By an analysis of the possible shape of the normal forms of terms of type $\top = 1_2 \rightarrow o \rightarrow o$. ■

3.4.22. THEOREM (1-section theorem, Statman [1985]). \mathcal{C} is complete iff there is an (at most countable) family $\{\mathcal{M}_i\}_{i \in I}$ of structures in \mathcal{C} such that

$$\mathcal{T} \hookrightarrow \Pi_{i \in I} \mathcal{M}_i.$$

PROOF. (\Rightarrow) Suppose \mathcal{C} is complete. Let $t, s \in \mathcal{T}$. Then

$$\begin{aligned}
 t \neq s &\Rightarrow t^{\text{cl}} \neq_{\beta\eta} s^{\text{cl}} \\
 &\Rightarrow \mathcal{C} \not\models t^{\text{cl}} = s^{\text{cl}}, && \text{by completeness,} \\
 &\Rightarrow \mathcal{M}_{ts} \models t^{\text{cl}} \neq s^{\text{cl}}, && \text{for some } \mathcal{M}_{st} \in \mathcal{C}, \\
 &\Rightarrow \mathcal{M}_{ts} \models t^{\text{cl}} b_{ts} c_{ts} \neq s^{\text{cl}} b_{ts} c_{ts},
 \end{aligned}$$

for some $b_{ts} \in \mathcal{M}(o \rightarrow o \rightarrow o)$, $c_{ts} \in \mathcal{M}(o)$ by extensionality. Note that in the third implication the axiom of (countable) choice is used.

It now follows by lemma 3.4.21(i) that

$$\prod_{t \neq s} \mathcal{M}_{ts} \models t^{\text{cl}} \neq s^{\text{cl}},$$

since they differ on the pair $b_0 c_0$ with $b_0(ts) = b_{ts}$ and similarly for c_0 .

(\Leftarrow) Suppose $\mathcal{T} \hookrightarrow \prod_{i \in I} \mathcal{M}_i$ with $\mathcal{M}_i \in \mathcal{C}$. Let M, N be closed terms of some type A . By soundness one has

$$M =_{\beta\eta} N \Rightarrow \mathcal{C} \models M = N.$$

For the converse, let by the reducibility theorem $F : A \rightarrow \top$ be such that

$$M =_{\beta\eta} N \iff FM =_{\beta\eta} FN,$$

for all $M, N \in \Lambda^\emptyset$. Then

$$\begin{aligned}
 \mathcal{C} \models M = N &\Rightarrow \prod_{i \in I} \mathcal{M}_i \models M = N, && \text{by the lemma,} \\
 &\Rightarrow \prod_{i \in I} \mathcal{M}_i \models FM = FN, \\
 &\Rightarrow \prod_{i \in I} \mathcal{M}_i \models t^{\text{cl}} = s^{\text{cl}},
 \end{aligned}$$

where t, s are such that

$$FM =_{\beta\eta} t^{\text{cl}}, FN =_{\beta\eta} s^{\text{cl}}, \tag{*}$$

noting that every closed term of type \top is $\beta\eta$ -convertible to some u^{cl} with $u \in \mathcal{T}$. Now the chain of arguments continues as follows

$$\begin{aligned}
 &\Rightarrow t \equiv s, && \text{by the embedding property,} \\
 &\Rightarrow FM =_{\beta\eta} FN, && \text{by (*),} \\
 &\Rightarrow M =_{\beta\eta} N, && \text{by reducibility. } \blacksquare
 \end{aligned}$$

3.4.23. COROLLARY. (i) [Friedman [1975]] $\{\mathcal{M}_{\mathbb{N}}\}$ is complete.

(ii) [Plotkin [1985?]] $\{\mathcal{M}_n\}_{n \in \mathbb{N}}$ is complete.

(iii) $\{\mathcal{M}_{\mathbb{N}_\perp}\}$ is complete.

(iv) $\{\mathcal{M}_D\}_{D \text{ a finite cpo,}}$ is complete.

PROOF. Immediate from the theorem. \blacksquare

The completeness of the collection $\{\mathcal{M}_n\}_{n \in \mathbb{N}}$ essentially states that for every pair of terms M, N of a given type A there is a number $n = n_{M,N}$ such that $\mathcal{M}_n \models M = N \Rightarrow M =_{\beta\eta} N$. Actually one can do better, by showing that n only depends on M .

3.4.24. PROPOSITION (Finite completeness theorem, Statman [1982]). *For every type $A \in \mathbb{T}(\lambda^o_>)$ and every closed term M of type A there is a number $n = n_M$ such that for all closed terms N of type A one has*

$$\mathcal{M}_n \models M = N \iff M =_{\beta\eta} N.$$

PROOF. By the reduction theorem 3.4.7 it suffices to show this for $A = \top$. Let M a closed term of type \top be given. Each closed term N of type \top has as long $\beta\eta$ -nf

$$N = \lambda b:1_2 \lambda c:o.s_N,$$

where $s_N \in \mathcal{T}$. Let $\mathbf{p} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ be an injective pairing on the integers such that $\mathbf{p}(k_1, k_2) > k_i$. Take

$$n_M = \llbracket M \rrbracket^{\mathcal{M}_\omega} \mathbf{p} 0 + 1.$$

Define $\mathbf{p}' : X_{n+1}^2 \rightarrow X_{n+1}$, where $X_{n+1} = \{0, \dots, n+1\}$, by

$$\begin{aligned} \mathbf{p}'(k_1, k_2) &= \mathbf{p}(k_1, k_2), & \text{if } k_1, k_2 \leq n\mathbf{p}(k_1, k_2) \leq n; \\ &= n+1 & \text{else.} \end{aligned}$$

Suppose $\mathcal{M}_n \models M = N$. Then $\llbracket M \rrbracket^{\mathcal{M}_n} \mathbf{p}' 0 = \llbracket N \rrbracket^{\mathcal{M}_n} \mathbf{p}' 0$. By the choice of n it follows that $\llbracket M \rrbracket^{\mathcal{M}_n} \mathbf{p} 0 = \llbracket N \rrbracket^{\mathcal{M}_n} \mathbf{p} 0$ and hence $s_M = s_N$. Therefore $M =_{\beta\eta} N$. ■

3.4.25. DEFINITION (Reducibility Hierarchy, Statman [1980a]). For the ordinals $\alpha \leq \omega + 3$ define the type $A_\alpha \in \mathbb{T}(\lambda^o_>)$ as follows.

$$\begin{aligned} A_0 &= o; \\ A_1 &= o \rightarrow o; \\ &\dots \\ A_k &= o^k \rightarrow o; \\ &\dots \\ A_\omega &= 1 \rightarrow o \rightarrow o; \\ A_{\omega+1} &= 1 \rightarrow 1 \rightarrow o; \\ A_{\omega+2} &= 3 \rightarrow o \rightarrow o; \\ A_{\omega+3} &= 1_2 \rightarrow o \rightarrow o. \end{aligned}$$

3.4.26. PROPOSITION. *For $\alpha, \beta \leq \omega + 3$ one has*

$$\alpha \leq \beta \Rightarrow A_\alpha \leq_{\beta\eta} A_\beta.$$

PROOF. For all finite k one has $A_k \leq_{\beta\eta} A_{k+1}$ via the map

$$\Phi_{k,k+1} \equiv \lambda m:A_k \lambda z x_1 \dots x_k : o. m x_1 \dots x_k =_{\beta\eta} \lambda m:A_k. \mathsf{K}m.$$

Moreover, $A_k \leq_{\beta\eta} A_\omega$ via

$$\Phi_{k,\omega} \equiv \lambda m:A_k \lambda f:1 \lambda x:o. m(\mathsf{c}_1 f x) \dots (\mathsf{c}_k f x).$$

Then $A_\omega \leq_{\beta\eta} A_{\omega+1}$ via

$$\Phi_{\omega,\omega+1} \equiv \lambda m:A_\omega \lambda f, g:1 \lambda x:o. m f x.$$

Now $A_{\omega+1} \leq_{\beta\eta} A_{\omega+2}$ via

$$\Phi_{\omega+1,\omega+2} \equiv \lambda m:A_{\omega+1} \lambda H:3 \lambda x:o. H(\lambda f:1. H(\lambda g:1. m f g x)).$$

Finally, $A_{\omega+2} \leq_{\beta\eta} A_{\omega+3} = \top$ because of the reducibility theorem 3.4.7. See also exercise 4.1.9 for a concrete term $\Phi_{\omega+2,\omega+3}$. ■

3.4.27. PROPOSITION. For $\alpha, \beta \leq \omega + 3$ one has

$$\alpha \leq \beta \Leftarrow A_\alpha \leq_{\beta\eta} A_\beta.$$

PROOF. This will be proved in 3.5.32. ■

3.4.28. COROLLARY. For $\alpha, \beta \leq \omega + 3$ one has

$$A_\alpha \leq_{\beta\eta} A_\beta \iff \alpha \leq \beta.$$

3.5. The five canonical term-models

The open terms of λ_{\rightarrow}^o form an extensional model, the term-model \mathcal{M}_{Λ_o} . One may wonder whether there are also closed term-models, like in the untyped lambda calculus. If no constants are present, then this is not the case, since there are e.g. no closed terms of ground type o . In the presence of constants matters change. We will first show how a set of constants \mathcal{D} gives rise to an extensional equivalence relation on $\Lambda_o^\emptyset[\mathcal{D}]$, the set of closed terms with constants from \mathcal{D} . Then we define canonical sets of constants and prove that for these the resulting equivalence relation is also a congruence, i.e. determines a term-model. After that it will be shown that for all sets \mathcal{D} of constants with enough closed terms the extensional equivalence determines a term-model. Up to elementary equivalence (satisfying the same set of equations between closed pure terms, i.e. closed terms without any constants) all models, for which the equality on type o coincides with $=_{\beta\eta}$, can be obtained in this way. From now on \mathcal{D} will range over sets of constants such that there are closed terms for every type A (i.e. in $\Lambda_o^\emptyset[\mathcal{D}](A)$).

3.5.1. DEFINITION. Let $M, N \in \Lambda_o^\emptyset[\mathcal{D}](A)$ with $A = A_1 \rightarrow \dots \rightarrow A_a \rightarrow o$.

(i) M is \mathcal{D} -extensionally equivalent with N , notation $M \approx_{\mathcal{D}}^{\text{ext}} N$, iff

$$\forall t_1 \in \Lambda_o^\emptyset[\mathcal{D}](A_1) \dots t_a \in \Lambda_o^\emptyset[\mathcal{D}](A_a). M\vec{t} =_{\beta\eta} N\vec{t}.$$

[If $a = 0$, then $M, N \in \Lambda_o^\emptyset[\mathcal{D}](o)$; in this case $M \approx_{\mathcal{D}}^{\text{ext}} N \iff M =_{\beta\eta} N$.]

(ii) M is \mathcal{D} -observationally equivalent with N , notation $M \approx_{\mathcal{D}}^{\text{obs}} N$, iff

$$\forall F \in \Lambda_o^\emptyset[\mathcal{D}](A \rightarrow o) FM =_{\beta\eta} FN. \blacksquare$$

3.5.2. REMARK. Note that if $M \approx_{\mathcal{D}}^{\text{obs}} N$, then for all $F \in \Lambda_o^\emptyset[\mathcal{D}](A \rightarrow B)$ one has $FM \approx_{\mathcal{D}}^{\text{obs}} FN$. Similarly, if $M \approx_{\mathcal{D}}^{\text{ext}} N$ of type $A \rightarrow B$, then for all $Z \in \Lambda_o^\emptyset[\mathcal{D}](A)$ one has $MZ \approx_{\mathcal{D}}^{\text{ext}} NZ$ of type B .

We will show that for all \mathcal{D} and $M, N \in \Lambda_o^\emptyset[\mathcal{D}]$ one has

$$M \approx_{\mathcal{D}}^{\text{ext}} N \iff M \approx_{\mathcal{D}}^{\text{obs}} N.$$

Therefore, as soon as this is proved, we can write simply $M \approx_{\mathcal{D}} N$.

Note that in the definition of extensional equivalence the \vec{t} range over closed terms (containing possibly constants). So this notion is not the same as $\beta\eta$ -convertibility: M and N may act differently on different variables, even if they act the same on all those closed terms. The relation $\approx_{\mathcal{D}}^{\text{ext}}$ is related to what is called in the untyped calculus the ω -rule, see Barendregt [1984], §17.3.

The intuition behind observational equivalence is that for M, N of higher type A one cannot ‘see’ that they are equal, unlike for terms of type o . But one can do ‘experiments’ with M and N , the outcome of which is observational, i.e. of type o , by putting these terms in a context $C[-]$ resulting in two terms of type o . For closed terms it amounts to the same to consider just FM and FN for all $F \in \Lambda_o^\emptyset[\mathcal{D}](A \rightarrow o)$.

3.5.3. LEMMA. Let S be an n -ary logical relation on $\Lambda_o^\emptyset[\mathcal{D}]$, that is closed under $=_{\beta\eta}$. If $S(\mathbf{d}, \dots, \mathbf{d})$ holds for all $\mathbf{d} \in \mathcal{D}$, then

$$S(M, \dots, M)$$

holds for all $M \in \Lambda_o^\emptyset[\mathcal{D}]$.

PROOF. Let $\mathcal{D} = \{\mathbf{d}_1^{A_1}, \dots, \mathbf{d}_n^{A_n}\}$. M can be written as

$$M \equiv M[\vec{\mathbf{d}}] =_{\beta\eta} (\lambda \vec{x}. M[\vec{x}])\vec{\mathbf{d}} \equiv M^+ \vec{\mathbf{d}},$$

with M^+ a closed and pure term (i.e. without variables or constants). Then

$$\begin{aligned} & S(M^+, \dots, M^+), && \text{by the fundamental theorem} \\ & && \text{for logical relations} \\ \Rightarrow & S(M^+ \vec{\mathbf{d}}, \dots, M^+ \vec{\mathbf{d}}), && \text{since } S \text{ is logical and } \forall \mathbf{d} \in \mathcal{D}. S(\vec{\mathbf{d}}), \\ \Rightarrow & S(M, \dots, M), && \text{since } S \text{ is } =_{\beta\eta} \text{ closed. } \blacksquare \end{aligned}$$

3.5.4. LEMMA. Let $S = S^{\mathcal{D}} = \{S_A\}_{A \in \mathbb{T}(\lambda_{\perp}^o)}$ be the logical relation on $\Lambda_o^{\emptyset}[\mathcal{D}](o)$ determined by

$$S_o(M, N) \iff M =_{\beta\eta} N,$$

for $M, N \in \Lambda_o^{\emptyset}[\mathcal{D}](o)$.

- (i) Suppose that for all $\mathbf{d} \in \mathcal{D}$ one has $S(\mathbf{d}, \mathbf{d})$. Then $\approx_{\mathcal{D}}^{\text{ext}}$ is logical.
- (ii) Let $\mathbf{d} \in \mathcal{D}$ be a constant of type $A \rightarrow o$ with $A = A_1 \rightarrow \dots \rightarrow A_m \rightarrow o$. Suppose
 - $\forall F, G \in \Lambda_o^{\emptyset}[\mathcal{D}](A) [F \approx_{\mathcal{D}}^{\text{ext}} G \Rightarrow F =_{\beta\eta} G]$;
 - $\forall t_i \in \Lambda_o^{\emptyset}[\mathcal{D}](A_i) S(t_i, t_i), 1 \leq i \leq m$.

Then $S(\mathbf{d}, \mathbf{d})$.

PROOF. (i) By the assumption and the fact that S is $=_{\beta\eta}$ closed (since S_o is), lemma 3.5.3 implies that

$$S(M, M) \tag{0}$$

for all $M \in \Lambda_o^{\emptyset}[\mathcal{D}]$. Hence S is an equivalence relation on $\Lambda_o^{\emptyset}[\mathcal{D}]$. Claim.

$$S_A(F, G) \iff F \approx_{\mathcal{D}}^{\text{ext}} G,$$

for all $F, G \in \Lambda_o^{\emptyset}[\mathcal{D}](A)$. This is proved by induction on the structure of A . If $A = o$, then this is trivial. If $A = B \rightarrow C$, then we proceed as follows.

$$\begin{aligned}
 (\Rightarrow) \quad S_{B \rightarrow C}(F, G) &\Rightarrow S_C(Ft, Gt), \text{ for all } t \in \Lambda_o^{\emptyset}[\mathcal{D}](B). S_B(t, t) \\
 &\quad \text{by the IH, since } t \approx_{\mathcal{D}}^{\text{ext}} t \text{ and hence } S_B(t, t), \\
 &\Rightarrow Ft \approx_{\mathcal{D}}^{\text{ext}} Gt, \text{ for all } t \in \Lambda_o^{\emptyset}[\mathcal{D}], \text{ by the IH,} \\
 &\Rightarrow F \approx_{\mathcal{D}}^{\text{ext}} G, \text{ by definition.} \\
 (\Leftarrow) \quad F \approx_{\mathcal{D}}^{\text{ext}} G &\Rightarrow Ft \approx_{\mathcal{D}}^{\text{ext}} Gt, \text{ for all } t \in \Lambda_o^{\emptyset}[\mathcal{D}], \\
 &\Rightarrow S_C(Ft, Gt)
 \end{aligned} \tag{1}$$

by the induction hypothesis. Now in order to prove $S_{B \rightarrow C}(F, G)$, assume $S_B(t, s)$ trying to show $S_C(Ft, Gs)$. Well, since also $S_{B \rightarrow C}(G, G)$, by (0), we have

$$S_C(Gt, Gs). \tag{2}$$

It follows from (1) and (2) and the transitivity of S (being by the IH on this type the same as $\approx_{\mathcal{D}}^{\text{ext}}$) that $S_C(Ft, Gs)$ indeed.

So we have proved the claim. Since $\approx_{\mathcal{D}}^{\text{ext}}$ is S and S is logical, it follows that $\approx_{\mathcal{D}}^{\text{ext}}$ is logical.

(ii) Let \mathbf{d} be given. Then

$$\begin{aligned}
 S(F, G) &\Rightarrow F\vec{t} =_{\beta\eta} G\vec{t}, \quad \text{since } \forall \vec{t} \in \Lambda_o^{\emptyset}[\mathcal{D}] S(t_i, t_i), \\
 &\Rightarrow F \approx_{\mathcal{D}}^{\text{ext}} G, \\
 &\Rightarrow F =_{\beta\eta} G, \quad \text{by assumption,} \\
 &\Rightarrow \mathbf{d}F =_{\beta\eta} \mathbf{d}G.
 \end{aligned}$$

Therefore we have by definition $S(\mathbf{d}, \mathbf{d})$. ■

3.5.5. LEMMA. Suppose that $\approx_{\mathcal{D}}^{\text{ext}}$ is logical. Then

$$\forall M, N \in \Lambda_o^\emptyset[\mathcal{D}] [M \approx_{\mathcal{D}}^{\text{ext}} N \iff M \approx_{\mathcal{D}}^{\text{obs}} N].$$

PROOF. (\Leftarrow) Assume $M \approx_{\mathcal{D}}^{\text{obs}} N$. Then $M \approx_{\mathcal{D}}^{\text{ext}} N$ by taking $F \equiv \lambda m:B.m\vec{t}$.

(\Rightarrow) Assume $M \approx_{\mathcal{D}}^{\text{ext}} N$. Let $F \in \Lambda_o^\emptyset[\mathcal{D}](A \rightarrow o)$. Then and

$$\begin{aligned} F &\approx_{\mathcal{D}}^{\text{ext}} F, \\ \Rightarrow FM &\approx_{\mathcal{D}}^{\text{ext}} FN, \text{ as by assumption } \approx_{\mathcal{D}}^{\text{ext}} \text{ is logical,} \\ \Rightarrow FM &=_{\beta\eta} FN, \text{ because the type is } o. \end{aligned}$$

Therefore $M \approx_{\mathcal{D}}^{\text{obs}} N$. ■

In order to show that for arbitrary \mathcal{D} extensional equivalence is the same as observational equivalence first this will be done for the following six sets of constants.

3.5.6. DEFINITION. The following sets of constants will play a crucial role in this section.

$$\begin{aligned} \mathcal{C}_0 &= \{\mathbf{c}^o\}; \\ \mathcal{C}_1 &= \{\mathbf{c}^o, \mathbf{d}^o\}; \\ \mathcal{C}_2 &= \{\mathbf{f}^1, \mathbf{c}^o\}; \\ \mathcal{C}_3 &= \{\mathbf{f}^1, \mathbf{g}^1, \mathbf{c}^o\}; \\ \mathcal{C}_4 &= \{\mathbf{\Phi}^3, \mathbf{c}^o\}; \\ \mathcal{C}_5 &= \{\mathbf{b}^{1^2}, \mathbf{c}^o\}. \blacksquare \end{aligned}$$

From now on in this section \mathcal{C} ranges over the canonical sets of constants $\{\mathcal{C}_0, \dots, \mathcal{C}_5\}$ and \mathcal{D} over an arbitrary set of constants.

3.5.7. DEFINITION. (i) We say that a type $A = A_1 \rightarrow \dots \rightarrow A_a \rightarrow o$ is *represented in \mathcal{D}* iff there are distinct constants \mathbf{d}_i of type A_i in \mathcal{D} .

(ii) Let \mathcal{C} be one of the canonical sets of constants. Define the *characteristic type* of \mathcal{C} , notation $\nabla(\mathcal{C})$, as follows.

$$\begin{aligned} \nabla(\mathcal{C}_0) &= o \rightarrow o; \\ \nabla(\mathcal{C}_1) &= o \rightarrow o \rightarrow o; \\ \nabla(\mathcal{C}_2) &= 1 \rightarrow o \rightarrow o; \\ \nabla(\mathcal{C}_3) &= 1 \rightarrow 1 \rightarrow o \rightarrow o; \\ \nabla(\mathcal{C}_4) &= 3 \rightarrow o \rightarrow o; \\ \nabla(\mathcal{C}_5) &= 1^2 \rightarrow o \rightarrow o. \end{aligned}$$

In other words, $\nabla(\mathcal{C}_i)$ is intuitively type of $\lambda \vec{\mathbf{c}}_i. \mathbf{c}^o$, where $\mathcal{C}_i = \{\vec{\mathbf{c}}_i\}$ (the order of the abstractions is immaterial). Note that $\nabla(\mathcal{C}_i)$ is represented in \mathcal{C}_i . Also one has

$$i \leq j \iff \nabla(\mathcal{C}_i) \leq_{\beta\eta} \nabla(\mathcal{C}_j),$$

as follows from the theory of type reducibility.

(iii) The *class* of \mathcal{D} is

$$\max\{i \mid \nabla(\mathcal{C}_i) \leq_{\beta\eta} D \text{ for some type } D \text{ represented in } \mathcal{D}\}.$$

(iv) The *characteristic type* of \mathcal{D} is $\nabla(\mathcal{C}_i)$, where i is the class of \mathcal{D} .

3.5.8. LEMMA. *Let A be represented in \mathcal{D} . Then for $M, N \in \Lambda_o^\emptyset(A)$ pure closed terms of type A , one has*

$$M \approx_{\mathcal{D}}^{\text{ext}} N \iff M =_{\beta\eta} N.$$

PROOF. The (\Leftarrow) direction is trivial. As to (\Rightarrow)

$$\begin{aligned} M \approx_{\mathcal{D}}^{\text{ext}} N &\iff \forall \vec{T} \in \Lambda_o^\emptyset[\mathcal{D}]. M\vec{T} =_{\beta\eta} N\vec{T} \\ &\Rightarrow M\vec{d} =_{\beta\eta} N\vec{d}, \text{ for some } \vec{d} \in \mathcal{D} \text{ since} \\ &\quad A \text{ is represented in } \mathcal{D}, \\ &\Rightarrow M\vec{x} =_{\beta\eta} N\vec{x}, \text{ since a constant may be replaced} \\ &\quad \text{by a variable if } M, N \text{ are pure,} \\ &\Rightarrow M =_{\eta} \lambda \vec{x}. M\vec{x} =_{\beta\eta} \lambda \vec{x}. N\vec{x} =_{\eta} N. \blacksquare \end{aligned}$$

We will need the following combinatorial lemma about $\approx_{\mathcal{C}_4}$.

3.5.9. LEMMA. *For every $F, G \in \Lambda[\mathcal{C}_4](2)$ one has*

$$F \approx_{\mathcal{C}_4} G \Rightarrow F =_{\beta\eta} G.$$

PROOF. We must show

$$[\forall h \in \Lambda[\mathcal{C}_4](1). Fh =_{\beta\eta} Gh] \Rightarrow F =_{\beta\eta} G. \quad (1)$$

In order to do this, a classification has to be given for the elements of $\Lambda[\mathcal{C}_4](2)$. Define for $A \in \mathbb{T}(\lambda_{\rightarrow}^o)$

$$A_{\Delta} = \{M \in \Lambda[\mathcal{C}_4](A) \mid \Delta \vdash M : A \text{ \& } M \text{ in } \beta\eta\text{-nf}\}.$$

It is easy to show that o_{Δ} and 2_{Δ} are generated by the following ‘two-level’ grammar, see van Wijngaarden et al. [1976].

$$\begin{aligned} 2_{\Delta} &= \lambda f : 1.o_{\Delta}, f : 1 \\ o_{\Delta} &= \mathbf{c} \mid \Phi 2_{\Delta} \mid \Delta.1 o_{\Delta}, \end{aligned}$$

where $\Delta.A$ consists of $\{v \mid v^A \in \Delta\}$.

It follows that a typical element of 2_{\emptyset} is

$$\lambda f_1 : 1. \Phi(\lambda f_2 : 1. f_1(f_2(\Phi(\lambda f_3 : 1. f_3(f_2(f_1(f_3 \mathbf{c}))))))).$$

Hence a general element can be represented by a list of words

$$\langle w_1, \dots, w_n \rangle,$$

with $w_i \in \Sigma_i^*$ and $\Sigma_i = \{f_1, \dots, f_i\}$, the representation of the typical element above being $\langle \epsilon, f_1 f_2, f_3 f_2 f_1 f_3 \rangle$.

Let $h_m = \lambda z^o. \Phi(\lambda g:1.g^m(z))$; then $h_n \in 1_\emptyset$. We claim that for some m we have

$$Fh_m =_{\beta\eta} Gh_m \Rightarrow F =_{\beta\eta} G.$$

For a given $F \in \Lambda[\mathcal{C}_4](2)$ one can find a representation of the $\beta\eta$ -nf of Fh_m from the representation of the $\beta\eta$ -nf $F^{\text{nf}} \in 2_\emptyset$ of F . It will turn out that if m is large enough, then F^{nf} can be determined (‘read back’) from the $\beta\eta$ -nf of Fh_m .

In order to see this, let F^{nf} be represented by the list of words $\langle w_1, \dots, w_n \rangle$, as above. The occurrences of f_1 can be made explicit and we write

$$w_i = w_{i0} f_1 w_{i1} f_1 w_{i2} \dots f_1 w_{ik_i}.$$

Some of the w_{ij} will be empty (in any case the w_{1j}) and $w_{ij} \in \Sigma_i^{-*}$ with $\Sigma_i^- = \{f_2, \dots, f_i\}$. Then F^{nf} can be written as (using for application—contrary to the usual convention—association to the right)

$$\begin{aligned} F^{\text{nf}} &\equiv \lambda f_1. w_{10} f_1 w_{11} \dots f_1 w_{1k_1} \\ &\quad \Phi(\lambda f_2. w_{20} f_1 w_{21} \dots f_1 w_{2k_2} \\ &\quad \dots \\ &\quad \Phi(\lambda f_n. w_{n0} f_1 w_{n1} \dots f_1 w_{nk_n} \\ &\quad c)..). \end{aligned}$$

Now we have

$$\begin{aligned}
(Fh_m)^{\text{nf}} &\equiv w_{10} \\
&\Phi(\lambda g.g^m w_{11}) \\
&\dots \\
&\Phi(\lambda g.g^m w_{1k_1}) \\
&\Phi(\lambda f_2.w_{20}) \\
&\Phi(\lambda g.g^m w_{21}) \\
&\dots \\
&\Phi(\lambda g.g^m w_{2k_2}) \\
&\Phi(\lambda f_3.w_{30}) \\
&\Phi(\lambda g.g^m w_{31}) \\
&\dots \\
&\Phi(\lambda g.g^m w_{3k_3}) \\
&\dots \\
&\dots \\
&\Phi(\lambda f_n.w_{n0}) \\
&\Phi(\lambda g.g^m w_{n1}) \\
&\dots \\
&\Phi(\lambda g.g^m w_{nk_n}) \\
&c)..))..))..))..)).
\end{aligned}$$

So if $m > \max_{ij} \{\text{length}(w_{ij})\}$ we can read back the w_{ij} and hence F^{nf} from $(Fh_m)^{\text{nf}}$. Therefore using an m large enough (1) can be shown as follows:

$$\begin{aligned}
\forall h \in \Lambda[\mathcal{C}_4](1). Fh =_{\beta\eta} Gh &\Rightarrow Fh_m =_{\beta\eta} Gh_m \\
&\Rightarrow (Fh_m)^{\text{nf}} \equiv (Gh_m)^{\text{nf}} \\
&\Rightarrow F^{\text{nf}} \equiv G^{\text{nf}} \\
&\Rightarrow F =_{\beta\eta} F^{\text{nf}} \equiv G^{\text{nf}} =_{\beta\eta} G. \blacksquare
\end{aligned}$$

3.5.10. PROPOSITION. *The relations $\approx_{\mathcal{C}_i}$ are logical, for $1 \leq i \leq 5$.*

PROOF. Let S be the logical relation determined by $=_{\beta\eta}$ on type o . By lemma 3.5.4 (i) we have to check $S(\mathbf{c}, \mathbf{c})$ for all constants \mathbf{c} in \mathcal{C}_i . For $i \neq 4$ this is easy (trivial for constants of type o and almost trivial for the ones of type 1 and 1^2 ; in fact for all terms $h \in \Lambda_o^\emptyset[\mathcal{C}]$ of these types one has $S(h, h)$).

For $i = 4$ it suffices by lemma 3.5.4 (ii) to show that

$$F \approx_{\mathcal{C}_4} G \Rightarrow F =_{\beta\eta} G$$

for all $F, G \in \Lambda_o^\emptyset[\mathcal{C}_4](2)$. This has been done in lemma 3.5.9. \blacksquare

It follows that for the canonical sets \mathcal{C} observational equivalence is the same as extensional equivalence.

3.5.11. THEOREM. *Let \mathcal{C} be one of the canonical classes of constants. Then*

$$\forall M, N \in \Lambda_o^\emptyset[\mathcal{C}] [M \approx_{\mathcal{C}}^{\text{obs}} N \iff M \approx_{\mathcal{C}}^{\text{ext}} N].$$

PROOF. By the proposition and lemma 3.5.5.

3.5.12. DEFINITION. Let \mathcal{D} be an arbitrary set of constants. Define

$$\mathcal{M}_{\mathcal{D}} = \Lambda_o^\emptyset[\mathcal{D}] / \approx_{\mathcal{D}}^{\text{ext}},$$

with application defined by

$$[F]_{\mathcal{D}}[M]_{\mathcal{D}} = [FM]_{\mathcal{D}}.$$

Here $[-]_{\mathcal{D}}$ denotes an equivalence class modulo $\approx_{\mathcal{D}}$.

3.5.13. THEOREM. *Let \mathcal{C} be one of the canonical sets of constants.*

- (i) *Application in $\mathcal{M}_{\mathcal{C}}$ is a well-defined.*
- (ii) *For all $M, N \in \Lambda_o^\emptyset[\mathcal{C}]$ one has*

$$\mathcal{M}_{\mathcal{C}} \models M = N \iff M \approx_{\mathcal{C}} N.$$

- (iii) *$\mathcal{M}_{\mathcal{C}}$ is an extensional term-model.*

PROOF. (i) Using theorem 3.5.11 one can show that the definition of the application operator is independent of the choice of representatives:

$$\begin{array}{ll} F \approx_{\mathcal{C}} F' \ \& \ M \approx_{\mathcal{C}} M' & \Rightarrow \\ F \approx_{\mathcal{C}}^{\text{ext}} F' \ \& \ M \approx_{\mathcal{C}}^{\text{obs}} M' & \Rightarrow \\ FM \approx_{\mathcal{C}}^{\text{ext}} F'M \approx_{\mathcal{C}}^{\text{obs}} F'M' & \Rightarrow \\ FM \approx_{\mathcal{C}} F'M'. & \blacksquare \end{array}$$

- (ii) Show by induction on M that

$$\llbracket M \rrbracket_{\rho} = [M[\vec{x} := \rho(x_1), \dots, \rho(x_n)]],$$

the $\approx_{\mathcal{C}}$ equivalence class, satisfies the semantic requirements.

- (iii) Use (ii) and the fact that $\approx_{\mathcal{C}}^{\text{ext}}$ the model is extensional. \blacksquare

3.5.14. DEFINITION. (i) If \mathcal{M} is a model of $\lambda_{\rightarrow}^o[\mathcal{C}]$, then for a type A its A -section is simply \mathcal{M}_A .

(ii) We say that \mathcal{M} is A -complete (respectively A -complete for pure terms) iff for all closed terms (respectively pure closed terms) M, N of type A one has

$$\mathcal{M} \models M = N \iff M =_{\beta_{\eta}} N.$$

(iii) \mathcal{M} is *complete (for pure terms)* if for all types $A \in \Lambda_o$ it is A -complete (for pure terms).

(iv) A model \mathcal{M} is called *fully abstract* if observational equivalence is the same as equality in \mathcal{M} .

Using this terminology lemma 3.5.8 states that if A is represented in \mathcal{C} , then $\mathcal{M}_{\mathcal{C}}$ is A -complete for pure terms.

3.5.15. COROLLARY. *Let \mathcal{C} one of the canonical classes of constants and let A be its characteristic type. Then $\mathcal{M}_{\mathcal{C}}$ has the following properties.*

- (i) $\mathcal{M}_{\mathcal{C}}$ is an extensional term-model.
- (ii) $\mathcal{M}_{\mathcal{C}}$ is fully abstract.
- (iii) $\mathcal{M}_{\mathcal{C}}$ is 0 -complete.
- (iv) $\mathcal{M}_{\mathcal{C}}$ is $\nabla(\mathcal{C})$ -complete for pure terms.

PROOF. (i) By theorem 3.5.11 the definition of application is well-defined. That extensionality holds follows from the definition of $\approx_{\mathcal{D}}$. Because all combinators $[K_{AB}]_{\mathcal{C}}, [S_{ABC}]_{\mathcal{C}}$ are in $\mathcal{M}_{\mathcal{C}}$ the structure is a model.

- (ii) Again by theorem 3.5.11 $\mathcal{M}_{\mathcal{C}}$ is fully abstract.
- (iii) Since on type o the relation $\approx_{\mathcal{C}}$ is just $=_{\beta\eta}$, the model is o -complete.
- (iv) By lemma 3.5.8. ■

3.5.16. PROPOSITION. (i) *Let $i \leq j$. Then for pure closed terms $M, N \in \Lambda_o^{\emptyset}$*

$$\mathcal{M}_{\mathcal{C}_j} \models M = N \Rightarrow \mathcal{M}_{\mathcal{C}_i} \models M = N.$$

(ii) $\text{Th}(\mathcal{M}_{\mathcal{C}_5}) \subseteq \dots \subseteq \text{Th}(\mathcal{M}_{\mathcal{C}_1})$.

PROOF. (i) $\mathcal{M}_{\mathcal{C}_i} \not\models M = N \Rightarrow M \not\approx_{\mathcal{C}_i} N$
 $\Rightarrow M\vec{t}[\vec{d}] \neq_{\beta\eta} N\vec{t}[\vec{d}]$, for some $\vec{t}[\vec{d}]$,
 $\Rightarrow \lambda\vec{d}.M\vec{t}[\vec{d}] \neq_{\beta\eta} \lambda\vec{d}.N\vec{t}[\vec{d}]$
 $\Rightarrow \Psi(\lambda\vec{d}.M\vec{t}[\vec{d}]) \neq_{\beta\eta} \Psi(\lambda\vec{d}.N\vec{t}[\vec{d}])$,
 since $\nabla(\mathcal{C}_i) \leq_{\beta\eta} \nabla(\mathcal{C}_j)$ via some Ψ ,
 $\Rightarrow \Psi(\lambda\vec{d}.M\vec{t}[\vec{d}]) \not\approx_{\mathcal{C}_j} \Psi(\lambda\vec{d}.N\vec{t}[\vec{d}])$
 $\Rightarrow \mathcal{M}_{\mathcal{C}_j} \not\models \Psi(\lambda\vec{d}.M\vec{t}[\vec{d}]) = \Psi(\lambda\vec{d}.N\vec{t}[\vec{d}])$
 $\Rightarrow \mathcal{M}_{\mathcal{C}_j} \not\models M = N$ since $\mathcal{M}_{\mathcal{C}_i}$ is a model.

(ii) By (i). ■

It is known that the inclusions $\text{Th}(\mathcal{M}_{\mathcal{C}_1}) \supseteq \text{Th}(\mathcal{M}_{\mathcal{C}_2})$, $\text{Th}(\mathcal{M}_{\mathcal{C}_2}) \supseteq \text{Th}(\mathcal{M}_{\mathcal{C}_3})$ and $\text{Th}(\mathcal{M}_{\mathcal{C}_4}) \supseteq \text{Th}(\mathcal{M}_{\mathcal{C}_5})$ are proper, do exercise 3.6.28. It is not known whether $\text{Th}(\mathcal{M}_{\mathcal{C}_3}) = \text{Th}(\mathcal{M}_{\mathcal{C}_4})$ holds.

3.5.17. LEMMA. *Let A, B be types such that $A \leq_{\beta\eta} B$. Suppose $\mathcal{M}_{\mathcal{D}}$ is B -complete for pure terms. Then $\mathcal{M}_{\mathcal{D}}$ is A -complete for pure terms.*

PROOF. Assume $\Phi : A \leq_{\beta\eta} B$. Then we have for $M, N \in \Lambda_o^\emptyset(A)$

$$\begin{array}{ccc} \mathcal{M}_{\mathcal{D}} \models M = N & \Leftarrow & M =_{\beta\eta} N \\ \Downarrow & & \Uparrow \\ \mathcal{M}_{\mathcal{D}} \models \Phi M = \Phi N & \Rightarrow & \Phi M =_{\beta\eta} \Phi N \end{array}$$

by the definition of reducibility. ■

3.5.18. COROLLARY. $\mathcal{M}_{\mathcal{C}_5}$ is complete for pure terms. In other words, for $M, N \in \Lambda_o^\emptyset$

$$\mathcal{M}_{\mathcal{C}_5} \models M = N \iff M =_{\beta\eta} N.$$

PROOF. By lemma 3.5.17 and the reducibility theorem 3.4.7. ■

So $\text{Th}(\mathcal{M}_{\mathcal{C}_5})$, the smallest theory, is actually just $\beta\eta$ -convertibility, which is decidable. At the other end something dual happens.

3.5.19. DEFINITION. $\mathcal{M}_{\min} = \mathcal{M}_{\mathcal{C}_1}$ is called the *minimal model* of λ_{\rightarrow} since it equates most terms.

3.5.20. PROPOSITION. Let $A \in \mathbb{T}_o$ be of the form $A = A_1 \rightarrow \dots A_n \rightarrow o$ and let $M, N \in \Lambda_o^\emptyset(A)$ be pure closed terms. Then the following statements are equivalent.

1. $M = N$ is inconsistent.
2. For all models \mathcal{M} of λ_{\rightarrow} one has $\mathcal{M} \not\models M = N$.
3. $\mathcal{M}_{\min} \not\models M = N$.
4. $\exists P_1 \in \Lambda^{x,y:o}(A_1) \dots P_n \in \Lambda^{x,y:o}(A_n). M\vec{P} = x \ \& \ N\vec{P} = y$.
5. $\exists F \in \Lambda^{x,y:o}(A \rightarrow o). FM = x \ \& \ FN = y$.
6. $\exists G \in \Lambda^\emptyset(A \rightarrow o^2 \rightarrow o). FM = \lambda xy.x \ \& \ FN = \lambda xy.y$.

PROOF. (1) \Rightarrow (2) By soundness. (2) \Rightarrow (3) Trivial. (3) \Rightarrow (4) Since \mathcal{M}_{\min} consists of $\Lambda^{x,y:o}/\approx_{\mathcal{C}_1}$. (4) \Rightarrow (5) By taking $F \equiv \lambda m.m\vec{P}$. (5) \Rightarrow (6) By taking $G \equiv \lambda mxy.Fm$. (6) \Rightarrow (1) Trivial. ■

3.5.21. COROLLARY. $\text{Th}(\mathcal{M}_{\min})$ is the unique maximally consistent extension of λ_{\rightarrow}^o .

PROOF. By taking in the proposition the negations one has $M = N$ is consistent iff $\mathcal{M}_{\min} \models M = N$. Hence $\text{Th}(\mathcal{M}_{\min})$ contains all consistent equations. Moreover this theory is consistent. Therefore the statement follows. ■

In Section 4.4 it will be proved that $\text{Th}(\mathcal{M}_{\mathcal{C}_1})$ is decidable. $\mathcal{M}_{\mathcal{C}_0}$ is the degenerate model consisting of one element at each type, since

$$\forall M, N \in \Lambda_o^\emptyset[\mathcal{C}_0](o) \ M = x = N.$$

Therefore its theory is inconsistent and hence decidable. For the other theories, $\text{Th}(\mathcal{M}_{\mathcal{C}_2})$, $\text{Th}(\mathcal{M}_{\mathcal{C}_3})$ and $\text{Th}(\mathcal{M}_{\mathcal{C}_4})$ it is not known whether they are decidable.

Now we turn attention again to arbitrary sets of constants \mathcal{D} . It will turn out that the results can be proved for arbitrary sets of constants \mathcal{D} .

3.5.22. DEFINITION. The set of types \mathbb{T}_i is called *resource conscious* iff for some $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_a \in \mathbb{T}_i$ one has $A_1 \rightarrow A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_a \notin \mathbb{T}_i$. ■

Note that only \mathbb{T}_2 and \mathbb{T}_0 are resource conscious.

3.5.23. LEMMA. Let \mathcal{D} have class i with \mathbb{T}_i not resource conscious and let $\mathcal{C} = \mathcal{C}_i$. Let $B = B_1 \rightarrow \dots \rightarrow B_b \rightarrow o$ and $P, Q : \Lambda_o^\emptyset[\mathcal{D}](B)$. Then

- (i) $\forall \vec{t} \in \Lambda_o^\emptyset[\mathcal{D}] \ P\vec{t} =_{\beta\eta} Q\vec{t} (: o) \Rightarrow \forall \vec{t} \in \Lambda_o^\emptyset[\mathcal{D} \cup \mathcal{C}] \ P\vec{t} =_{\beta\eta} Q\vec{t}.$
- (ii) $P\vec{c} \neq_{\beta\eta} Q\vec{c} \Rightarrow PU[\vec{d}] \neq_{\beta\eta} QU[\vec{d}],$ for some $U[\vec{d}] \in \Lambda_o^\emptyset[\mathcal{D}].$

PROOF. (i) Suppose $P\vec{t} \neq_{\beta\eta} Q\vec{t}$ for some $\vec{t} \in \Lambda[\mathcal{D} \cup \mathcal{C}]$, in order to show that $P\vec{s} \neq_{\beta\eta} Q\vec{s}$ for some $\vec{s} \in \Lambda_o^\emptyset[\mathcal{D}]$. Write $\vec{t} [\vec{c}, \vec{d}]$, displaying explicitly the constants from \mathcal{C} and \mathcal{D} . Then

$$\begin{aligned} & P\vec{t} [\vec{c}, \vec{d}] \neq_{\beta\eta} Q\vec{t} [\vec{c}, \vec{d}] && : o, \\ \Rightarrow & P\vec{t} [\vec{c}, \vec{d}] \neq_{\beta\eta} Q\vec{t} [\vec{c}, \vec{d}] && : o, \\ & \text{using variables } \vec{c} \text{ corresponding to } \vec{c} \text{ (same number and types),} \\ \Rightarrow & \lambda \vec{c}. P\vec{t} [\vec{c}, \vec{d}] \neq_{\beta\eta} \lambda \vec{c}. Q\vec{t} [\vec{c}, \vec{d}] && : C = \nabla(\mathcal{C}) \in \mathbb{T}_i, \\ \Rightarrow & \lambda \vec{d}\vec{c}. P\vec{t} [\vec{c}, \vec{d}] \neq_{\beta\eta} \lambda \vec{d}\vec{c}. Q\vec{t} [\vec{c}, \vec{d}] && : C' \in \mathbb{T}_i, \\ & \text{since } \mathbb{T}_i \text{ is not resource conscious,} \\ \Rightarrow & \Phi_k(\lambda \vec{d}\vec{c}. P\vec{t} [\vec{c}, \vec{d}]) \neq_{\beta\eta} \Phi_k(\lambda \vec{d}\vec{c}. Q\vec{t} [\vec{c}, \vec{d}]), && : D, \\ & \text{for some } D \text{ represented in } \mathcal{D} \text{ with } C' \leq_{h^+} D \text{ via } \Phi_1, \dots, \Phi_m, \\ \Rightarrow & (\lambda \vec{d}\vec{c}. P\vec{t} [\vec{c}, \vec{d}])\vec{M} \neq_{\beta\eta} (\lambda \vec{d}\vec{c}. Q\vec{t} [\vec{c}, \vec{d}])\vec{M} && : D, \\ \Rightarrow & (\lambda \vec{d}\vec{c}. P\vec{t} [\vec{c}, \vec{d}])\vec{M}\vec{d} \neq_{\beta\eta} (\lambda \vec{d}\vec{c}. Q\vec{t} [\vec{c}, \vec{d}])\vec{M}\vec{d} && : o, \\ \Rightarrow & P\vec{s} [\vec{d}] \neq_{\beta\eta} Q\vec{s} [\vec{d}] && : o. \end{aligned}$$

(ii) By (i). ■

In exercise 3.6.25 it is shown that this lemma is false for \mathcal{D} of class 0 and 2.

3.5.24. PROPOSITION. Let \mathcal{D} be of class i with \mathbb{T}_i not resource conscious and canonical set $\mathcal{C} = \mathcal{C}_i$. Let A be an arbitrary type.

- (i) For all $P[\vec{d}], Q[\vec{d}] \in \Lambda_o^\emptyset[\mathcal{D}](A)$, one has

$$P[\vec{d}] \approx_{\mathcal{D}}^{\text{ext}} Q[\vec{d}] \iff \lambda \vec{x}. P[\vec{x}] \approx_{\mathcal{C}} \lambda \vec{x}. Q[\vec{x}]$$

(ii) In particular, for pure closed terms $P, Q \in \Lambda_o^\emptyset(A)$ one has

$$P \approx_{\mathcal{D}}^{\text{ext}} Q \iff P \approx_{\mathcal{C}} Q.$$

PROOF. (i) Although the proof is given in contrapositive form, it nevertheless can be made constructive.

(\Leftarrow) We will show $P[\vec{d}] \not\approx_{\mathcal{D}}^{\text{ext}} Q[\vec{d}] \Rightarrow \lambda \vec{x}.P[\vec{x}] \not\approx_{\mathcal{C}} \lambda \vec{x}.Q[\vec{x}]$. Indeed

$$\begin{aligned} P[\vec{d}] \not\approx_{\mathcal{D}}^{\text{ext}} Q[\vec{d}] &\Rightarrow P[\vec{d}]\vec{T}[\vec{d}, \vec{e}] \neq_{\beta\eta} Q[\vec{d}]\vec{T}[\vec{d}, \vec{e}], \text{ for some} \\ &\quad \vec{T}[\vec{d}, \vec{e}] \in \Lambda_o^\emptyset[\mathcal{D}], \text{ each } T_i[\vec{d}, \vec{e}] \text{ of the form } T_i \vec{d} \vec{e} \text{ with} \\ &\quad \text{the } T_i \text{ pure and closed and the } \vec{d}, \vec{e} \text{ all distinct,} \\ &\Rightarrow P[\vec{x}]\vec{T}[\vec{x}, \vec{y}] \neq_{\beta\eta} Q[\vec{x}]\vec{T}[\vec{x}, \vec{y}] \text{ (: o),} \\ &\Rightarrow \lambda \vec{x}.P[\vec{x}] \not\approx_{\mathcal{C}} \lambda \vec{x}.Q[\vec{x}], \text{ applying theorem 3.5.11 to} \\ &\quad P \equiv \lambda z \vec{x} \vec{y}. z \vec{x} \vec{T}[\vec{x}, \vec{y}]. \end{aligned}$$

(\Rightarrow) Again we show the contrapositive.

$$\begin{aligned} \lambda \vec{x}.P[\vec{x}] \not\approx_{\mathcal{C}} \lambda \vec{x}.Q[\vec{x}] &\Rightarrow P[\vec{V}[\vec{c}]]\vec{W}[\vec{c}] \neq_{\beta\eta} Q[\vec{V}[\vec{c}]]\vec{W}[\vec{c}], \\ &\quad \text{where } V[\vec{c}], W[\vec{c}] \in \Lambda_o^\emptyset[\mathcal{C}], \\ &\Rightarrow P[\vec{x}]\vec{W}[\vec{c}] \neq_{\beta\eta} Q[\vec{x}]\vec{W}[\vec{c}], \\ &\quad \text{otherwise one could substitute for the } \vec{x}, \\ &\Rightarrow P[\vec{d}]\vec{W}[\vec{c}] \neq_{\beta\eta} Q[\vec{d}]\vec{W}[\vec{c}], \\ &\quad \text{with } \vec{d} \in \mathcal{D}, \\ &\Rightarrow P[\vec{d}]\vec{W}[\vec{U}[\vec{d}]] \neq_{\beta\eta} Q[\vec{d}]\vec{W}[\vec{U}[\vec{d}]], \\ &\quad \text{by lemma 3.5.23(ii),} \\ &\Rightarrow P[\vec{d}] \not\approx_{\mathcal{D}}^{\text{ext}} Q[\vec{d}]. \end{aligned}$$

(ii) By (i). ■

3.5.25. COROLLARY. Let \mathcal{D} be a class that is not resource conscious.

- (i) The relation $\approx_{\mathcal{D}}^{\text{ext}}$ is logical.
- (ii) The relations $\approx_{\mathcal{D}}^{\text{ext}}$ and $\approx_{\mathcal{D}}^{\text{obs}}$ on $\Lambda_o^\emptyset[\mathcal{D}]$ coincide.

PROOF. (i) We have to show for all $F, G \in \Lambda_o^\emptyset[\mathcal{D}](A \rightarrow B)$ that $F \approx_{\mathcal{D}}^{\text{ext}} G \iff$

$$\forall M, N \in \Lambda_o^\emptyset[\mathcal{D}](A) [M \approx_{\mathcal{D}}^{\text{ext}} N \Rightarrow FM \approx_{\mathcal{D}}^{\text{ext}} GN]. \quad (1)$$

(\Rightarrow) Assume $F[\vec{d}] \approx_{\mathcal{D}}^{\text{ext}} G[\vec{d}]$ and $M[\vec{d}] \approx_{\mathcal{D}}^{\text{ext}} N[\vec{d}]$, with $F, G \in \Lambda_o^\emptyset[\mathcal{D}](B \rightarrow C)$ and $M, N \in \Lambda_o^\emptyset[\mathcal{D}](B)$. By the proposition one has $\lambda \vec{x}.F[\vec{x}] \approx_{\mathcal{C}} \lambda \vec{x}.G[\vec{x}]$ and $\lambda \vec{x}.M[\vec{x}] \approx_{\mathcal{C}} \lambda \vec{x}.N[\vec{x}]$. Consider the pure closed term

$$H \equiv \lambda f:(B \rightarrow C) \lambda m:B \lambda \vec{x}. f \vec{x}(m \vec{x}).$$

Then $H \approx_C H$, since \approx_C is logical. It follows that

$$\begin{aligned} \lambda \vec{x}.F[\vec{x}]M[\vec{x}] &=_{\beta\eta} H(\lambda \vec{x}.F[\vec{x}])(\lambda \vec{x}.M[\vec{x}]) \\ &\approx_C H(\lambda \vec{x}.F[\vec{x}])(\lambda \vec{x}.M[\vec{x}]) \\ &=_{\beta\eta} \lambda \vec{x}.G[\vec{x}]N[\vec{x}]. \end{aligned}$$

But then again by the proposition

$$F[\vec{d}]M[\vec{d}] \approx_{\mathcal{D}}^{\text{ext}} G[\vec{d}]N[\vec{d}].$$

(\Leftarrow) Assume $F[\vec{d}] \not\approx_{\mathcal{D}}^{\text{ext}} G[\vec{d}]$. Then by the proposition

$$\begin{aligned} &\lambda \vec{x}.F[\vec{x}] \not\approx_C \lambda \vec{x}.G[\vec{x}], \\ \Rightarrow &F[\vec{T}[\vec{c}]]S[\vec{c}] \neq_{\beta\eta} G[\vec{T}[\vec{c}]]S[\vec{c}], \\ \Rightarrow &F[\vec{d}]S[\vec{c}] \neq_{\beta\eta} G[\vec{d}]S[\vec{c}], \\ &\text{otherwise one could substitute the } \vec{T}[\vec{c}] \text{ for the } \vec{d}, \\ \Rightarrow &F[\vec{d}]S[\vec{U}[\vec{d}]] \neq_{\beta\eta} G[\vec{d}]S[\vec{U}[\vec{d}]], \\ &\text{by lemma 3.5.23(ii),} \end{aligned}$$

contradicting (1), since of course $S[\vec{U}[\vec{d}]] \approx_{\mathcal{D}}^{\text{ext}} S[\vec{U}[\vec{d}]]$.

(ii) That $\approx_{\mathcal{D}}^{\text{ext}}$ is $\approx_{\mathcal{D}}^{\text{obs}}$ on $\Lambda_o^\emptyset[\mathcal{D}]$ follows by lemma 3.5.5. ■

3.5.26. LEMMA. *Let \mathcal{D} be of class 2. Then one of the following cases holds.*

$$\begin{aligned} \mathcal{D} &= \{\mathbf{F}:2, \mathbf{x}_1 \dots, \mathbf{x}_n:o\}, \quad n \geq 0; \\ \mathcal{D} &= \{\mathbf{f}:1, \mathbf{x}_1 \dots, \mathbf{x}_n:o\}, \quad n \geq 1. \end{aligned}$$

PROOF. Do exercise 3.6.19. ■

3.5.27. PROPOSITION. *Let \mathcal{D} be of class 2. Then the following hold.*

- (i) *The relation $\approx_{\mathcal{D}}^{\text{ext}}$ is logical.*
- (ii) *The relations $\approx_{\mathcal{D}}^{\text{ext}}$ and $\approx_{\mathcal{D}}^{\text{obs}}$ on $\Lambda_o^\emptyset[\mathcal{D}]$ coincide.*
- (iii) $\forall M, N \in \Lambda_o^\emptyset[\mathcal{D}] [M \approx_{\mathcal{D}} N \iff M \approx_{\mathcal{C}_2} N]$.

PROOF. (i) Assume that $\mathcal{D} = \{\mathbf{F}, \mathbf{x}_1, \dots, \mathbf{x}_n\}$ (the other possibility according lemma 3.5.26 is more easy). By proposition 3.5.4 (i) it suffices to show that for $\mathbf{d} \in \mathcal{D}$ one has $S(\mathbf{d}, \mathbf{d})$. This is easy for the ones of type o . For $\mathbf{F}:2$ we must show $f \approx_{\mathcal{D}}^{\text{ext}} g \Rightarrow f =_{\beta\eta} g$ for $f, g \in \Lambda_o^\emptyset[\mathcal{D}](1)$. Now elements of $\Lambda_o^\emptyset[\mathcal{D}](1)$ are of the form

$$\lambda x_1.\mathbf{F}(\lambda x_2.\mathbf{F}(\dots(\lambda x_{m-1}.\mathbf{F}(\lambda x_m.c))\dots)),$$

where $c \equiv x_i$ or $c \equiv \mathbf{x}_j$. Therefore if $f \neq_{\beta\eta} g$, then inspecting the various possibilities (e.g. one has

$$\begin{aligned} f &\equiv \lambda x_1.\mathbf{F}(\lambda x_2.\mathbf{F}(\dots(\lambda x_{m-1}.\mathbf{F}(\lambda x_m.x_n))\dots)) \equiv \mathbf{K}A \\ g &\equiv \lambda x_1.\mathbf{F}(\lambda x_2.\mathbf{F}(\dots(\lambda x_{m-1}.\mathbf{F}(\lambda x_m.x_1))\dots)), \end{aligned}$$

do exercise 3.6.18) one has $f(\mathbf{F}f) \neq_{\beta\eta} g(\mathbf{F}f)$ or $f(\mathbf{F}g) \neq_{\beta\eta} g(\mathbf{F}g)$, hence $f \not\approx_{\mathcal{D}}^{\text{ext}} g$.

- (ii) By (i) and lemma 3.5.5.
- (iii) Let $M, N \in \Lambda_o^\emptyset$. Then, using exercise 3.6.21 (iii),

$$\begin{aligned}
M \not\approx_{\mathcal{D}}^{\text{ext}} N &\iff \\
&\iff Mt_1[\vec{d}] \dots t_n[\vec{d}] \neq_{\beta\eta} Nt_1[\vec{d}] \dots t_n[\vec{d}] \\
&\quad \text{for some } t_1[\vec{d}] \dots t_n[\vec{d}] \in \Lambda_o^\emptyset[\mathcal{D}] \\
&\iff \lambda\vec{d}.Mt_1[\vec{d}] \dots t_n[\vec{d}] \neq_{\beta\eta} \lambda\vec{d}.Nt_1[\vec{d}] \dots t_n[\vec{d}], \\
&\Rightarrow \lambda fx.(\lambda\vec{d}.Mt_1[\vec{d}] \dots t_n[\vec{d}]) (Ofx)(O_1fx) \dots (O_nfx) \neq_{\beta\eta} \\
&\quad \lambda fx.(\lambda\vec{d}.Nt_1[\vec{d}] \dots t_n[\vec{d}]) (Ofx)(O_1fx) \dots (O_nfx) \\
&\Rightarrow Mt_1[O\mathbf{F}\mathbf{x}, O_1\mathbf{F}\mathbf{x}, \dots] \dots t_n[O\mathbf{F}\mathbf{x}, O_1\mathbf{F}\mathbf{x}, \dots] \neq_{\beta\eta} \\
&\quad Nt_1[O\mathbf{F}\mathbf{x}, O_1\mathbf{F}\mathbf{x}, \dots] \dots t_n[O\mathbf{F}\mathbf{x}, O_1\mathbf{F}\mathbf{x}, \dots] \\
&\Rightarrow M \not\approx_{\mathcal{C}}^{\text{ext}} N.
\end{aligned}$$

The converse can be proved similarly, using exercise 3.6.21 (iv). ■

3.5.28. PROPOSITION. *Let \mathcal{D} be of class 0. Then the following hold.*

- (i) $\approx_{\mathcal{D}}^{\text{ext}}$ is a logical relation, hence $\approx_{\mathcal{D}}^{\text{ext}}$ is $\approx_{\mathcal{D}}^{\text{obs}}$ on $\Lambda_o^\emptyset[\mathcal{D}]$.
- (ii) $\forall M, N \in \Lambda_o^\emptyset [M \approx_{\mathcal{D}} N \iff M \approx_{\mathcal{C}_0} N \iff \models_{\beta\eta}]$.

PROOF. It is not hard to see that \mathcal{D} is of class 0 iff $\Lambda_o^\emptyset[\mathcal{D}](o)$ is a singleton. Therefore for these \mathcal{D} the statements hold trivially.

Harvesting the results we obtain the following.

3.5.29. THEOREM. [Statman [1980b]] *Let \mathcal{D} be a set of constants such that there are enough closed terms. Then we have the following.*

- (i) $\approx_{\mathcal{D}}^{\text{ext}}$ is logical.
- (ii) $\approx_{\mathcal{D}}^{\text{ext}}$ is $\approx_{\mathcal{D}}^{\text{obs}}$.
- (iii) $\mathcal{M}_{\mathcal{D}}$ is a well defined extensional term-model.
- (iv) $\mathcal{M}_{\mathcal{D}}$ is fully abstract and o-complete.

Moreover, if \mathcal{D} is of class i , then

- (v) $\mathcal{M}_{\mathcal{D}} \equiv \mathcal{M}_{\mathcal{C}_i}$, i.e. $\forall M, N \in \Lambda_o^\emptyset [M \approx_{\mathcal{D}} N \iff M \approx_{\mathcal{C}_i} N]$.
- (vi) $\mathcal{M}_{\mathcal{D}}$ is $\nabla(\mathcal{C}_i)$ -complete for pure terms.

PROOF. (i), (ii) By corollary 3.5.25 and propositions 3.5.27 and 3.5.28.

(iii), (iv) As in theorem 3.5.13 and corollary 3.5.15.

(v) By proposition 3.5.27 (ii).

(vi) By (v) and corollary 3.5.15. ■

3.5.30. REMARK. So there are at most five canonical term-models that are not elementary equivalent (plus the degenerate term-model that does not count).

Applications

3.5.31. LEMMA. Let $A \in \mathbb{T}(\lambda_{\rightarrow}^o)$ and suppose $\mathcal{M}_{\mathcal{D}}$ is A -complete for pure terms. Then for $M, N \in \Lambda_o^{\emptyset}(B)$ one has

$$[\mathcal{M}_{\mathcal{D}} \models M = N \ \& \ M \neq_{\beta\eta} N] \Rightarrow B \not\leq_{\beta\eta} A.$$

PROOF. Suppose $B \leq_{\beta\eta} A$ via $F:B \rightarrow A$. Then

$$\begin{aligned} M \neq_{\beta\eta} N &\Rightarrow FM \neq_{\beta\eta} FN \\ &\Rightarrow \mathcal{M}_{\mathcal{D}} \not\models FM = FN, \quad \text{because of } A\text{-completeness,} \\ &\Rightarrow \mathcal{M}_{\mathcal{D}} \not\models M = N, \end{aligned}$$

a contradiction. ■

In the previous section the types A_{α} were introduced. The next proposition is needed to prove that they form a hierarchy.

3.5.32. PROPOSITION. For $\alpha, \beta \leq \omega + 3$ one has

$$\alpha \leq \beta \Leftarrow A_{\alpha} \leq_{\beta\eta} A_{\beta}.$$

PROOF. Notice that for $\alpha \leq \omega$ the cardinality of $\Lambda_o^{\emptyset}(A_{\alpha})$ equals α : For example $\Lambda_o^{\emptyset}(A_2) = \{\lambda xy:o.x, \lambda xy:o.y\}$ and $\Lambda_o^{\emptyset}(A_{\omega}) = \{\lambda f:1\lambda x:o.f^k x \mid k \in \mathbb{N}\}$. Therefore for $\alpha, \alpha' \leq \omega$ one has $A_{\alpha} \leq_{\beta\eta} A_{\alpha'} \Rightarrow \alpha = \alpha'$.

It remains to show that $A_{\omega+1} \not\leq_{\beta\eta} A_{\omega}, A_{\omega+2} \not\leq_{\beta\eta} A_{\omega+1}, A_{\omega+3} \not\leq_{\beta\eta} A_{\omega+2}$.

As to $A_{\omega+1} \not\leq_{\beta\eta} A_{\omega}$, consider

$$\begin{aligned} M &\equiv \lambda f, g:1\lambda x:o.f(g(f(gx))), \\ N &\equiv \lambda f, g:1\lambda x:o.f(g(g(fx))). \end{aligned}$$

Then $M, N \in \Lambda_o^{\emptyset}(A_{\omega+1})$, and $M \neq_{\beta\eta} N$. Note that $\mathcal{M}_{\mathcal{C}_2}$ is A_{ω} -complete. It is not difficult to show that $\mathcal{M}_{\mathcal{C}_2} \models M = N$, by analyzing the elements of $\Lambda_o^{\emptyset}[\mathcal{C}_2](1)$. Therefore, by lemma 3.5.31, the conclusion follows.

As to $A_{\omega+2} \not\leq_{\beta\eta} A_{\omega+1}$, this is proved in Dekkers [1988].

As to $A_{\omega+3} \not\leq_{\beta\eta} A_{\omega+2}$, consider

$$\begin{aligned} M &\equiv \lambda h:1_2\lambda x:o.h(hx(hxx))(hxx), \\ N &\equiv \lambda h:1_2\lambda x:o.h(hxx)(h(hxx)x). \end{aligned}$$

Then $M, N \in \Lambda_o^{\emptyset}(A_{\omega+3})$, and $M \neq_{\beta\eta} N$. Note that $\mathcal{M}_{\mathcal{C}_4}$ is $A_{\omega+2}$ -complete. It is not difficult to show that $\mathcal{M}_{\mathcal{C}_4} \models M = N$, by analyzing the elements of $\Lambda_o^{\emptyset}[\mathcal{C}_4](1_2)$. Therefore, by lemma 3.5.31, the conclusion follows. ■

3.6. Exercises

3.6.1. The iterated exponential function 2_n is

$$\begin{aligned} 2_0 &= 1, \\ 2_{n+1} &= 2^{2_n}. \end{aligned}$$

One has $2_n = 2_n(1)$, according to the definition before Exercise 2.5.16. Define $s(A)$ to be the number of occurrences of atoms in the type A , i.e.

$$\begin{aligned} s(o) &= 1 \\ s(A \rightarrow B) &= s(A) + s(B). \end{aligned}$$

Write $\#X$ for the cardinality of the set X . Show the following.

- (i) $2_n \leq 2_{n+p}$.
- (ii) $2_{n+2}^{2^{p+1}} \leq 2_{n+p+3}$.
- (iii) $2_n^{2^p} \leq 2_{n+p}$.
- (iv) If $X = \{0, 1\}$, then $\forall A \in \mathbb{T}. \#(X(A)) \leq 2_{s(A)}$.
- (v) For which types A do we have $=$ in (iv)?

3.6.2. Show that if \mathcal{M} is a type model, then for the corresponding polynomial type model \mathcal{M}^* one has $\text{Th}(\mathcal{M}^*) = \text{Th}(\mathcal{M})$.

3.6.3. Show that

$$A_1 \rightarrow \dots \rightarrow A_n \rightarrow o \leq_{\beta\eta} A_{\pi 1} \rightarrow \dots \rightarrow A_{\pi n} \rightarrow o,$$

for any permutation $\pi \in S_n$

3.6.4. Let $A = (2 \rightarrow 2 \rightarrow o) \rightarrow 2 \rightarrow o$ and
 $B = (o \rightarrow 1^2 \rightarrow o) \rightarrow 1_2 \rightarrow (o \rightarrow 1 \rightarrow o) \rightarrow o^2 \rightarrow o$. Show that

$$A \leq_{\beta\eta} B.$$

[Hint. Use the term $\lambda z:A \lambda u_1:(o \rightarrow 1^2 \rightarrow o) \lambda u_2:1_2 \lambda u_3:(o \rightarrow 2) \lambda x_1 x_2:o.$
 $z[\lambda y_1, y_2:2. u_1 x_1 (\lambda w:o. y_1 (u_2 w)) (\lambda w:o. y_2 (u_2 w))] [u_3 x_2].$]

3.6.5. Let $A = (1^2 \rightarrow o) \rightarrow o$. Show that

$$A \leq_{\beta\eta} 1_2 \rightarrow 2 \rightarrow o.$$

[Hint. Use the term
 $\lambda M:(A \rightarrow o) \lambda p:1_2 \lambda F:2. M(\lambda f, g:1. F(\lambda z:o. p(fz)(gz)))$.]

3.6.6. (i) Show that

$$\begin{pmatrix} 2 & \\ 3 & 4 \end{pmatrix} \leq_{\beta\eta} 1 \rightarrow 1 \rightarrow \begin{pmatrix} 2 & \\ 3 & 3 \end{pmatrix}.$$

(ii) Show that

$$\begin{pmatrix} 2 & \\ 3 & 3 \end{pmatrix} \leq_{\beta\eta} 1 \rightarrow 1 \rightarrow \begin{pmatrix} 2 \\ 3 \end{pmatrix}.$$

(iii) * Show that

$$\begin{pmatrix} 2 & 2 \\ 3 & 2 \end{pmatrix} \leq_{\beta\eta} 1_2 \rightarrow \begin{pmatrix} 2 \\ 3 & 2 \end{pmatrix}.$$

[Hint. Use $\Phi = \lambda M \lambda p : 1_2 \lambda H'_1 H_2 . M$
 $[\lambda f_{11}, f_{12} : 1_2 . H'_1 (\lambda xy : o . p(f_{12} xy, H_2 f_{11}))]$
 $[\lambda f_{21} : 1_3 \lambda f_{22} : 1_2 . H_2 f_{21} f_{22}].]$

3.6.7. Show that $2 \rightarrow o \leq_{\beta\eta} 1 \rightarrow 1 \rightarrow o \rightarrow o$. [Hint. Use

$$\Phi \equiv \lambda M : 2 \lambda f, g : 1 \lambda z : o . M(\lambda h : 1 . f(h(g(hz))))).$$

Typical elements of type 3 are $M_i \equiv \lambda F : 2 . F(\lambda x_1 . F(\lambda x_2 . x_i))$. Show that Φ acts injectively (modulo $\beta\eta$) on these.]

3.6.8. Give example of $F, G \in \Lambda[C_4]$ such that $Fh_2 =_{\beta\eta} Gh_2$, but $F \neq_{\beta\eta} G$, where $h_2 \equiv \lambda z : o . \Phi(\lambda g : 1 . g(gz))$.

3.6.9. (Joly [2001], Lemma 2, p. 981, based on an idea of Dana Scott) Show that any type A is reducible to

$$1_2 \rightarrow 2 \rightarrow o = (o \rightarrow (o \rightarrow o)) \rightarrow ((o \rightarrow o) \rightarrow o) \rightarrow o.$$

[Hint. We regard each closed term of type A as an untyped lambda term and then we retype all the variables as type o replacing applications XY by $fXY (= X \bullet Y)$ and abstractions $\lambda x . X$ by $g(\lambda x . X) (= \lambda^\bullet x . X)$ where $f : 1_2, g : 2$. Scott thinks of f and g as a retract pair satisfying $g \circ f = \text{id}$ (of course in our context they are just variables which we abstract at the end). The exercise is to define terms which ‘do the retyping’ and insert the f and g , and to prove that they work. For $A = A_1 \rightarrow \dots \rightarrow A_s \rightarrow 0$ define terms $U_A : A \rightarrow 0$ and $V_A : 0 \rightarrow A$ as follows. If $s = 0$, i.e. $A = o$, then

$$U_o := \lambda x : o . x; \quad V_o := \lambda x : o . x.$$

If $s > 0$, then

$$\begin{aligned} U_A &:= \lambda u . \lambda^\bullet x_1 \dots \lambda^\bullet x_s . u(V_{A_1} x_1) \dots (V_{A_s} x_s) \\ V_A &:= \lambda v \lambda y_1 \dots y_s . v \bullet (U_{A_1} y_1) \bullet \dots \bullet (U_{A_s} y_s). \end{aligned}$$

Let $A_i = A_{i_1} \rightarrow \dots \rightarrow A_{i_{r_i}} \rightarrow 0$ and suppose that $M : A$ is in long $\beta\eta$ -nf

$$M = \lambda u_1 \dots u_s . u_i Y_1 \dots Y_{r_i}.$$

Then

$$\begin{aligned} UM &\rightarrow \lambda^\bullet x_1 \dots \lambda^\bullet x_s . M(V_1 x_1) \dots (V_s x_s) \\ &\rightarrow \lambda^\bullet x_1 \dots \lambda^\bullet x_s . V_i x_i (@Y_1) \dots (@Y_{r_i}) \\ &\Rightarrow \lambda^\bullet x_1 \dots \lambda^\bullet x_s . x_i \bullet (U_1 @Y_1) \bullet \dots \bullet (U_s @Y_s). \end{aligned}$$

where $@Y = [\dots u_j := V_j x_j \dots]Y$. Show that for all closed X, Y

$$U_A X =_{\beta\eta} U_A Y \Rightarrow X =_{\beta\eta} Y,$$

by making an appropriate induction hypothesis for open terms. Conclude that $A \leq_{\beta\eta} 1_2 \rightarrow 2 \rightarrow o$ via $\Phi \equiv \lambda b f g. U_A b.$

3.6.10. In this exercise the combinatorics of the argument needed in the proof of 3.4.5 is analyzed. Let $(\lambda F:2.M) : 3$. Define M^+ to be the long $\beta\eta$ nf of $M[F := H]$, where

$$H = (\lambda h:1.f(h(g(hz)))) \in \Lambda_{\rightarrow}^{\{f,g:1,z:o\}}(2).$$

Write $\text{cut}_{g \rightarrow z}(P) = P[g := Kz]$.

- (i) Show by induction on M that if $g(P) \subseteq M^+$ is maximal (i.e. $g(P)$ is not a proper subterm of a $g(P') \subseteq M^+$), then $\text{cut}_{g \rightarrow z}(P)$ is a proper subterm of $\text{cut}_{g \rightarrow z}(M^+)$.
- (ii) Let $M \equiv F(\lambda x:o.N)$. Then we know

$$M^+ =_{\beta\eta} f(N^+[x := g(N^+[x := z])]).$$

Show that if $g(P) \subseteq M^+$ is maximal and $\text{length}(\text{cut}_{g \rightarrow z}(P)) + 1 = \text{length}(\text{cut}_{g \rightarrow z}(M^+))$, then $g(P)$ is $\equiv g(N^+[x := z])$ being substituted for an occurrence of x in N^+ .

- (iii) Show that the occurrences of $g(P)$ in M^+ that are maximal and satisfy $\text{length}(\text{cut}_{g \rightarrow z}(P)) + 1 = \text{length}(\text{cut}_{g \rightarrow z}(M^+))$ are exactly those that were substituted for the occurrences of x in N^+ .
- (iv) Show that (up to $=_{\beta\eta}$) M can be reconstructed from M^+ .

3.6.11. Show directly that (without the reducibility theorem)

$$3 \rightarrow o \rightarrow o \leq_{\beta\eta} 1_2 \rightarrow o \rightarrow o = \top.$$

3.6.12. Show directly the following.

- (i) $1_3 \rightarrow 1_2 \rightarrow o \leq_{\beta\eta} \top$.
- (ii) For any type A of rank ≤ 2 one has $A \leq_{\beta\eta} \top$.

3.6.13. Show that all elements $g \in \mathcal{M}_2(o \rightarrow o)$ satisfy $g^2 = g^4$. Conclude that $\mathcal{T} \not\vdash \mathcal{M}_2$.

3.6.14. Show that $\mathcal{M}_n \hookrightarrow \mathcal{M}_\omega$, for $n \in \omega$.

3.6.15. A model \mathcal{M} is called finite iff $\mathcal{M}(A)$ is finite for all types A . Find out which of the five canonical termmodels is finite.

3.6.16. Let $\mathcal{M} = \mathcal{M}_{\min}$.

- (i) Determine in $\mathcal{M}(1 \rightarrow o \rightarrow o)$ which of the three Church's numerals $\mathbf{c}_0, \mathbf{c}_{10}$ and \mathbf{c}_{100} are equal and which not.
- (ii) Determine the elements in $\mathcal{M}(1_2 \rightarrow o \rightarrow o)$.

3.6.17. Let \mathcal{M} be a model and let $|\mathcal{M}_0| \leq \kappa$. By Example 3.3.24 there exists a partial surjective homomorphism $h : \mathcal{M}_\kappa \rightarrow \mathcal{M}$.

- (i) Show that $h^{-1}(\mathcal{M}) \subseteq \mathcal{M}_\kappa$ is closed under λ -definability. [Hint. Use Example 3.3.36.]
- (ii) Show that as in Example 3.3.37 one has $h^{-1}(\mathcal{M})^E = h^{-1}(\mathcal{M})$.
- (iii) Show that the Gandy Hull $h^{-1}(\mathcal{M})/E$ is isomorphic to \mathcal{M} .
- (iv) For the 5 canonical models \mathcal{M} construct $h^{-1}(\mathcal{M})$ directly without reference to \mathcal{M} .
- (v) (Plotkin) Do the same as (iii) for the free open term model.

3.6.18. Let $\mathcal{D} = \{\mathbf{F}:2, \mathbf{x}_1, \dots, \mathbf{x}_n\}$.

- (i) Give a characterisation of the elements of $\Lambda_o^\emptyset[\mathcal{D}](1)$.
- (ii) For $f, g \in \Lambda_o^\emptyset[\mathcal{D}](1)$ show that $f \neq_{\beta\eta} g \Rightarrow f \not\approx_{\mathcal{D}} g$ by applying both f, g to $\mathbf{F}f$ or $\mathbf{F}g$.

3.6.19. Let \mathcal{D} be of class 2. Show that either one of the following cases holds.

$$\begin{aligned} \mathcal{D} &= \{\mathbf{F}:2, \mathbf{x}_1 \dots, \mathbf{x}_n:o\}, \quad n \geq 0; \\ \mathcal{D} &= \{\mathbf{f}:1, \mathbf{x}_1 \dots, \mathbf{x}_n:o\}, \quad n \geq 1. \end{aligned}$$

3.6.20. Prove the following.

$$\begin{aligned} 1_2 \rightarrow o \rightarrow o &\leq_{\beta\eta} ((1_2 \rightarrow o) \rightarrow o) \rightarrow o \rightarrow o, \text{ via} \\ &\lambda m \lambda F:((1_2 \rightarrow o) \rightarrow o) \lambda x:o.F(\lambda h:1_2.mhx) \text{ or via} \\ &\lambda m \lambda F:((1_2 \rightarrow o) \rightarrow o) \lambda x:o.m(\lambda pq:o.F(\lambda h:1_2.hpq))x. \\ 1_2 \rightarrow o \rightarrow o &\leq_{\beta\eta} (1 \rightarrow 1 \rightarrow o) \rightarrow o \rightarrow o \\ &\text{via } \lambda m Hx.m(\lambda ab.H(\mathbf{K}a)(\mathbf{K}b))x. \end{aligned}$$

3.6.21. (i) Prove $2 \rightarrow o^n \rightarrow o \leq_{\beta\eta} 1 \rightarrow o \rightarrow o$. [Construct first three terms O_1, O_2, O_3 such that if $M, N \in \Lambda^\emptyset(2 \rightarrow o^n \rightarrow o)$ and $M \neq_{\beta\eta} N$, then one of the inequalities $M(O_i f x) \neq_{\beta\eta} N(O_i f x)$ holds. Complete the construction by taking as reducing term

$$\Phi \equiv \lambda m f x. P_3(m(O_1 f x))(m(O_2 f x))(m(O_3 f x)),$$

where P_3 is a polynomially definable coding of $\mathbb{N}^3 \rightarrow \mathbb{N}$.]

- (ii) Prove $1 \rightarrow o \rightarrow o \leq_{\beta\eta} 2 \rightarrow o^n \rightarrow o$ using $\Phi \equiv \lambda m F x_1 \dots x_n. m(FI)$.
- (iii) Let $M, N \in \Lambda^\emptyset(2 \rightarrow o^n \rightarrow o)$. Show, using (i), that if $M \neq_{\beta\eta} N$, then for some $O \in \Lambda^\emptyset$ one has

$$\lambda f x. M(O f x)(O_1 f x) \dots (O_n f x) \neq_{\beta\eta} \lambda f x. N(O f x)(O_1 f x) \dots (O_n f x).$$

- (iv) Let $M, N \in \Lambda^\emptyset(1 \rightarrow o \rightarrow o)$. Show, using (ii), that if $M \neq_{\beta\eta} N$, then for some $O \in \Lambda^\emptyset$ one has

$$\lambda F x_1 \dots x_n. M(O F x_1 \dots x_n) \neq_{\beta\eta} \lambda F x_1 \dots x_n. N(O F x_1 \dots x_n).$$

3.6.22. (i) Using exercise 3.6.21 prove that for $p > 0, q \geq 0$ one has

$$1 \rightarrow o \rightarrow o \sim_{\nabla} (1_p \rightarrow o) \rightarrow o^q \rightarrow o.$$

- (ii) Prove that for $q > 0$ one has $1 \rightarrow o \rightarrow o \sim \nabla 1 \rightarrow o^q \rightarrow o$.
- 3.6.23. Show that for all $A, B \in \mathbb{T}_2$ one has $A \sim_{h+} B$.
- 3.6.24. Show that for all $A, B \notin \mathbb{T}_2$ one has $A \sim_h B$.
- 3.6.25. (i) Show that lemma 3.5.23 is false for $\mathcal{D} = \{\mathbf{g}^1, \mathbf{d}^o\}$ and $\mathcal{C}_2 = \{\mathbf{f}^2, \mathbf{c}^o\}$. [Hint. Consider $P \equiv \lambda f g c. f(g(fc))$ and $Q \equiv \lambda f g c. f(f(gc))$.]
(ii) Show that that lemma is also false for $\mathcal{D} = \{\mathbf{d}^o\}$ and $\mathcal{C} = \{\mathbf{c}^o\}$.
- 3.6.26. Show that if $M \in \Lambda^\emptyset(3 \rightarrow o \rightarrow o)$, then M is of the form

$$\begin{aligned} \lambda \Phi : 3 \lambda c \ o. \quad & \Phi(\lambda f_1 : 1. w_{f_1} \\ & \Phi(\lambda f_1, f_2 : 1. w_{f_1, f_2} \\ & \dots \\ & \Phi(\lambda f_1, \dots, f_n : 1. w_{f_1, \dots, f_n} \\ & c) \dots), \end{aligned}$$

where w_{f_1, \dots, f_n} stands for a “word” over the alphabet $\Sigma = \{f_1, \dots, f_n\}$ in the sense that e.g. $f_1 f_2 f_1$ is to be interpreted as $f_1 \circ f_2 \circ f_1$. [Hint. See the proof of lemma 3.5.9.]

- 3.6.27. Let A be an inhabited small type of rank > 3 . Show that

$$3 \rightarrow o \rightarrow o \leq_m A.$$

[Hint. For small B of rank ≥ 2 one has $B \equiv B_1 \rightarrow \dots B_b \rightarrow o$ with $B_i \equiv B_{i_1} \rightarrow o$ for all i and $\text{rank}(B_{i_{01}}) = \text{rank}(B) - 2$ for some i_o . Define for such B the term

$$X^B \in \Lambda^\emptyset[F^2](B),$$

where F^2 is a variable of type 2.

$$\begin{aligned} X^B &\equiv \lambda x_1 \dots x_b. F^2 x_{i_0}, & \text{if } \text{rank}(B) = 2; \\ &\equiv \lambda x_1 \dots x_b. F^2(\lambda y : 0. x_{i_0}(\lambda y_1 \dots y_k. y)), & \text{if } \text{rank}(B) = 3 \text{ and} \\ & & \text{where } B_{i_0} \text{ having} \\ & & \text{rank 1 is } o^k \rightarrow o; \\ &\equiv \lambda x_1 \dots x_b. x_{i_0} X^{B_{i_{01}}}, & \text{if } \text{rank}(B) > 3. \end{aligned}$$

(Here $X^{B_{i_{01}}}$ is well-defined since $B_{i_{01}}$ is also small.) As A is inhabited, take $\lambda x_1 \dots x_b. N \in \Lambda^\emptyset(A)$. Define $\Psi : (3 \rightarrow o \rightarrow o) \rightarrow A$ by

$$\Psi(M) = \lambda x_1 \dots x_b. M(\lambda F^2. x_i X^{A_{i1}}) N,$$

where i is such that A_{i1} has rank ≥ 2 . Show that Ψ works.]

- 3.6.28. Consider

1. $\lambda f : 1 \lambda x : o. f x = \lambda f : 1 \lambda x : o. f(f x);$
2. $\lambda f, g : 1 \lambda x : o. f(g(f x)) = \lambda f, g : 1 \lambda x : o. f(g(f(g x)));$

$$3. \lambda h:1_2 \lambda x:o.h(hx(hxx))(hxx) = \lambda h:1_2 \lambda x:o.h(hxx)(h(hxx)x).$$

- (i) Show that 1 holds in $\mathcal{M}_{\mathcal{C}_1}$, but not in $\mathcal{M}_{\mathcal{C}_2}$.
- (ii) Show that 2 holds in $\mathcal{M}_{\mathcal{C}_2}$, but not in $\mathcal{M}_{\mathcal{C}_3}$.
- (iii) Show that 3 holds in $\mathcal{M}_{\mathcal{C}_3}$ and $\mathcal{M}_{\mathcal{C}_4}$, but not in $\mathcal{M}_{\mathcal{C}_5}$.

3.6.29. Construct as much as possible (6 at most) pure closed terms to show that the five canonical theories are maximally different. I.e. we want terms M_1, \dots, M_6 such that in $\text{Th}(\mathcal{M}_{\mathcal{C}_5})$ the M_1, \dots, M_6 are mutually different; $M_1 = M_2$ in $\text{Th}(\mathcal{M}_{\mathcal{C}_4})$, but different from M_3, \dots, M_6 ; $M_2 = M_3$ in $\text{Th}(\mathcal{M}_{\mathcal{C}_3})$, but different from M_4, \dots, M_6 ; $M_3 = M_4$ in $\text{Th}(\mathcal{M}_{\mathcal{C}_2})$, but different from M_5, M_6 ; $M_4 = M_5$ in $\text{Th}(\mathcal{M}_{\mathcal{C}_1})$, but different from M_6 . [Hint. Use the previous exercise and a polynomially defined pairing operator. A complicating factor is that it is an open question whether the theories of $\mathcal{M}_{\mathcal{C}_3}$ and $\mathcal{M}_{\mathcal{C}_4}$ are different.]

3.6.30. (Finite generation, Joly [2002]) Let A be a type. Then A is said to be *finitely generated* if there exist types A_1, \dots, A_t and terms $M_1 : A_1, \dots, A_t : M_t$ such that for any $M : A$, M is $\beta\eta$ convertible to an applicative combination of M_1, \dots, M_t . Example. $1 \rightarrow o \rightarrow o$ is finitely generated by $\mathbf{c}_0 \equiv (\lambda f x.x) : 1 \rightarrow 1$ and $S = \lambda n f x.f(n f x) : (1 \rightarrow o \rightarrow o) \rightarrow (1 \rightarrow o \rightarrow o)$.

A *slantwise enumerates* a type B if there exists a type substitution $@$ and $F : @A \rightarrow B$ such that for each $N : B$ there exists $M : A$ such that $F @ M =_{\beta\eta} N$ (F is *surjective*).

A type A is said to be *poor* if every beta-eta normal form of type A can be written with the same finite number of bound variables. Otherwise A is said to be *rich*. Example. Let $A = (1 \rightarrow o) \rightarrow o \rightarrow o$ is poor. A typical $\beta\eta$ -nf of type A has the shape $\lambda F \lambda x(F(\lambda x(\dots(F(\lambda y(F(\lambda y \dots x \dots))))))$. One allows the term to violate the variable convention (to have different bound variables). The monster type $3 \rightarrow 1$ is rich.

The goal of this exercise is to prove that the following are equivalent.

1. A slantwise enumerates the monster type \mathbf{M}
2. The lambda definability problem for A is undecidable.
3. A is not finitely generated
4. A is rich.

However, we will not ask the reader to prove $(4) \Rightarrow (1)$ since this involves more knowledge of and practice with slantwise enumerations than one can get from this book. For that proof we refer the reader to Joly's paper. We have already shown that the lambda definability problem for the monster \mathbf{M} is undecidable. In addition, we make the following steps.

- (i) Show A is rich iff A has rank > 3 or A is large of rank 3 (for A inhabited; especially for \Rightarrow). Use this to show

$$\neg(4) \Rightarrow \neg(3) \text{ and } \neg(3) \Rightarrow \neg(2).$$

- (ii) (Alternative to show $\neg(4) \Rightarrow \neg(3)$.) Suppose that every closed term of type A beta eta converts to a special one built up from a fixed finite set of variables. Show that it suffices to bound the length of the lambda prefix of any subterm of such a special term in order to conclude finite generation. Suppose that we consider only terms X built up only from the variables $v_1:A_1, \dots, v_m:A_m$ both free and bound. We shall transform X using a fixed set of new variables. First we assume the set of A_i is closed under subtype. (a) Show that we can assume that X is fully expanded. For example, if X has the form

$$\lambda x_1 \dots x_t. (\lambda x. X_0) X_1 \dots X_s$$

then $(\lambda x. X_0) X_1 \dots X_s$ has one of the A_i as a type (just normalize and consider the type of the head variable). Thus we can eta expand

$$\lambda x_1 \dots x_t. (\lambda x. X_0) X_1 \dots X_s$$

and repeat recursively. We need only double the set of variable to do this. We do this keeping the same notation. (b) Thus given

$$X = \lambda x_1 \dots x_t. (\lambda x. X_0) X_1 \dots X_s$$

we have $X_0 = \lambda y_1 \dots y_r. Y$, where $Y : o$. Now if $r > m$, each multiple occurrence of v_i in the prefix $\lambda y_1 \dots y_r$ is dummy and those that occur in the initial segment $\lambda y_1 \dots y_s$ can be removed with the corresponding X_j . The remaining variables will be labelled z_1, \dots, z_k . The remaining X_j will be labelled Z_1, \dots, Z_l . Note that $r - s + t < m + 1$. Thus

$$X = \lambda x_1 \dots x_t. (\lambda z_1 \dots z_k Y) Z_1 \dots Z_l,$$

where $k < 2m + 1$. We can now repeat this analysis recursively on Y , and Z_1, \dots, Z_l observing that the types of these terms must be among the A_i . We have bounded the length of a prefix.

- (iii) As to (1) \Rightarrow (2). We have already shown that the lambda definability problem for the monster \mathbf{M} is undecidable. Suppose (1) and $\neg(2)$ towards a contradiction. Fix a type B and let $B(n)$ be the cardinality of B in $P(n)$. Show that for any closed terms $M, N : C$

$$P(B(n)) \models M = N \Rightarrow P(n) \models [o := B]M = [o := B]N.$$

Conclude from this that lambda definability for \mathbf{M} is decidable, which is not the case.

Chapter 4

Definability, Unification and Matching

4.1. Undecidability of lambda definability

The finite standard models

Recall that the full type structure over a set X , notation \mathcal{M}_X , is defined in Definition 2.4.18 as follows.

$$\begin{aligned} X(o) &= X, \\ X(A \rightarrow B) &= X(B)^{X(A)}; \\ \mathcal{M}_X &= \{X(A)\}_{A \in \mathbb{T}}. \end{aligned}$$

Remark that if X is finite then all the $X(A)$ are finite. In that case we can represent each element of \mathcal{M}_X by a finite piece of data and hence (through Gödel numbering) by a natural number. For instance for $X = \{0, 1\}$ we can represent the four elements of $X(o \rightarrow o)$ as follows.

0	0	0	1	0	0	1
1	0	1	1	1	1	0

Any element of the model can be expressed in a similar way, for instance the following table represents an element of $X((o \rightarrow o) \rightarrow o)$.

0	0	0
1	0	0
0	1	0
1	1	0
0	0	1
1	1	1

We know that $I \equiv \lambda x.x$ is the only closed $\beta\eta$ -nf of type $o \rightarrow o$. It is easy to prove from this that identity is the only function of $X(o \rightarrow o)$ that is denoted by a closed term.

4.1.1. DEFINITION. Let \mathcal{M} be a type structure and let $d \in \mathcal{M}(A)$. Then d is called *λ -definable* iff $\exists M \in \Lambda^\emptyset(A). d = \llbracket M \rrbracket^{\mathcal{M}}$.

The main result in this section is the undecidability of λ -definability in \mathcal{M}_X , for X of cardinality >6 . This means that there is no algorithm deciding whether a table describes a λ -definable element in this model. This result is due to Loader [2001b], and was already proved by him in 1993.

The proof proceeds by reducing the two-letter word rewriting problem, a well-known undecidable problem, to the λ -definability problem. It follows that if the λ -definability problem were decidable, then this also would be the case for the two-letter word rewriting problem, *quod non*.

4.1.2. DEFINITION (Word rewriting problem). Let $\Sigma = \{A, B\}$ be a two letter alphabet.

(i) A *word* is a finite sequence of letters $w = w_1 \dots w_n$ with $w_i \in \Sigma$. The set of words over Σ is denoted by Σ^* .

(ii) If $w = w_1 \dots w_n$, then $\text{length}(w) = n$ is called the length of w . If $\text{length}(w) = 0$, then w is called the *empty word* and is denoted by ϵ .

(iii) A *rewrite rule* is a pair of non empty words v, w denoted as $v \hookrightarrow w$.

(iv) Given a word u and a finite set $\mathcal{R} = \{R_1, \dots, R_r\}$ of rewrite rules $R_i = v_i \hookrightarrow w_i$. Then a *derivation* of a word s is a finite sequence of words starting by u finishing by s and such that each word is obtained from the previous by replacing a subword v_i by w_i for some rule $v_i \hookrightarrow w_i \in \mathcal{R}$.

(v) A word s is said to be *derivable* from u if it has a derivation. In this case we write $u \vdash_{\mathcal{R}} s$.

4.1.3. EXAMPLE. Consider the word AB and the rule $AB \hookrightarrow AABB$. Then $AB \vdash AAABBB$, but $AB \not\vdash AAB$.

We will need the following well known result, see e.g. Post [1947].

4.1.4. THEOREM. *There is a word $u_0 \in \Sigma^*$ and a finite set of rewrite rules $v_1 \hookrightarrow w_1, \dots, v_r \hookrightarrow w_r$ such that $\{u \in \Sigma^* \mid u_0 \vdash u\}$ is undecidable.*

4.1.5. DEFINITION. Given the alphabet $\Sigma = \{A, B\}$, define the set

$$X = X_\Sigma = \{*, A, B, L, R, Y, N\}.$$

The objects L and R are suggested to be read *left* and *right* and Y and N *yes* and *no*. We will consider the standard model $\mathcal{M} = \mathcal{M}_X$ built over the set X .

4.1.6. DEFINITION (Word encoding). Remember $1_n = o^n \rightarrow o$ and $MN \sim^n \equiv MN \dots N$, with n times the same term N . Let $w = (w_1 \dots w_n) \in \Sigma^*$ be a non empty word of length n .

(i) The word w is *encoded* as the object $\underline{w} \in \mathcal{M}(1_n)$ defined as follows.

$$\begin{aligned} \underline{w} *^{\sim(i-1)} w_i *^{\sim(n-i)} &= Y; \\ \underline{w} *^{\sim(i-1)} LR *^{\sim(n-i-1)} &= Y; \\ \underline{w} x_1 \dots x_n &= N, \quad \text{otherwise.} \end{aligned}$$

(ii) The word w is *weakly encoded* by an object $h \in \mathcal{M}(1_n)$ iff

$$\begin{aligned} h *^{\sim(i-1)} w_i *^{\sim(n-i)} &= Y; \\ h *^{\sim(i-1)} LR *^{\sim(n-i-1)} &= Y. \end{aligned}$$

4.1.7. DEFINITION. (Encoding of a rule) In order to define the encoding of a rule we use the notation $(a_1 \dots a_k \mapsto Y)$ to denote the element $h \in \mathcal{M}(1_k)$ defined by

$$\begin{aligned} h(a_1 \dots a_k) &= Y; \\ h(x_1 \dots x_k) &= N, \quad \text{otherwise.} \end{aligned}$$

Now a rule $v \hookrightarrow w$ where $\text{1th}v = m$ and $\text{1th}w = n$ is encoded as the object $\underline{v \hookrightarrow w} \in \mathcal{M}(1_m \rightarrow 1_n)$ defined as follows.

$$\begin{aligned} \underline{v \hookrightarrow w}(\underline{v}) &= \underline{w}; \\ \underline{v \hookrightarrow w}(*^{\sim m} \mapsto Y) &= (*^{\sim n} \mapsto Y); \\ \underline{v \hookrightarrow w}(R *^{\sim(m-1)} \mapsto Y) &= (R *^{\sim(n-1)} \mapsto Y); \\ \underline{v \hookrightarrow w}(*^{\sim(m-1)} L \mapsto Y) &= (*^{\sim(n-1)} L \mapsto Y); \\ \underline{v \hookrightarrow w}(h) &= \lambda x_1 \dots x_n. N, \quad \text{otherwise.} \end{aligned}$$

As usual we identify a term $M \in \Lambda(A)$ with its denotation $\llbracket M \rrbracket \in X(A)$.

4.1.8. LEMMA. Let s, u be two words over Σ and let $v \hookrightarrow w$ be a rule. Then $svu \vdash swu$ and

$$\underline{swu} \vec{s} \vec{w} \vec{u} = \underline{v \hookrightarrow w} (\lambda \vec{v}. \underline{svu} \vec{s} \vec{v} \vec{u}) \vec{w}, \quad (1)$$

where the $\vec{s}, \vec{u}, \vec{v}, \vec{w}$ are sequences of elements in X with lengths equal the lengths of the words s, u, v, w , say p, q, m, n , respectively.

PROOF. The RHS of (1) is obviously either Y or N . Now $\text{RHS} = Y$ iff one of the following holds

- $\lambda \vec{v}. \underline{svu} \vec{s} \vec{v} \vec{u} = \underline{v}$ and $\vec{w} = *^{\sim(i-1)} w_i *^{\sim(n-i)}$
- $\lambda \vec{v}. \underline{svu} \vec{s} \vec{v} \vec{u} = \underline{v}$ and $\vec{w} = *^{\sim(i-1)} LR *^{\sim(n-i-1)}$
- $\lambda \vec{v}. \underline{svu} \vec{s} \vec{v} \vec{u} = (*^{\sim m} \mapsto Y)$ and $\vec{w} = *^{\sim n}$
- $\lambda \vec{v}. \underline{svu} \vec{s} \vec{v} \vec{u} = (R *^{\sim(m-1)} \mapsto Y)$ and $\vec{w} = R *^{\sim(n-1)}$

- $\lambda \vec{v}. \underline{svu} \vec{s} \vec{v} \vec{u} = (*^{\sim(m-1)} L \mapsto Y)$ and $\vec{w} = *^{\sim(n-1)} L$

iff one of the following holds

- $\vec{s} = *^{\sim p}, \vec{u} = *^{\sim q}$ and $\vec{w} = *^{\sim(i-1)} w_i *^{\sim(n-i)}$
- $\vec{s} = *^{\sim p}, \vec{u} = *^{\sim q}$ and $\vec{w} = *^{\sim(i-1)} LR *^{\sim(n-i-1)}$
- $\vec{s} = *^{\sim(i-1)} s_i *^{\sim(p-i)}, \vec{u} = *^{\sim q}$ and $\vec{w} = *^{\sim n}$
- $\vec{s} = *^{\sim(i-1)} LR *^{\sim(p-i-1)}, \vec{u} = *^{\sim q}$ and $\vec{w} = *^{\sim n}$
- $\vec{s} = *^{\sim p}, \vec{u} = *^{\sim(i-1)} u_i *^{\sim(q-i)}$ and $\vec{w} = *^{\sim n}$
- $\vec{s} = *^{\sim p}, \vec{u} = *^{\sim(i-1)} LR *^{\sim(q-i-1)}$ and $\vec{w} = *^{\sim n}$
- $\vec{s} = *^{\sim p}, \vec{u} = R *^{\sim(q-1)}$ and $\vec{w} = *^{\sim(n-1)} L$
- $\vec{s} = *^{\sim(p-1)} L, \vec{u} = *^{\sim q}$ and $\vec{w} = R *^{\sim(n-1)}$

iff one of the following holds

- $\vec{s} \vec{w} \vec{u} = *^{\sim(i-1)} a_i *^{\sim(p+n+q-i)}$ and a_i is the i -th letter of swu
- $\vec{s} \vec{w} \vec{u} = * \dots * LR * \dots *$

iff $\underline{swu} \vec{s} \vec{w} \vec{u} = Y$. ■

4.1.9. PROPOSITION. Let $\mathcal{R} = \{R_1, \dots, R_r\}$ be a set of rules. Then

$$s \vdash_{\mathcal{R}} u \Rightarrow \exists F \in \Lambda^{\emptyset} \underline{u} = F \underline{s} \underline{R_1} \dots \underline{R_r}.$$

In other words, (the code of) a word s that can be produced from u and some rules is definable from the (codes) of u and the rules.

PROOF. By induction on the length of the derivation of u , using the previous lemma. ■

We now want to prove the converse of this result. We shall prove a stronger result, namely that if a word has a definable weak encoding then it is derivable.

4.1.10. CONVENTION. For the rest of this subsection we consider a fixed word W and set of rewrite rules $\mathcal{R} = \{R_1, \dots, R_k\}$ with $R_i = V_i \hookrightarrow W_i$. Moreover we let w, r_1, \dots, r_k be variables of the thypes of $\underline{W}, \underline{R_1}, \dots, \underline{R_k}$ respectively. Finally ρ is a valuation such that $\rho(w) = \underline{W}$, $\rho(r_i) = \underline{R_i}$ and $\rho(x^o) = *$ for all variables of type o .

The first lemma classifies the long normal terms whose free variables are among W, F_1, \dots, F_r and that denote a word weak encoding.

4.1.11. LEMMA. *Let M be a long normal form with $\text{FV}(M) \subseteq \{w, r_1, \dots, r_k\}$. Suppose $\llbracket M \rrbracket_\rho = \underline{V}$ for some word $V \in \Sigma^*$. Then M has one of the two following forms*

$$\begin{aligned} M &\equiv \lambda \vec{x}. w \vec{x}_1, \\ M &\equiv \lambda \vec{x}. r_i (\lambda \vec{y}. N) \vec{x}_1, \end{aligned}$$

where $\vec{x}, \vec{x}_1, \vec{y}$ are type o variables and the \vec{x}_1 are distinct elements of the \vec{x} .

PROOF. Since $\llbracket M \rrbracket_\rho$ is a weak encoding for V , the term M is of type 1_n and hence has a long normal form $M = \lambda \vec{x}. P$, with P of type o . The head variable of P is either w , some r_i or a bound variable x_i . It cannot be a bound variable, because then the term M would have the form

$$M = \lambda \vec{x}. x_i,$$

which does not denote a word weak encoding.

If the head variable of P is w then

$$M = \lambda \vec{x}. w \vec{P}.$$

The terms \vec{P} must all be among the \vec{x} . This is so because otherwise some P_j would have one of the w, \vec{r} as head variable; for all valuations this term P_j would denote Y or N , the term $w \vec{P}$ would then denote N and consequently M would not denote a weak word encoding. Moreover these variables must be distinct, as otherwise M would not denote a word weak encoding.

If the head variable of M is some r_i then

$$M = \lambda \vec{x}. r_i (\lambda \vec{y}. N) \vec{P}.$$

By the same reasoning as before it follows that the terms \vec{P} must all be among \vec{x} and different. ■

In the next four lemmas, we focus on the terms of the form

$$M = \lambda \vec{x}. r_i (\lambda \vec{y}. N) \vec{x}_1.$$

We prove that if such a term denotes a weak word encoding, then

- the variables \vec{x}_1 do not occur in $\lambda \vec{y}. N$,
- $\llbracket \lambda \vec{y}. N \rrbracket_\rho = \underline{v}_i$.
- and none of the variables \vec{x}_1 is the variable x_k .

4.1.12. LEMMA. *Let t be a long normal term of type o that is not a variable and whose free variables are among W, F_1, \dots, F_r and x_1, \dots, x_k of type o . If x_1 is free in t and there is a valuation φ such that $\varphi(x_1) = A$ or $\varphi(x_1) = B$ and $|t|_\varphi = Y$ then φ takes the value $*$ for all other variables of type o free in t .*

PROOF. By induction over the structure of t .

- If the head variable of t is W then

$$t = (Wt_1 \dots t_n)$$

The terms t_1, \dots, t_n must all be variables otherwise, some t_j would have W, F_1, \dots or F_r as head variable, $|t_j|_\varphi$ would be Y or N and the term $|t|_\varphi$ would then denote N . The variable x_1 is among these variables and if some other variable free in this term were not associated to a $*$, it would not denote Y .

- If the head variable of t is some F_i then

$$t = (F_i(\lambda \vec{w}.t')\vec{t})$$

As above, the terms \vec{t} must all be variables.

If x_1 is equal to some t_j then $|\lambda \vec{w}.t'|_\varphi$ is the word v_i , t' is not a variable and all the other variables in \vec{t} denote $*$. Let l be the first letter of v_i . We have $|\lambda \vec{w}.t'|_\varphi l * \dots * = Y$ and hence

$$|t'|_{\varphi + \langle w_1, l \rangle, \langle w_2, * \rangle, \dots, \langle w_m, * \rangle} = Y$$

hence by induction hypothesis $\varphi + \langle w_1, l \rangle, \langle w_2, * \rangle, \dots, \langle w_m, * \rangle$ takes the value $*$ on all free variables of t' but w_1 . Hence φ takes the value $*$ on all free variables of $\lambda \vec{w}.t'$. Therefore φ takes the value $*$ on all free variables of t except x_1 .

If x_1 is not among \vec{t} then it is free in $\lambda \vec{w}.t'$. Since $|(F_i(\lambda \vec{w}.t')\vec{t})|_\varphi = Y$ it follows that $|\lambda \vec{w}.t'|_\varphi$ is not the constant function equal to N hence there are objects a_1, \dots, a_m such that $|\lambda \vec{w}.t'|_\varphi(a_1) \dots (a_m) = Y$. Therefore

$$|t'|_{\varphi + \langle w_1, a_1 \rangle, \dots, \langle w_m, a_m \rangle} = Y$$

and by induction hypothesis $\varphi + \langle w_1, a_1 \rangle, \dots, \langle w_m, a_m \rangle$ takes the value $*$ on all the variables free in t' but x_1 . So φ takes the value $*$ on all the variables free in $\lambda \vec{w}.t'$ but x_1 . Moreover $a_1 = \dots = a_m = *$, and thus $|\lambda \vec{w}.t'|_\varphi * \dots * = Y$. Therefore the function $|\lambda \vec{w}.t'|_\varphi$ can only be the function mapping $* \dots *$ to Y and the other values to N . Hence $|F_i(\lambda \vec{w}.t')|_\varphi$ is the function mapping $* \dots *$ to Y and the other values to N and φ takes the value $*$ on \vec{t} . Therefore φ takes the value $*$ on all free variables of t except x_1 . ■

4.1.13. LEMMA. *If the term $t = \lambda \vec{x}(F_i(\lambda \vec{w}.t')\vec{y})$ denotes a word weak encoding, then the variables \vec{y} do not occur free in $\lambda \vec{w}.t'$ and $|\lambda \vec{w}.t'|_{\varphi_0}$ is the encoding of the word v_i .*

PROOF. Consider a variable y_j . This variable is some $x_{j'}$. Let l be the j'^{th} letter of the word w' , we have

$$|t|_{* \sim (j'-1) \ l * \sim (k-j')} = Y$$

Let $\varphi = \varphi_0 + \langle x_{j'}, l \rangle$. We have

$$f_i(|\lambda \vec{w}.t'|_\varphi) *^{\sim(j-1)} l *^{\sim(m-j)} = Y$$

Hence $|\lambda \vec{w}.t'|_\varphi$ is the encoding of the word v_i . Let l' be the first letter of this word, we have

$$|\lambda \vec{w}.t'|_\varphi(l') * \dots * = Y$$

and hence

$$|t'|_{\varphi + \langle w_1, l' \rangle, \langle w_2, * \rangle, \dots, \langle w_m, * \rangle} = Y$$

By lemma 4.1.12, $\varphi + \langle w_1, l' \rangle, \langle w_2, * \rangle, \dots, \langle w_m, * \rangle$ takes the value $*$ on all variables free in t' except w_1 . Hence y_j is not free in t' nor in $\lambda \vec{w}.t'$.

At last $|\lambda \vec{w}.t'|_\varphi$ is the encoding of v_i and y_j does not occur in it. Thus $|\lambda \vec{w}.t'|_{\varphi_0}$ is the encoding of v_i . ■

4.1.14. LEMMA. *Let t be a term of type o that is not a variable and whose free variables are among W, F_1, \dots, F_r and x_1, \dots, x_k of type o . Then there is a variable z such that for all valuations $\varphi(z) = L$ implies $|t|_\varphi = N$ or for all valuations $\varphi(z) = A$ or $\varphi(z) = B$ implies $|t|_\varphi = N$.*

PROOF. By induction over the structure of t .

- If the head variable of t is W then $t = (W\vec{t})$ the terms $\vec{t} = t_1, \dots, t_n$ must be variables and we take $z = t_n$. If $\varphi(z) = L$ then $|t|_\varphi = N$.
- If the head variable of t is F_i then $t = (F_i(\lambda \vec{w}.t')\vec{t})$. By induction hypothesis, there is a variable z' free in t' such that for all valuations $\varphi(z') = L$ implies $|t|_\varphi = N$ or for all valuations $\varphi(z') = A$ or $\varphi(z') = B$ implies $|t|_\varphi = N$.

If the variable z' is not among w_1, \dots, w_n we take $z = z'$. Either for all valuations such that $\varphi(z) = L$, $|\lambda \vec{w}.t'|_\varphi$ is the constant function equal to N and thus $|t|_\varphi = N$, or for all valuations such that $\varphi(z) = A$ or $\varphi(z) = B$, $|\lambda \vec{w}.t'|_\varphi$ is the constant function equal to N and thus $|t|_\varphi = N$.

If the variable $z' = w_j$ ($j \leq m-1$) then for all valuations $|\lambda \vec{w}.t'|_\varphi$ is a function taking the value N when applied to any sequence of arguments whose j^{th} element is L or when applied to any sequence of arguments whose j^{th} element is A or B . For all valuations, $|\lambda \vec{w}.t'|_\varphi$ is not the encoding of the word v_i and hence $|F_i(\lambda \vec{w}.t')|_\varphi$ is either the function mapping $* \dots *$ to Y and other arguments to N , the function mapping $R * \dots *$ to Y and other arguments to N , the function mapping $* \dots * L$ to Y and other arguments to N or the function mapping all arguments to N . We take $z = t_n$ and for all valuations such that $\varphi(z) = A$ or $\varphi(z) = B$ we have $|t|_\varphi = N$.

At last if $z' = w_m$ then for all valuations $|\lambda \vec{w}.t'|_\varphi$ is a function taking the value N when applied to any sequence of arguments whose m^{th} element is L or for all valuations $|\lambda \vec{w}.t'|_\varphi$ is a function taking the value N when applied to any sequence of arguments whose m^{th} element is A or B . In the first case, for all valuations,

$|\lambda\vec{w}.t'|_\varphi$ is a not the function mapping $*\dots*L$ to Y and other arguments to N . Hence $|F_i(\lambda\vec{w}.t')|_\varphi$ is either w_i or the function mapping $*\dots*$ to Y and other arguments to N the function mapping $R*\dots*$ to Y and other arguments to N or the function mapping all arguments to N . We take $z = t_n$ and for all valuations such that $\varphi(z) = A$ or $\varphi(z) = B$ we have $|t|_\varphi = N$.

In the second case, for all valuations, $|\lambda\vec{w}.t'|_\varphi$ is a not the encoding of the word v_i . Hence $|F_i(\lambda\vec{w}.t')|_\varphi$ is either the function mapping $*\dots*$ to Y and other arguments to N the function mapping $R*\dots*$ to Y and other arguments to N , the function mapping $*\dots*L$ to Y and other arguments to N or the function mapping all arguments to N . We take $z = t_n$ and for all valuations such that $\varphi(z) = L$ we have $|t|_\varphi = N$. ■

4.1.15. LEMMA. *If the term $t = \lambda\vec{x}.(F_i(\lambda w_1 \dots w_m t')\vec{y})$ denotes a word weak encoding, then none of the variables \vec{y} is the variable x_k .*

PROOF. By the lemma 4.1.14, we know that there is a variable z such that either for all valuations such that $\varphi(z) = L$ we have

$$|(F_i(\lambda\vec{w}.t')\vec{y})|_\varphi = N$$

or for all valuations such that $\varphi(z) = A$ or $\varphi(z) = B$ we have

$$|(F_i(\lambda\vec{w}.t')\vec{y})|_\varphi = N.$$

Since t denotes a word weak encoding, the only solution is that $z = x_k$ and for all valuations such that $\varphi(x_k) = L$ we have

$$|(F_i(\lambda\vec{w}.t')\vec{y})|_\varphi = N.$$

Then, if y_j were equal to x_k and y_{j+1} to some $x_{j'}$ the object

$$|(F_i(\lambda\vec{w}.t')\vec{y})|_{\varphi_0 + \langle x_k, L \rangle, \langle x_{j'}, R \rangle}$$

would be equal to $f_i(|\lambda\vec{w}.t'|_{\varphi_0}) * \dots * LR * \dots *$ and, as $|\lambda\vec{w}.t'|_{\varphi_0}$ is the encoding of the word v_i , also to Y , a contradiction.

We are now ready to conclude the proof.

4.1.16. PROPOSITION. *If there is a long normal term t whose free variables are among W, F_1, \dots, F_r that denotes a word weak encoding w' , then w' is derivable.*

PROOF. Case $t = \lambda\vec{x}.(W\vec{y})$. Then, as t denotes a word weak encoding, it depends on all its arguments and thus all the variables x_1, \dots, x_k are among \vec{y} . Since \vec{y} are distinct, \vec{y} is a permutation of x_1, \dots, x_k . As t denotes a word weak encoding, $|t| * \dots * LR * \dots * = Y$. Hence this permutation is the identity and

$$t = \lambda\vec{x}.(W\vec{x}).$$

The word w' is the word w and hence it is derivable.

Case $t = \lambda \vec{x}.(F_i(\lambda \vec{w}.t')\vec{y})$. We know that $|\lambda \vec{w}.t'|_{\varphi_0}$ is the encoding of the word v_i and thus $|(F_i(\lambda \vec{w}.t'))|_{\varphi_0}$ is the encoding of the word w_i .

Since t denotes a word weak encoding $|t| * \dots * LR * \dots * = Y$.

If some y_j ($j \leq n-1$) is the variable $x_{j'}$ then, by lemma 4.1.15, $j' \neq k$ and thus $|t| * \dots * (j'-1) LR * \dots * (k-j'-1) = Y$ and $y_{i+1} = x_{j'+1}$. Hence the variables \vec{y} are consecutive: $\vec{y} = x_{p+1}, \dots, x_{p+n}$. Call $\vec{z} = z_1, \dots, z_q$ the variables x_{p+n+1}, \dots, x_k . We have

$$t = \lambda \vec{x} \vec{y} \vec{z}.(F_i(\lambda \vec{w}.t')\vec{y})$$

We write $w' = u_1 w u_2$ (where u_1 has length p , w length n and u_2 length q).

The variables \vec{y} are not free in $\lambda \vec{w}.t'$, hence the term $\lambda \vec{x} \vec{w} \vec{z}.t'$ is closed. We verify that it denotes a weak encoding of the word $u_1 v_i u_2$.

- First clause.

- If l be the j^{th} letter of u_1 . We have

$$|\lambda \vec{x} \vec{y} \vec{z}.(F_i(\lambda \vec{w}.t')\vec{y})| * \dots * (j-1) l * \dots * (p-j+n+q) = Y$$

Let $\varphi = \varphi_0 + \langle x_j, l \rangle$. The function $|F_i(\lambda \vec{w}.t')|_{\varphi}$ maps $* \dots *$ to Y . Hence, the function $|\lambda \vec{w}.t'|_{\varphi}$ maps $* \dots *$ to Y and other arguments to N . Hence

$$|\lambda \vec{x} \vec{w} \vec{z}.t'| * \dots * (j-1) l * \dots * (p-j+m+q) = Y$$

- We know that $|\lambda \vec{w}.t'|_{\varphi_0}$ is the encoding of the word v_i . Hence if l is the j^{th} letter of the word v_i then

$$|\lambda \vec{x} \vec{w} \vec{z}.t'| * \dots * (p+j-1) l * \dots * (n-j+q) = Y$$

- In a way similar to the first case, we prove that if l is the j^{th} letter of u_2 . We have

$$|\lambda \vec{x} \vec{w} \vec{z}.t'| * \dots * (p+m+j-1) l * \dots * (q-j) = Y$$

- Second clause.

- If $j \leq p-1$, we have

$$|\lambda \vec{x} \vec{y} \vec{z}.(F_i(\lambda \vec{w}.t')\vec{y})| * \dots * (j-1) LR * \dots * (p-j-1+m+q) = Y$$

Let φ be φ_0 but x_j to L and x_{j+1} to R . The function $|F_i(\lambda \vec{w}.t')|_{\varphi}$ maps $* \dots *$ to Y . Hence, the function $|\lambda \vec{w}.t'|_{\varphi}$ maps $* \dots *$ to Y and other arguments to N and

$$|\lambda \vec{x} \vec{w} \vec{z}.t'| * \dots * (j-1) LR * \dots * (p-j-1+m+q) = Y$$

- We have

$$|\lambda\vec{x}\vec{y}\vec{z}.(F_i(\lambda\vec{w}.t')\vec{y})| *^{\sim(p-1)} LR *^{\sim(n-1+q)} = Y$$

Let φ be φ_0 but x_p to L . The function $|F_i(\lambda\vec{w}.t')|_\varphi$ maps $R * \dots *$ to Y . Hence, the function $|\lambda\vec{w}.t'|_\varphi$ maps $R * \dots *$ to Y and other arguments to N and

$$|\lambda\vec{x}\vec{w}\vec{z}.t'| *^{\sim(p-1)} LR *^{\sim(m-1+q)} = Y$$

- We know that $|\lambda\vec{w}.t'|_{\varphi_0}$ is the encoding of the word v_i . Hence if $j \leq m - 1$ then

$$|\lambda\vec{x}\vec{w}\vec{z}.t'| *^{\sim(p+j-1)} LR *^{\sim(m-j-1+q)} = Y$$

- In a way similar to the second, we prove that

$$|\lambda\vec{x}\vec{w}\vec{z}.t'| *^{\sim(p+m-1)} LR *^{\sim(q-1)} = Y$$

- In a way similar to the first, we prove that if $j \leq q - 1$, we have

$$|\lambda\vec{x}\vec{w}\vec{z}.t'| *^{\sim(p+m+j-1)} LR *^{\sim(q-j-1)} = Y$$

Hence the term $\lambda\vec{x}\vec{w}\vec{z}.t'$ denotes a weak encoding of the word $u_1v_iu_2$. By induction hypothesis, the word $u_1v_iu_2$ is derivable and hence $u_1w_iu_2$ is derivable.

At last we prove that $w = w_i$, i.e. that $w' = u_1w_iu_2$. We know that $|F_i(\lambda\vec{w}.t')|_{\varphi_0}$ is the encoding of the word w_i . Hence

$$|\lambda\vec{x}\vec{y}\vec{z}.(F_i(\lambda\vec{w}.t')\vec{y})| *^{\sim(p+j-1)} l *^{\sim(n-j+q)} = Y$$

if and only if l is the j^{th} letter of the word w_i .

Since $|\lambda\vec{x}\vec{y}\vec{z}.(F_i(\lambda\vec{w}.t')\vec{y})|$ is a weak encoding of the word u_1wu_2 , if l is the j^{th} letter of the word w , we have

$$|\lambda\vec{x}\vec{y}\vec{z}.(F_i(\lambda\vec{w}.t')\vec{y})| *^{\sim(p+j-1)} l *^{\sim(n-j+q)} = Y$$

and l is the j^{th} letter of the word w_i . Hence $w = w_i$ and $w' = u_1w_iu_2$ is derivable. ■

From proposition 4.1.9 and 4.1.16, we conclude.

4.1.17. PROPOSITION. *The word w' is derivable if and only if there is a term whose free variables are among W, F_1, \dots, F_r that denotes the encoding of w' .*

4.1.18. COROLLARY. *Let w and w' be two words and $v_1 \hookrightarrow w_1, \dots, v_r \hookrightarrow w_r$ be rewrite rules. Let h be the encoding of w , h' be the encoding of w' , f_1 be the encoding of $v_1 \hookrightarrow w_1, \dots, f_n$ be the encoding of $v_r \hookrightarrow w_r$.*

The word w' is derivable from w with the rules $v_1 \hookrightarrow w_1, \dots, v_r \hookrightarrow w_r$ if and only if there is a definable function that maps h, f_1, \dots, f_n to h' .

4.1.19. THEOREM. (Loader) λ -definability is undecidable, i.e. there is no algorithm deciding whether a table describes a λ -definable element of the model.

PROOF. If there were a algorithm to decide if a function is definable or not, then a generate and test algorithm would permit to decide if there is a definable function that maps h, f_1, \dots, f_n to h' and hence if w' is derivable from w with the rules $v_1 \hookrightarrow w_1, \dots, v_r \hookrightarrow w_r$ contradicting the undecidability of the word rewriting problem. ■

Joly has extended Loader's result in two directions as follows. Let $\mathcal{M}_n = \mathcal{M}_{\{0, \dots, n-1\}}$. Define for $n \in \mathbb{N}$, $A \in \mathbb{T}$, $d \in \mathcal{M}_n(A)$

$$D(n, A, d) \iff d \text{ is } \lambda\text{-definable in } \mathcal{M}_n.$$

Since for a fixed n_0 and A_0 the set $\mathcal{M}_{n_0}(A_0)$ is finite, it follows that $D(n_0, A_0, d)$ as predicate in d is decidable. One has the following.

4.1.20. PROPOSITION. Undecidability of λ -definability is monotonic in the following sense.

$$\lambda Ad.D(n_0, A, d) \text{ undecidable} \ \& \ n_0 \leq n_1 \Rightarrow \lambda Ad.D(n_1, A, d) \text{ undecidable}.$$

PROOF. Use Exercise 3.6.17(i). ■

Loader's proof above shows in fact that $\lambda Ad.D(7, A, d)$ is undecidable. It was sharpened in Loader [2001a] showing that $\lambda Ad.D(3, A, d)$ is undecidable.

4.1.21. THEOREM (Joly [2005]). $\lambda Ad.D(2, A, d)$ is undecidable.

4.1.22. THEOREM (Joly [2005]). $\lambda nd.P(n, 3 \rightarrow o \rightarrow o, d)$ is undecidable.

Loosely speaking one can say that λ -definability at the monster type $M = 3 \rightarrow 1$ is undecidable. Moreover, Joly also has characterised those types A that are undecidable in this sense.

4.1.23. DEFINITION. A type A is called *finitely generated* iff there are closed terms M_1, \dots, M_n , not necessarily of type A such that every closed term of type A is an applicative product of the M_1, \dots, M_n .

4.1.24. THEOREM (Joly [2002]). Let $A \in \mathbb{T}$. Then $\lambda nd.D(n, A, d)$ is decidable iff the closed terms of type A can be finitely generated.

For a sketch of the proof see Exercise 3.6.30.

4.1.25. COROLLARY. The monster type $M = 3 \rightarrow o \rightarrow o$ is not finitely generated.

PROOF. By Theorems 4.1.24 and 4.1.22. ■

4.2. Undecidability of Unification

4.2.1. DEFINITION. (i) Let $M, N \in \Lambda^\emptyset(A \rightarrow B)$. A *pure unification problem* is of the form

$$\exists X \ A.MX = NX,$$

where one searches for an $X \in \Lambda^\emptyset(A)$ (and the equality is $=_{\beta\eta}$). A is called the *search-type* and B the *output-type* of the problem.

(ii) Let $M \in \Lambda^\emptyset(A \rightarrow B), N \in \Lambda^\emptyset(B)$. A *pure matching problem* is of the form

$$\exists X \ A.MX = N,$$

where one searches for an $X \in \Lambda^\emptyset(A)$. Again A, B are the search- and output types, respectively.

(iii) Often we write for a unification or matching problem (when the types are known from the context or are not relevant) simply

$$MX = NX$$

or

$$MX = N.$$

and speak about the unification (matching) problem with *unknown* X .

Of course matching problems are a particular case of unification problems: solving the matching problem $MX = N$ amounts to solving the unification problem $MX = (\lambda x.N)X$.

4.2.2. DEFINITION. The *rank* (*order*) of a unification or matching problem is $\text{rk}(A)$ ($\text{ord}(A)$ respectively), where A is the search-type. Remember that $\text{rk}(A) = \text{ord}(A) + 1$.

The rank of the output-type is less relevant. Basically one may assume that it is $\top = 1_2 \rightarrow o \rightarrow o$. Indeed, if $\Phi : B \leq \top$ then

$$MX = NX : B \iff (\Phi \circ M)X = (\Phi \circ N)X : \top.$$

One has $\text{rk}(\top) = 2$. The unification and matching problems with an output type of rank < 2 are decidable, see Exercise 4.5.7.

The main results of this section are that unification in general is undecidable from a low level onward (Goldfarb) and matching up to order 4 is decidable (Padovani). It is an open problem whether matching in general is decidable. As a spin-off of the study of matching problems it will be shown that the maximal theory is decidable.

4.2.3. EXAMPLE. The following are two examples of pure unification problems.

- (i) $\exists X \ (1 \rightarrow o). \lambda f \ 1.f(Xf) = X.$
- (ii) $\exists X \ (1 \rightarrow o \rightarrow o). \lambda f a.X(Xf)a = \lambda f a.Xf(Xfa).$

This is not in the format of the previous Definition, but we mean of course

$$\begin{aligned} (\lambda x (1 \rightarrow o) \lambda f 1.f(xf))X &= (\lambda x (1 \rightarrow o) \lambda f 1.xf)X; \\ (\lambda x : (1 \rightarrow o \rightarrow o) \lambda f 1 \lambda a o.x(xf)a)X &= (\lambda x : (1 \rightarrow o \rightarrow o) \lambda f 1 \lambda a o.xf(xfa))X. \end{aligned}$$

The most understandable form is as follows (provided we remember the types)

$$\begin{aligned} \text{(i)} \quad \lambda f.f(Xf) &= X; \\ \text{(ii)} \quad X(Xf)a &= Xf(Xfa). \end{aligned}$$

The first problem has no solution, because there is no fixed-point combinator in λ_{\rightarrow}^o . The second one does ($\lambda fa.f(fa)$ and $\lambda fa.a$), because $n^2 = 2n$ for $n \in \{2, 4\}$.

4.2.4. EXAMPLE. The following are two pure matching problems.

$$\begin{aligned} (X(Xf)a &= f^{10}a & X 1 \rightarrow o \rightarrow o; f 1, a o; \\ f(X(Xf)a &= f^{10}a & X 1 \rightarrow o \rightarrow o; f 1, a o. \end{aligned}$$

The first problem is without a solution, because $\sqrt{10} \notin \mathbb{N}$. The second with a solution ($X \equiv \lambda fa.f^3a$), because $3^2 + 1 = 10$.

Now the unification and matching problems will be generalized. First of all we will consider more unknowns. Then more equations. Finally, in the general versions of unification and matching problems one does not require that the $\vec{M}, \vec{N}, \vec{X}$ are closed but they may contain a fixed finite number of constants (free variables). All these generalized problems will be reducible to the pure case, but (only in the transition from non-pure to pure problems) at the cost of possibly raising the rank (order) of the problem.

4.2.5. DEFINITION. (i) Let M, N be closed terms of the same type. A *pure unification problem with several unknowns*

$$M\vec{X} =_{\beta\eta} N\vec{X} \tag{1}$$

searches for closed terms \vec{X} of the right type satisfying (1). The rank of a problem with several unknowns \vec{X} is

$$\max\{\text{rk}(A_i) \mid 1 \leq i \leq n\},$$

where the A_i are the types of the X_i . The order is defined similarly.

(ii) A *system of pure unification problems* starts with terms M_1, \dots, M_n and N_1, \dots, N_n such that M_i, N_i are of the same type for $1 \leq i \leq n$. searching for closed terms $\vec{X}_1, \dots, \vec{X}_n$ all occurring among \vec{X} such that

$$\begin{aligned} M_1\vec{X}_1 &=_{\beta\eta} N_1\vec{X}_1 \\ &\dots \\ M_n\vec{X}_n &=_{\beta\eta} N_n\vec{X}_n \end{aligned}$$

The rank (order) of such a system of problems the maximum of the ranks (orders) of the types of the unknowns.

(iii) In the general (non-pure) case it will also be allowed to have the M, N, \vec{X} range over Λ^Γ rather than Λ^\emptyset . We call this a unification problem *with constants from Γ* . The rank of a non-pure system of unknowns is defined as the maximum of the rank (orders) of the types of the unknowns.

(iv) The same generalizations are made to the matching problems.

4.2.6. EXAMPLE. A pure system of matching problem in the unknowns P, P_1, P_2 is the following. It states the existence of a pairing and is solvable depending on the types involved, see Barendregt [1974].

$$\begin{aligned} P_1(Pxy) &= x \\ P_2(Pxy) &= y. \end{aligned}$$

One could add a third equation (for surjectivity of the pairing)

$$P(P_1z)(P_2z) = z,$$

causing this system never to have solutions, see Barendregt [1974].

4.2.7. EXAMPLE. An example of a unification problem with constants from $\Gamma = \{a1, b1\}$ is the following. We search for unknowns $W, X, Y, Z \in \Lambda^\Gamma(1)$ such that

$$\begin{aligned} X &= Y \circ W \circ Y \\ b \circ W &= W \circ b \\ W \circ W &= b \circ W \circ b \\ a \circ Y &= Y \circ a \\ X \circ X &= Z \circ b \circ b \circ a \circ a \circ b \circ b \circ Z, \end{aligned}$$

where $f \circ g = \lambda x.f(gx)$ for $f, g \neq 1$, having as unique solution $W = b$, $X = a \circ b \circ b \circ a$, $Y = Z = a$. This example will be expanded in Exercise 4.5.6.

4.2.8. PROPOSITION. *All unification (matching) problems reduce to pure ones with just one unknown and one equation. In fact we have the following.*

(i) *A problem of rank k with several unknowns can be reduced to a problem with one unknown with rank $\text{rk}(A) = \max\{k, 2\}$.*

(ii) *Systems of problems can be reduced to one problem, without altering the rank. The rank of the output type will be $\max\{\text{rk}(B_i), 2\}$, where B_i are the output types of the respective problems in the system.*

(iii) *Non-pure problems with constants from Γ can be reduced to pure problems. In this process a problem of rank k becomes of rank*

$$\max\{\text{rk}(\Gamma), k\}.$$

PROOF. We give the proof for unification.

(i) In the notation of Definition 1.3.19 we have

$$\exists \vec{X}. M\vec{X} = N\vec{X} \quad (1)$$

$$\iff \exists X. (\lambda x. M(x \cdot 1) \dots (x \cdot n))X = (\lambda x. N(x \cdot 1) \dots (x \cdot n))X. \quad (2)$$

Indeed, if the \vec{X} work for (1), then $X \equiv \langle \vec{X} \rangle$ works for (2). Conversely, if X works for (2), then $\vec{X} \equiv X \cdot 1, \dots, X \cdot n$ work for (1). By Proposition 5.2 we have $A = A_1 \times \dots \times A_n$ is the type of X and $\text{rk}(A) = \max\{\text{rk}(A_1), \dots, \text{rk}(A_n), 2\}$.

(ii) Similarly for $\vec{X}_1, \dots, \vec{X}_n$ being subsequences of \vec{X} one has

$$\exists \vec{X} \quad M_1 \vec{X}_1 = N_1 \vec{X}_1$$

...

$$M_n \vec{X}_n = N_n \vec{X}_n$$

$$\iff \exists \vec{X} \quad (\lambda \vec{x}. \langle M_1 \vec{x}_1, \dots, M_n \vec{x}_n \rangle) \vec{X} = (\lambda \vec{x}. \langle N_1 \vec{x}_1, \dots, N_n \vec{x}_n \rangle) \vec{X}.$$

(iii) Write a non-pure problem with $M, N \in \Lambda^\Gamma(A \rightarrow B)$, and $\text{dom}(\Gamma) = \{\vec{y}\}$ as

$$\exists X[\vec{y}] \quad A.M[\vec{y}]X[\vec{y}] = N[\vec{y}]X[\vec{y}].$$

This is equivalent to the pure problem

$$\exists X \left(\bigwedge \Gamma \rightarrow A \right). (\lambda x \vec{y}. M[\vec{y}](x \vec{y}))X = (\lambda x \vec{y}. N[\vec{y}](x \vec{y}))X. \blacksquare$$

Although the ‘generalized’ unification and matching problems all can be reduced to the pure case with one unknown and one equation, one usually should not do this if one wants to get the right feel for the question.

Decidable case of unification

4.2.9. PROPOSITION. *Unification with unknowns of type 1 and constants of types 0, 1 is decidable.*

PROOF. The essential work to be done is the solvability of Markov’s problem by Makanin. See Exercise 4.5.6 for the connection and a reference. ■

Undecidability of unification

The undecidability of unification was first proved by Huet. This was done before the undecidability of Hilbert’s 10-th problem (Is it decidable whether an arbitrary Diophantine equation over \mathbb{Z} is solvable?) was established. Huet reduced Post’s correspondence problem to the unification problem. The theorem by Matijasevic makes things more easy.

4.2.10. THEOREM (Matijasevič). (i) *There are two polynomials p_1, p_2 over \mathbb{N} (of degree 7 with 13 variables¹) such that*

$$D = \{\vec{n} \in \mathbb{N} \mid \exists \vec{x} \in \mathbb{N}. p_1(\vec{n}, \vec{x}) = p_2(\vec{n}, \vec{x})\}$$

is undecidable.

¹This can be pushed to polynomials of degree 4 and 58 variables or of degree $1.6 \cdot 10^{45}$ and 9 variables, see Jones [1982].

(ii) *There is a polynomial $p(\vec{x}, \vec{y})$ over \mathbb{Z} such that*

$$D = \{\vec{n} \in \mathbb{N} \mid \exists \vec{x} \in \mathbb{Z}. p(\vec{n}, \vec{x}) = 0\}$$

is undecidable. Therefore Hilbert's 10-th problem is undecidable.

PROOF. (i) This was done by coding arbitrary RE sets as Diophantine sets of the form D . See Matijasevič [1971], Davis [1973] or Matiyasevič [1993].

(ii) Take $p = p_1 - p_2$ with the p_1, p_2 from (i). Using the theorem of Lagrange

$$\forall n \in \mathbb{N} \exists a, b, c, d \in \mathbb{N}. n = a^2 + b^2 + c^2 + d^2,$$

it follows that for $n \in \mathbb{Z}$ one has

$$n \in \mathbb{N} \iff \exists a, b, c, d \in \mathbb{N}. n = a^2 + b^2 + c^2 + d^2.$$

Finally write $\exists x \in \mathbb{N}. p(x, \dots) = 0$ as $\exists a, b, c, d \in \mathbb{Z}. p(a^2 + b^2 + c^2 + d^2, \dots) = 0$. ■

4.2.11. COROLLARY. *The solvability of pure unification problems of order 3 (rank 2) is undecidable.*

PROOF. Take the two polynomials p_1, p_2 and D from (i) of the theorem. Find closed terms M_{p_1}, M_{p_2} representing the polynomials, as in Corollary 1.3.5. Let $U_{\vec{n}} = \{M_{p_1} \ulcorner \vec{n} \urcorner \vec{x} = M_{p_2} \ulcorner \vec{n} \urcorner \vec{x}\}$. Using that every $X \in \Lambda^\emptyset(\mathbf{Nat})$ is a numeral, Proposition 2.1.22, it follows that this unification problem is solvable iff $\vec{n} \in D$. ■

The construction of Matijasevic is involved. The encoding of Post's correspondence problem by Huet is a more natural way to show the undecidability of unification. It has as disadvantage that needs to use unification at variable types. There is a way out. In Davis et al. [1960] it is proved that every RE predicate is of the form $\exists \vec{x} \forall y_1 < t_1 \dots \forall y_n < t_n. p_1 = p_2$. Using this result and higher types (\mathbf{Nat}_A , for some non-atomic A) one can get rid of the bounded quantifiers. The analogon of Proposition 2.1.22 ($X \mathbf{Nat} \Rightarrow X$ a numeral) does not hold but one can filter out the 'numerals' by a unification (with $f A \rightarrow A$):

$$f \circ (Xf) = (Xf) \circ f.$$

This yields without Matijasevic's theorem that unification with for the unknown a fixed type is undecidable.

4.2.12. THEOREM. *Unification of order 2 (rank 1) with constants is undecidable.*

PROOF. See Exercise 4.5.4. ■

This implies that pure unification of order 3 is undecidable, something we already saw in Corollary 4.2.11. The interest in this result comes from the fact that unification over order 2 variables plays a role in automated deduction and the undecidability of this problem, being a subcase of a more general situation, is not implied by Corollary 4.2.11.

4.3. Decidability of matching of rank 3

The main result will be that matching of rank 3 (which is the same as order 4) is decidable and is due to Padovani [2000]. On the other hand Loader [2003] has proved that general matching modulo $=_\beta$ is undecidable. The decidability of general matching modulo $=_{\beta\eta}$, which is the intended case, remains open.

The structure of this section is as follows. First the notion of interpolation problem is introduced. Then by using tree automata it is shown that these problems restricted to rank 3 are decidable. Then at rank 3 the problem of matching is reduced to interpolation and hence solvable. At rank 1 matching with several unknowns is already NP-complete.

4.3.1. PROPOSITION. (i) *Matching with unknowns of rank 1 is NP-complete.*

(ii) *Pure matching of rank 2 is NP-complete.*

PROOF. (i) Consider $A = o^2 \rightarrow o = \text{Bool}_o$. Using Theorem 2.1.19, Proposition 1.2.3 and Example 1.2.8 it is easy to show that if $M \in \Lambda^\emptyset(A)$, then $M \in \beta\eta\{\text{true}, \text{false}\}$ [We should have something better: be able to refer to ONE result]. By Proposition 1.3.2 a Boolean function $p(X_1, \dots, X_n)$ in the variables X_1, \dots, X_n is λ -definable by a term $M_p \in \Lambda^\emptyset(A^n \rightarrow A)$. Therefore

$$p \text{ is satisfiable} \iff M_p X_1 \dots X_n = \text{true} \text{ is solvable.}$$

This is a matching problem of rank 1.

(ii) By (i) and Proposition 4.2.8. ■

In this chapter, we prove the decidability of fourth-order matching. A higher-order matching problem is a set of equations $t = u$ such that t contains both variables and constants and u contains only constants. A solution of such a problem is a substitution σ such that, for each equation, $\sigma t =_{\beta\eta} u$. A matching problem is said to be of rank n if all the variables of t have at most rank n .

Following an idea of Statman [1982], the decidability of the matching problem can be reduced to the existence for every term u of a logical relation \parallel_u on terms λ_{\rightarrow}^o such that

- \parallel_u is an equivalence relation;
- for all types T the quotient $\mathcal{T}_T / \parallel_u$ is finite;
- there is an algorithm that enumerates $\mathcal{T}_T / \parallel_u$, i.e. that takes in argument a type T and returns a finite sequence of terms representing all the classes.

Indeed, if such a relation exists, then a simple generate and test algorithm permits to solve the higher-order matching problem.

Similarly the decidability of the matching problem of rank n can be reduced to the existence of a relation such that $\mathcal{T}_T / \parallel_u$ can be enumerated up to rank n .

The finite completeness theorem yields the existence of a standard model \mathcal{M} such that the relation $\mathcal{M} \models t = u$ meets the two first requirements, but Loader's theorem shows that it does not meet the third.

Padovani has proposed another relation - the *relative observational equivalence* - that is enumerable up to order 4. Like in the construction of the finite completeness theorem, the relative observational equivalence relation identifies terms of type o that are $\beta\eta$ -equivalent and also all terms of type o that are not subterms of u . But this relation disregards the result of the application of a term to a non definable element.

Padovani has proved that the enumerability of this relation up to rank n can be reduced to the decidability of a variant of the matching problem of rank n : the *dual interpolation problem* of rank n . Interpolation problems have been introduced in Dowek [1994] as a first step toward decidability of third-order matching. The decidability of the dual interpolation problem of order 4 has been also proved by Padovani. However, here we shall not present the original proof, but a simpler one proposed by Comon and Jurski Comon and Jurski [1998].

Rank 3 interpolation problems

An *interpolation equation* is an equation is a particular matching problem

$$X \vec{t} = u,$$

where t_1, \dots, t_n and u are closed terms. That is, the unknown X occurs at the head. A solution of such an equation is a term v such that

$$v \vec{t} =_{\beta\eta} u.$$

An *interpolation problem* is a conjunction of such equations with the same unknown. A solution of such a problem is a term v that is a solution for all the equations simultaneously. A *dual interpolation problem* is a conjunction of equations and negated equations. A solution of such a problem is a term solution of all the equations but solution of none of the negated equations. If a dual interpolation problem has a solution it has also a closed solution in Inf . Hence, without loss of generality, we can restrict the search to such terms.

To prove the decidability of the fourth-order dual interpolation problem, we shall prove that the solutions of an interpolation equation can be recognized by a finite tree automaton. Then, the results will follow from the decidability of the non-emptiness of a set of terms recognized by a finite tree automaton and the closure of recognizable sets of terms by intersection and complement.

Relevant solution

In fact, it is not exactly quite so that the solutions of a fourth-order interpolation equation can be recognized by a finite state automaton. Indeed, a solutions of an

interpolation equation may contain an arbitrary number of variables. For instance the equation

$$XK = a$$

where X is a variable of type $(o \rightarrow 1 \rightarrow o) \rightarrow o$ has all the solutions

$$\lambda f.(fa(\lambda z_1.(fa(\lambda z_2.(fa...(\lambda z_n(fz_1(\lambda y.(fz_2(\lambda y.(fz_3... (fz_n(\lambda y.a))..)))))).))).)).$$

Moreover since each z_i has z_1, \dots, z_{i-1} in its scope it is not possible to rename these bound variables so that the variables of all these solutions are in a fixed finite set.

Thus the language of the solution cannot be *a priori* limited. In this example, it is clear however that there is another solution

$$\lambda f.(f a \square)$$

where \square is a new constant of type $o \rightarrow o$. Moreover all the solutions above can be retrieved from this one by replacing the constant \square by an appropriate term (allowing captures in this replacement).

4.3.2. DEFINITION. For each simple type T , we consider a constant \square_T . Let t be a term solution of an interpolation equation. A subterm occurrence of t of type T is *irrelevant* if replacing it by the constant \square_T yields a solution. A *relevant* solution is a closed solution where all irrelevant subterm occurrences are the constant \square_T .

Now we prove that relevant solutions of an interpolation equations can be recognized by a finite tree automaton.

An example

Consider the problem

$$Xc_1 = ha$$

where X is a variable of type $(1 \rightarrow o \rightarrow o) \rightarrow o$ and a and h are constants of type o and 1_2 . A relevant solution of this equation substitutes X by the term $\lambda f.v$ where v is a relevant solution of the equation $v[f := c_1] = ha$.

Let \mathcal{Q}_{ha} be the set of the relevant solutions v of the equation $v[f := c_1] = ha$. More generally, let \mathcal{Q}_w be the set of relevant solutions v of the equation $v[f := c_1] = w$.

Notice that terms in \mathcal{Q}_w can only contain the constants and the free variables that occur in w , plus the variable f and the constants \square_T . We can determine membership of such a set (and in particular to \mathcal{Q}_{ha}) by induction over the structure of a term.

- *analysis of membership to \mathcal{Q}_{ha}*

A term is in \mathcal{Q}_{ha} if it has either the form (hv_1) and v_1 is in \mathcal{Q}_a or the form (fv_1v_2) and $(v_1[f := c_1]v_2[f := c_1]) = ha$. This means that there are terms v'_1 and v'_2 such that $v_1[f := c_1] = v'_1$, $v_2[f := c_1] = v'_2$ and $(v'_1v'_2) = ha$, in other words there are terms v'_1 and v'_2 such that v_1 is in $\mathcal{Q}_{v'_1}$, v_2 is in $\mathcal{Q}_{v'_2}$ and $(v'_1v'_2) = ha$. As $(v'_1v'_2) = ha$

there are three possibilities for v'_1 and v'_2 : $v'_1 = \mathbf{l}$ and $v'_2 = ha$, $v'_1 = \lambda z.hz$ and $v'_2 = a$ and $v'_1 = \lambda z.ha$ and $v'_2 = \square_o$. Hence (fv_1v_2) is in \mathcal{Q}_{ha} if either v_1 is in $\mathcal{Q}_\mathbf{l}$ and v_2 in \mathcal{Q}_{ha} or v_1 is in $\mathcal{Q}_{\lambda z.hz}$ and v_2 in \mathcal{Q}_a or v_1 is in $\mathcal{Q}_{\lambda z.ha}$ and $v_2 = \square_o$.

Hence, we have to analyze membership to \mathcal{Q}_a , $\mathcal{Q}_\mathbf{l}$, $\mathcal{Q}_{\lambda z.hz}$, $\mathcal{Q}_{\lambda z.ha}$.

- *analysis of membership to \mathcal{Q}_a*

A term is in \mathcal{Q}_a if it has either the form a or the form (fv_1v_2) and v_1 is in $\mathcal{Q}_\mathbf{l}$ and v_2 is in \mathcal{Q}_a or v_1 in $\mathcal{Q}_{\lambda z.a}$ and $v_2 = \square_o$.

Hence, we have to analyze membership to $\mathcal{Q}_{\lambda z.a}$,

- *analysis of membership to $\mathcal{Q}_\mathbf{l}$*

A term is in $\mathcal{Q}_\mathbf{l}$ if it has the form $\lambda z.v_1$ and v_1 is in \mathcal{Q}_z

Hence, we have to analyze membership to \mathcal{Q}_z .

- *analysis of membership to $\mathcal{Q}_{\lambda z.hz}$*

A term is in $\mathcal{Q}_{\lambda z.hz}$ if it has the form $\lambda z.v_1$ and v_1 is in \mathcal{Q}_{hz}

Hence, we have to analyze membership to \mathcal{Q}_{hz} .

- *analysis of membership to $\mathcal{Q}_{\lambda z.ha}$*

A term is in $\mathcal{Q}_{\lambda z.ha}$ if it has the form $\lambda z.v_1$ and v_1 is in \mathcal{Q}_{ha} .

- *analysis of membership to $\mathcal{Q}_{\lambda z.a}$*

A term is in $\mathcal{Q}_{\lambda z.a}$ if it has the form $\lambda z.v_1$ and v_1 is in \mathcal{Q}_a .

- *analysis of membership to \mathcal{Q}_z*

A term is in \mathcal{Q}_z if it has the form z or the form (fv_1v_2) and either v_1 is in $\mathcal{Q}_\mathbf{l}$ and v_2 is in \mathcal{Q}_z or v_1 is in $\mathcal{Q}_{\lambda z'.z}$ and $v_2 = \square_o$.

Hence, we have to analyze membership to $\mathcal{Q}_{\lambda z'.z}$.

- *analysis of membership to \mathcal{Q}_{hz}*

A term is in \mathcal{Q}_{hz} if it has the form (hv_1) and v_1 is in \mathcal{Q}_z or the form (fv_1v_2) and either v_1 is in $\mathcal{Q}_\mathbf{l}$ and v_2 is in \mathcal{Q}_{hz} or v_1 is in $\mathcal{Q}_{\lambda z.hz}$ and v_2 is in \mathcal{Q}_z or v_1 is in $\mathcal{Q}_{\lambda z'.hz}$ and $v_2 = \square_o$.

Hence, we have to analyze membership to $\mathcal{Q}_{\lambda z'.hz}$.

- *analysis of membership to $\mathcal{Q}_{\lambda z'.z}$*

A term is in $\mathcal{Q}_{\lambda z'.z}$ if it has the form $\lambda z'.v_1$ and v_1 is in \mathcal{Q}_z .

- *analysis of membership to $\mathcal{Q}_{\lambda z'.hz}$*

A term is in $\mathcal{Q}_{\lambda z'.hz}$ if it has the form $\lambda z'.v_1$ and v_1 is in \mathcal{Q}_{hz} .

In this way we can build an automaton that recognizes in q_w the terms of \mathcal{Q}_w .

$$\begin{aligned}
(hq_a) &\rightarrow q_{ha} \\
(fq_l q_{ha}) &\rightarrow q_{ha} \\
(fq_{\lambda z.hz} q_a) &\rightarrow q_{ha} \\
(fq_{\lambda z.ha} q_{\square_o}) &\rightarrow q_{ha} \\
a &\rightarrow q_a \\
(fq_l q_a) &\rightarrow q_a \\
(fq_{\lambda z.a} q_{\square_o}) &\rightarrow q_a \\
\lambda z.q_z &\rightarrow q_l \\
\lambda z.q_{hz} &\rightarrow q_{\lambda z.hz} \\
\lambda z.q_{ha} &\rightarrow q_{\lambda z.ha} \\
\lambda z.q_a &\rightarrow q_{\lambda z.a} \\
z &\rightarrow q_z \\
(fq_l q_z) &\rightarrow q_z \\
(fq_{\lambda z'.z} q_{\square_o}) &\rightarrow q_z \\
(hq_z) &\rightarrow q_{hz} \\
(fq_l q_{hz}) &\rightarrow q_{hz} \\
(fq_{\lambda z.hz} q_z) &\rightarrow q_{hz} \\
(fq_{\lambda z'.hz} q_{\square_o}) &\rightarrow q_{hz} \\
\lambda z'.q_z &\rightarrow q_{\lambda z'.z} \\
\lambda z'.q_{hz} &\rightarrow q_{\lambda z'.hz}
\end{aligned}$$

Then we need a rule that permits to recognize \square_o in the state q_{\square_o}

$$\square_o \rightarrow q_{\square_o}$$

and at last a rule that permits to recognize in q_0 the relevant solution of the equation $(X\mathbf{c}_1) = ha$

$$\lambda f.q_{ha} \rightarrow q_0$$

Notice that as a spin off we have proved that besides f all relevant solutions of this problem can be expressed with two bound variables z and z' .

The states of this automaton are labeled by the terms ha , a , l , $\lambda z.a$, $\lambda z.hz$, $\lambda z.ha$, z , hz , $\lambda z'.z$ and $\lambda z'.hz$. All these terms have the form

$$u = \lambda y_1 \dots \lambda y_p.C$$

where C is a context of a subterm of ha and the free variables of C are in the set $\{z, z'\}$.

Tree automata for relevant solutions

Let t be a normal term and V be a set of k variables of type o not occurring in t where k is the size of t . A *context* of t is a term C such that there exists a substitution σ mapping the variables of V to terms of type o such that $\sigma C = t$.

Consider an equation

$$X\vec{t} = u$$

where X is a variable of fourth-order type at most. Consider a finite number of constants \square_T for each type T of a subterm of u . Let k be the size of u . Consider a fixed set V of k variables of type o . Let N be the finite set of term of the form $\lambda y_1 \dots \lambda y_p C$ where C is a context of a subterm of u and the free variables of C are in V . We define a tree automaton with the states q_w for w in N and q_{\square_T} for each constant \square_T , and the transitions

- $(f_i q_{w_1} \dots q_{w_n}) \rightarrow q_w$ if $(t_i \vec{w}) = w$ and replacing a w_i different from \square_T by a \square_T does not yield a solution,
- $(h q_{u_1} \dots q_{u_n}) \rightarrow q_{(hu_1 \dots u_n)}$ (for u_1, \dots, u_n in N)
- $\square_T \rightarrow q_{\square_T}$
- $\lambda z. q_t \rightarrow q_{\lambda z. t}$,
- $\lambda f_1 \dots \lambda f_n. q_u \rightarrow q_0$

4.3.3. PROPOSITION. *Let a and b be two elements of N and X_1, \dots, X_n be variables of order at most two. Let σ be a relevant solution of the second-order matching problem*

$$(aX_1 \dots X_n) = b$$

then for each i , either σX_i is in N (modulo alpha-conversion) or is equal to \square_T .

PROOF. Let a' be the normal form of $(a\sigma X_1 \dots \sigma X_{i-1} X_i \sigma X_{i+1} \dots \sigma X_n)$. If X_i has no occurrence in a' then as σ is relevant $\sigma X_i = \square_T$.

Otherwise consider the higher occurrence l of a subterm of type o of a' that has the form $(X_i v_1 \dots v_p)$. The terms v_1, \dots, v_p have type o . Let b' be the subterm of b at the same occurrence l . The term b' has type o , it is a context of a subterm of u .

Let v'_i be the normal form of $v_i[\sigma X_i / X_i]$. We have $(\sigma X_i v'_1 \dots v'_p) = b'$. Consider p variables y_1, \dots, y_p of V that not free in b' . We have $\sigma X_i = \lambda y_1 \dots \lambda y_p C$ and $C[v'_1 / y_1, \dots, v'_p / y_p] = b'$. Hence C is a context of a subterm of u and $\sigma X_i = \lambda y_1 \dots \lambda y_p. C$ is an element of N . ■

4.3.4. REMARK. As a corollary of proposition 4.3.3, we get an alternative proof of the decidability of second-order matching.

4.3.5. PROPOSITION. *Let*

$$X\vec{t} = u$$

be an equation, and \mathcal{A} the associated automaton. Then a term is recognized by \mathcal{A} (in q_0) if and only if it is a relevant solution of this equation.

PROOF. We want to prove that a term v is recognized in q_0 if and only if it is a relevant solution of the equation $v \vec{t} = u$. It is sufficient to prove that v is recognized in the state q_u if and only if it is a relevant solution of the equation $v[f_1 := t_1, \dots, f_n := t_n] = u$. We prove, more generally, that for any term w of N , v is recognized in q_w if and only if $v[f_1 := t_1, \dots, f_n := t_n] = w$.

The direct sense is easy. We prove by induction over the structure of v that if v is recognized in q_w then v is a relevant solution of the equation $v[f_1 := t_1, \dots, f_n := t_n] = w$. If $v = (f_i \ v_1 \ \dots \ v_p)$ then the term v_i is recognized in a state q_{w_i} where w_i is either a term of N or \square_T and $(t_i \ \vec{w}) = w$. In the first case, by induction hypothesis v_i is a relevant solution of the equation $v_i[f_1 := t_1, \dots, f_n := t_n] = t_i$ and in the second $v_i = \square_T$. Thus $(t_i \ v_1[f_1 := t_1, \dots, f_n := t_n] \ \dots \ v_p[f_1 := t_1, \dots, f_n := t_n]) = u$ i.e. $v[f_1 := t_1, \dots, f_n := t_n] = u$, and moreover v is relevant. If $v = (h \ v_1 \ \dots \ v_p)$ then the v_i are recognized in states q_{w_i} with w_i in N . By induction hypothesis v_i are relevant solutions of $v_i[f_1 := t_1, \dots, f_n := t_n] = t_i$. Hence $v[f_1 := t_1, \dots, f_n := t_n] = u$ and moreover v is relevant. The case where v is an abstraction is similar.

Conversely, assume that v is a relevant solution of the equation $v[f_1 := t_1, \dots, f_n := t_n] = w$. We prove, by induction over the structure of v , that v is recognized in q_w .

If $v = (f_i \ v_1 \ \dots \ v_p)$ then $(t_i \ v_1[f_1 := t_1, \dots, f_n := t_n] \ \dots \ v_p[f_1 := t_1, \dots, f_n := t_n]) = u$. Let $v'_i = v_i[f_1 := t_1, \dots, f_n := t_n]$. The v'_i are a relevant solutions of the second-order matching problem $(t_i \ v'_1 \ \dots \ v'_p) = u$.

Hence, by proposition 4.3.3, each v'_i is either an element of N or the constant \square_T . In both cases v_i is a relevant solution of the equation $v_i[f_1 := t_1, \dots, f_n := t_n] = v'_i$ and by induction hypothesis v_i is recognized in q_{w_i} . Thus v is recognized in q_w . If $v = (h \ v_1 \ \dots \ v_p)$ then $(h \ v_1[f_1 := t_1, \dots, f_n := t_n] \ \dots \ v_p[f_1 := t_1, \dots, f_n := t_n]) = w$. Let $w_i = v_i[f_1 := t_1, \dots, f_n := t_n]$. We have $(h \ \vec{w}) = w$ and v_i is a relevant solution of the equation $v_i[f_1 := t_1, \dots, f_n := t_n] = w_i$. By induction hypothesis v_i is recognized in q_{w_i} . Thus v is recognized in q_w . The case where w is an abstraction is similar. ■

4.3.6. PROPOSITION. *Fourth-order dual interpolation is decidable.*

PROOF. Consider a system of equations and disequations and the automata associated to all these equations. Let \mathcal{L} be the language containing the union of the languages of these automata and an extra constant of type o . Obviously the system has a solution if and only if it has a solution in the language \mathcal{L} . Each automaton recognizing the relevant solutions can be transformed into one recognizing all the solutions in \mathcal{L} (adding a finite number of rules, so that the state \square_T recognizes all terms of type T in the language \mathcal{L}). Then using the fact that languages recognized by a tree automaton are closed by intersection and complement, we build a automaton recognizing all the solutions of the system in the language \mathcal{L} . The system has a solution if and only if the language recognized by this automaton is non empty.

Decidability follows from the decidability of the emptiness of a language recognized by a tree automaton. ■

Decidability of rank 3 matching

A particular case

We shall start by proving the decidability of a subcase of fourth-order matching where problems are formulated in a language without any constant and the solutions also must not contain any constant.

Consider a problem $t = u$. The term u contains no constant. Hence, by Theorem 3.4.7 there is a closed term r of type $T \rightarrow o$, whose constants have order at most two (i.e. level at most one), such that for each term t of type T

$$t =_{\beta\eta} u \iff \forall i. (r\ t) =_{\beta\eta} (r\ u).$$

The normal form of $(r\ u) \in \Lambda^\emptyset(o)$ is a closed term whose constants have order at most two, thus it contains no bound variables. Let U be the set of all subterms of type o of the normal forms of $(r\ u)$. All these terms are closed. Like in the relation defined by equality in the model of the finite completeness theorem, we define a congruence on closed terms of type o that identifies all terms that are not in U . This congruence has $\text{card}(U) + 1$ equivalence classes.

4.3.7. DEFINITION. $t =_{\beta\eta u} t' \iff \forall s \in U [t =_{\beta\eta} s \iff t' =_{\beta\eta} s]$.

Notice that if $t, t' \in \Lambda^\emptyset(o)$ one has the following

$$\begin{aligned} t =_{\beta\eta u} t' &\iff t =_{\beta\eta} t' \text{ or } \forall s \in U (t \neq_{\beta\eta} s \ \& \ t' \neq_{\beta\eta} s) \\ &\iff [t =_{\beta\eta} t' \\ &\quad \text{or neither the normal form of } t \text{ nor that of } t' \text{ is in } U] \end{aligned}$$

Now we extend this to a logical relation on closed terms of arbitrary types. The following construction could be considered as an application of the Gandy Hull defined in Example 3.3.37. However, we choose to do it explicitly so as to prepare for Definition 4.3.16.

4.3.8. DEFINITION. Let \parallel_v be the logical relation lifted from $=_{\beta\eta u}$ on closed terms.

4.3.9. LEMMA. (i) \parallel_v is head-expansive.

(ii) For each constant F of type of rank ≤ 1 one has $F \parallel_v F$.

(iii) For any $X \in \Lambda(A)$ one has $X \parallel_v X$.

(iv) \parallel_v is an equivalence relation.

(v) $P \parallel_v Q \iff \forall R_1, \dots, R_k. P \vec{R} \parallel_v Q \vec{R}$.

We want to prove, using the decidability of the dual interpolation problem, that the equivalence classes of this relation can be enumerated up to order four, i.e. that we can compute a set \mathcal{E}_T of closed terms containing a term in each class.

More generally, we shall prove that if dual interpolation of rank n is decidable, then the sets $\mathcal{T}_T / \parallel_u$ can be enumerated up to rank n . [We first prove the following Proposition.](#)

4.3.10. PROPOSITION (Substitution lemma). *Let t be a normal term of type o , whose free variables are x_1, \dots, x_n . Let $v_1, \dots, v_n, v'_1, \dots, v'_n$ be closed terms such that $v_1 \parallel_u v'_1, \dots, v_n \parallel_u v'_n$. Let $\sigma = v_1/x_1, \dots, v_n/x_n$ and $\sigma' = v'_1/x_1, \dots, v'_n/x_n$. Then*

$$\sigma t =_{\beta\eta u} \sigma' t$$

PROOF. By induction on the pair formed with the length of the longest reduction in σt and the size of t . The term t is normal and has type o , thus it has the form $(f w_1 \dots w_k)$.

If f is a constant, then let us write $w_i = \bar{\lambda}s_i$ with s_i of type o . We have $\sigma t = (f \bar{\lambda} \sigma s_1 \dots \bar{\lambda} \sigma s_k)$ and $\sigma' t = (f \bar{\lambda} \sigma' s_1 \dots \bar{\lambda} \sigma' s_k)$. By induction hypothesis (as the s_i 's are subterms of a) we have $\sigma s_1 =_{\beta\eta u} \sigma' s_1, \dots, \sigma s_k =_{\beta\eta u} \sigma' s_k$ thus either for all i , $\sigma s_i =_{\beta\eta} \sigma' s_i$ and in this case $\sigma t =_{\beta\eta} \sigma' t$ or for some i , neither the normal forms of σs_i nor that of $\sigma' s_i$ is an element of U . In this case neither the normal form of σt nor that of $\sigma' t$ is in U and $\sigma t =_{\beta\eta u} \sigma' t$.

If f is a variable x_i and $k = 0$ then $t = x_i$, $\sigma t = v_i$ and $\sigma' t = v'_i$ and v_i and v'_i have type o . Thus $\sigma t =_{\beta\eta u} \sigma' t$.

Otherwise, f is a variable x_i and $k \neq 0$. The term v_i has the form $\lambda z_1 \dots \lambda z_k s$ and the term v'_i has the form $\lambda z_1 \dots \lambda z_k s'$. We have

$$\sigma t = (v_i \sigma w_1 \dots \sigma w_k) =_{\beta\eta} s[\sigma w_1/z_1, \dots, \sigma w_k/z_k]$$

and $\sigma' t = (v'_i \sigma' w_1 \dots \sigma' w_k)$. As $v_i \parallel_u v'_i$, we get

$$\sigma' t =_{\beta\eta u} (v_i \sigma' w_1 \dots \sigma' w_k) =_{\beta\eta u} s[\sigma' w_1/z_1, \dots, \sigma' w_k/z_k]$$

It is routine to check that for all i , $(\sigma w_i) \parallel_u (\sigma' w_i)$. Indeed, if the term w_i has the form $\lambda_1 \dots \lambda_{y_p} a$, then for all closed terms $b_1 \dots b_p$, we have

$$\begin{aligned} \sigma w_i b_1 \dots b_p &= ((b_1/y_1, \dots, b_p/y_p) \circ \sigma) a \\ \sigma' w_i b_1 \dots b_p &= ((b_1/y_1, \dots, b_p/y_p) \circ \sigma') a. \end{aligned}$$

Applying the induction hypothesis to a that is a subterm of t , we get

$$(\sigma w_i) b_1 \dots b_p =_{\beta\eta u} (\sigma' w_i) b_1 \dots b_p$$

and thus $(\sigma w_i) \parallel_u (\sigma' w_i)$.

As $(\sigma w_i) \parallel_u (\sigma' w_i)$ we can apply the induction hypothesis again (because $s[\sigma w_1/z_1, \dots, \sigma w_k/z_k]$ is a reduct of σt) and get

$$s[\sigma w_1/z_1, \dots, \sigma w_k/z_k] =_{\beta\eta u} s[\sigma' w_1/z_1, \dots, \sigma' w_k/z_k]$$

Thus $\sigma t =_{\beta\eta u} \sigma' t$. ■

The next proposition is a direct corollary.

4.3.11. PROPOSITION (Application lemma). *If $v_1 \parallel_u v'_1, \dots, v_n \parallel_u v'_n$ then for all term t of type $T_1 \rightarrow \dots \rightarrow T_n \rightarrow o$,*

$$(t v_1 \dots v_n) =_{\beta\eta u} (t v'_1 \dots v'_n)$$

PROOF. Applying proposition 4.3.10 to the term $(t \ x_1 \ \dots \ x_n)$. ■

We then prove the following lemma that justifies the use of the relations $=_{\beta\eta u}$ and \parallel_u .

4.3.12. PROPOSITION (Discrimination lemma). *For every term t , if $t \parallel_u u$ then $t =_{\beta\eta} u$.*

PROOF. As $t \parallel_u u$, by proposition 4.3.11, we have for all i , $(r_i \ t) =_{\beta\eta u} (r_i \ u)$. Hence, as the normal form of $(r_i \ u)$ is in U , $(r_i \ t) =_{\beta\eta} (r_i \ u)$. Thus $t =_{\beta\eta} u$. ■

Let us discuss now how we can decide and enumerate the relation \parallel_u . If t and t' of type $T_1 \rightarrow \dots \rightarrow T_n \rightarrow o$, then, by definition, $t \parallel_u t'$ if and only if

$$\forall w_1 \in \mathcal{T}_{T_1} \dots \forall w_n \in \mathcal{T}_{T_n} (t \ \vec{w} =_{\beta\eta E} t' \ \vec{w})$$

The fact that $t \ \vec{w} =_{\beta\eta E} t' \ \vec{w}$ can be reformulated

$$\forall s \in U (t \ \vec{w} =_{\beta\eta} s \text{ if and only if } t' \ \vec{w} =_{\beta\eta} s)$$

Thus $t \parallel_u t'$ if and only if

$$\forall w_1 \in \mathcal{T}_{T_1} \dots \forall w_n \in \mathcal{T}_{T_n} \ \forall s \in U (t \ \vec{w} =_{\beta\eta} s \text{ if and only if } t' \ \vec{w} =_{\beta\eta} s)$$

Thus to decide if $t \parallel_u t'$, we should list all the sequences s, w_1, \dots, w_n where s is an element of U and w_1, \dots, w_n are closed terms of type T_1, \dots, T_n , and check that the set of sequences such that $t \ \vec{w} =_{\beta\eta} s$ is the same as the set of sequences such that $t' \ \vec{w} =_{\beta\eta} s$.

Of course, the problem is that there is an infinite number of such sequences. But by proposition 4.3.11 the fact that $t \ \vec{w} =_{\beta\eta u} t' \ \vec{w}$ is not affected if we replace the terms w_i by \parallel_u -equivalent terms. Hence, if we can enumerate the sets $\mathcal{T}_{T_1} / \parallel_u, \dots, \mathcal{T}_{T_n} / \parallel_u$ by sets $\mathcal{E}_{T_1}, \dots, \mathcal{E}_{T_n}$, then we can decide the relation \parallel_u for terms of type $T_1 \rightarrow \dots \rightarrow T_n \rightarrow o$ by enumerating the sequences in $U \times \mathcal{E}_{T_1} \times \dots \times \mathcal{E}_{T_n}$, and checking that the set of sequences such that $t \ \vec{w} =_{\beta\eta} s$ is the same as the set of sequences such that $t' \ \vec{w} =_{\beta\eta} s$.

As class of a term t for the relation \parallel_u is completely determined, by the set of sequences s, w_1, \dots, w_n such that $t \ \vec{w} =_{\beta\eta} s$ and there are a finite number of subsets of the set $E = U \times \mathcal{E}_{T_1} \times \dots \times \mathcal{E}_{T_n}$, we get this way that the set $\mathcal{T}_T / \parallel_u$ is finite.

To obtain an enumeration \mathcal{E}_T of the set $\mathcal{T}_T / \parallel_u$ we need to be able to select the subsets A of $U \times \mathcal{E}_{T_1} \times \dots \times \mathcal{E}_{T_n}$, such that there is a term t such that $t \ \vec{w} =_{\beta\eta} s$ if and only if the sequence s, \vec{w} is in A . This condition is exactly the decidability of the dual interpolation problem. This leads to the following proposition.

4.3.13. PROPOSITION (Enumeration lemma). *If dual interpolation of rank n is decidable, then the sets $\mathcal{T}_T / \parallel_u$ can be enumerated up to rank n .*

PROOF. By induction on the order of $T = T_1 \rightarrow \dots \rightarrow T_n \rightarrow o$. By the induction hypothesis, the sets $\mathcal{T}_{T_1}/\parallel_u, \dots, \mathcal{T}_{T_n}/\parallel_u$ can be enumerated by sets $\mathcal{E}_{T_1}, \dots, \mathcal{E}_{T_n}$.

Let x be a variable of type T . For each subset A of $E = U \times \mathcal{E}_{T_1} \times \dots \times \mathcal{E}_{T_n}$ we define the dual interpolation problem containing the equation $x\vec{w} = s$ for $s, w_1, \dots, w_p \in A$ and the negated equation $x\vec{w} \neq s$ for $s, w_1, \dots, w_p \notin A$. Using the decidability of dual interpolation of rank n , we select those of such problems that have a solution and we chose a closed solution for each problem. We get this way a set \mathcal{E}_T .

We prove that this set is an enumeration of $\mathcal{T}_T/\parallel_u$, i.e. that for every term t of type T there is a term t' in \mathcal{E}_T such that $t' \parallel_u t$. Let A be the set of sequences s, w_1, \dots, w_p such that $(t \vec{w}) =_{\beta_\eta} s$. The dual interpolation problem corresponding to A has a solution (for instance t). Thus one of its solutions t' is in \mathcal{E}_T . We have

$$\forall w_1 \in \mathcal{E}_{T_1} \dots \forall w_n \in \mathcal{E}_{T_n} \forall s \in U ((t \vec{w}) =_{\beta_\eta} s \iff (t' \vec{w}) =_{\beta_\eta} s)$$

Thus

$$\forall w_1 \in \mathcal{E}_{T_1} \dots \forall w_n \in \mathcal{E}_{T_n} (t \vec{w}) =_{\beta_\eta u} (t' \vec{w})$$

and thus, by proposition 4.3.11

$$\forall w_1 \in \mathcal{T}_{T_1} \dots \forall w_n \in \mathcal{T}_{T_n} (t \vec{w}) =_{\beta_\eta u} (t' \vec{w})$$

Thus $t \parallel_u t'$. ■

Then, we prove that if the sets $\mathcal{T}_T/\parallel_u$ can be enumerated up to rank n , then matching of rank n is decidable. The idea is that we can restrict the search of solutions to the sets \mathcal{E}_T .

4.3.14. PROPOSITION (Matching lemma). *If the sets $\mathcal{T}_T/\parallel_u$ can be enumerated up to order N , then matching problems of rank n whose right hand side is u can be decided.*

PROOF. Let $\vec{X} = X_1, \dots, X_n$. We prove that if a matching problem $t\vec{X} = u$ has a solution \vec{v} , then it has also a solution \vec{v}' , such that $v'_i \in \mathcal{E}_T$, for each i .

As \vec{v} is a solution of the problem $t = u$, we have $t\vec{v} =_{\beta_\eta} u$.

For all i , let v'_i be a representative in \mathcal{E}_{T_i} of the class of v_i . We have

$$v'_1 \parallel_u v_1, \dots, v'_n \parallel_u v_n.$$

Thus by proposition 4.3.10

$$t\vec{v} =_{\beta_\eta u} \vec{v}',$$

thus

$$t\vec{v}' =_{\beta_\eta u} u$$

and by proposition 4.3.12

$$t\vec{v}' =_{\beta_\eta} u$$

Thus to check if a problem has a solution it is sufficient to check if it has a solution \vec{v}' , such that each v'_i is a member of \mathcal{E}_T , and such substitutions can be enumerated. ■

4.3.15. THEOREM. *Fourth-order matching problems whose right hand side contain no constants can be decided.*

PROOF. Dual interpolation of order 4 is decidable, hence, by proposition 4.3.13, if u is a closed term containing no constants, then the sets $\mathcal{T} / \parallel_u$ can be enumerated up to order 4, hence, by proposition 4.3.14, we can decide if a problem of the form $t = u$ has a solution. ■

The general case

We consider now terms formed in a language containing an infinite number of constants of each type and we want to generalize the result. The difficulty is that we cannot apply Statman's result anymore to eliminate bound variables. Hence we shall define directly the set U as the set of subterms of u of type o . The novelty here is that the bound variables of U may now appear free in the terms of U . It is important here to choose the names x_1, \dots, x_n of these variables, once for all.

We define the congruence $t =_{\beta\eta u} t'$ on terms of type o that identifies all terms that are not in U .

4.3.16. DEFINITION. (i) Let $t, t' \in \Lambda(o)$ (not necessarily closed). Define

$$t =_{\beta\eta u} t' \iff \forall s \in U. [t =_{\beta\eta} s \iff t' =_{\beta\eta} s].$$

(ii) Define the logical relation \parallel_u by lifting $=_{\beta\eta U}$ to all open terms at higher types.

4.3.17. LEMMA. (i) \parallel_v is head-expansive.

(ii) For any variable of arbitrary type A one has $x \parallel_v x$.

(iii) For each constant $F \in \Lambda(A)$ one has $F \parallel_v F$.

(iv) For any $X \in \Lambda(A)$ one has $X \parallel_v X$.

(v) \parallel_v is an equivalence relation at all types.

(vi) $P \parallel_v Q \iff \forall R_1, \dots, R_k. P\tilde{R} \parallel_v Q\tilde{R}$.

PROOF. (i) By definition the relation is closed under arbitrary $\beta\eta$ expansion.

(ii) By induction on the generation of the type A .

(iii) Similarly.

(iv) Easy.

(v) Easy.

(vi) Easy. ■

Then we can turn to the enumerability lemma (proposition 4.3.13). Due to the presence of the free variables, the proof of this lemma introduces several novelties. Given a subset A of $E = U \times \mathcal{E}_{T_1} \times \dots \times \mathcal{E}_{T_n}$ we cannot define the dual interpolation problem containing the equation $(x \vec{w}) = s$ for $s, w_1, \dots, w_p \in A$ and the negated equation $(x \vec{w}) \neq s$ for $s, w_1, \dots, w_p \notin A$, because the right hand side of these equations may contain free variables. Thus, we shall replace these variables by fresh constants c_1, \dots, c_n . Let θ

be the substitution $c_1/x_1, \dots, c_n/x_n$. To each set of sequences, we associate the dual interpolation problem containing the equation $(x \vec{w}) = \theta s$ or its negation.

This introduces two difficulties: first the term θs is not a subterm of u , thus, besides the relation \parallel_u , we shall need to consider also the relation $\parallel_{\theta s}$, and one of its enumerations, for each term s in U . Then, the solutions of such interpolation problems could contain the constants c_1, \dots, c_n , and we may have difficulties proving that they represent their \parallel_u -equivalence class. To solve this problem we need to duplicate the constants c_1, \dots, c_n with constants d_1, \dots, d_n . This idea goes back to [Goldfarb].

Let us consider a fixed set of constants $c_1, \dots, c_n, d_1, \dots, d_n$ that do not occur in u , and if t is a term containing constants c_1, \dots, c_n , but not the constants d_1, \dots, d_n , we write \tilde{t} for the term t where each constant c_i is replaced by the constant d_i .

Let $T = T_1 \rightarrow \dots \rightarrow T_n \rightarrow o$ be a type. We assume that for any closed term s of type o , the sets $\mathcal{T}_{T_i} / \parallel_s$ can be enumerated up to rank n by sets $\mathcal{E}_{T_i}^s$.

4.3.18. DEFINITION. We define the set of sequences E containing for each term s in U and sequence w_1, \dots, w_n in $\mathcal{E}_{T_1}^{\theta s} \times \dots \times \mathcal{E}_{T_n}^{\theta s}$, the sequence $\theta s, w_1, \dots, w_n$. Notice that the terms in these sequences may contain the constants c_1, \dots, c_n but not the constants d_1, \dots, d_n .

To each subset A of E we associate a dual interpolation problem containing the equations $x \vec{w} = s$ and $x \tilde{w}_1 \dots \tilde{w}_n = \tilde{s}$ for $s, w_1, \dots, w_n \in A$ and the disequations $x \vec{w} \neq s$ and $x \tilde{w}_1 \dots \tilde{w}_n \neq \tilde{s}$ for $s, w_1, \dots, w_n \notin A$.

The first lemma justifies the use of constants duplication.

4.3.19. PROPOSITION. *If an interpolation problem of definition 4.3.18 has a solution t , then it also has a solution t' that does not contain the constants $c_1, \dots, c_n, d_1, \dots, d_n$.*

PROOF. Assume that the term t contains a constant, say c_1 . Then by replacing this constant c_1 by a fresh constant e , we obtain a term t' . As the constant e is fresh, all the disequations that t verify are still verified by t' . If t verifies the equations $x \vec{w} = s$ and $x \tilde{w}_1 \dots \tilde{w}_n = \tilde{s}$ then the constant e does not occur in the normal form of $t' \vec{w}$. Otherwise the constant c_1 would occur in the normal form of $t \tilde{w}_1 \dots \tilde{w}_n$, i.e. in the normal form of \tilde{s} which is not the case. Thus t' also verifies the equations $x \vec{w} = s$ and $x \tilde{w}_1 \dots \tilde{w}_n = \tilde{s}$.

We can replace this way all the constants $c_1, \dots, c_n, d_1, \dots, d_n$ by fresh constants, obtaining a solution where these constants do not occur. ■

Then, we prove that the interpolation problems of definition 4.3.18 characterize the equivalence classes of the relation \parallel_u .

4.3.20. PROPOSITION. *Every term t of type T not containing the constants $c_1, \dots, c_n, d_1, \dots, d_n$ is the solution of a unique problem of definition 4.3.18.*

PROOF. Consider the subset A of E formed with sequences s, w_1, \dots, w_n such that $t \vec{w} = s$. The term t is the solution of the interpolation problem associated to A and A is the only subset of E such that t is a solution to the interpolation problem associated to. ■

4.3.21. PROPOSITION. *Let t and t' be two terms of type T not containing the constants $c_1, \dots, c_n, d_1, \dots, d_n$. Then t and t' are solutions of the same problem if and only if $t \parallel_u t'$.*

PROOF. By definition if $t \parallel_u t'$ then for all w_1, \dots, w_n and for all s in U $t \vec{w} =_{\beta_\eta} s \iff t' \vec{w} =_{\beta_\eta} s$. Thus for any s, \vec{w} in E , $\theta^{-1}s$ is in U and $t \theta^{-1}w_1 \dots \theta^{-1}w_n =_{\beta_\eta} \theta^{-1}s \iff t' \theta^{-1}w_1 \dots \theta^{-1}w_n =_{\beta_\eta} \theta^{-1}s$. Then as the constants $c_1, \dots, c_n, d_1, \dots, d_n$ do not appear in t and t' $t \vec{w} =_{\beta_\eta} s \iff t' \vec{w} =_{\beta_\eta} s$ and $t \tilde{w}_1 \dots \tilde{w}_n =_{\beta_\eta} s \iff t' \tilde{w}_1 \dots \tilde{w}_n =_{\beta_\eta} s$. Thus t and t' are the solutions of the same problems.

Conversely, assume that $t \not\parallel_u t'$. Then there exists terms w_1, \dots, w_n and a term s in U such that $t \vec{w} =_{\beta_\eta} s$ and $t' \vec{w} \neq_{\beta_\eta} s$. Hence $t \theta w_1 \dots \theta w_n =_{\beta_\eta} \theta s$ and $t' \theta w_1 \dots \theta w_n \neq_{\beta_\eta} \theta s$. As the sets $\mathcal{E}_{T_i}^{\theta s}$ are enumeration of the sets $\mathcal{T}_{T_i} / \parallel_{\theta s}$ there exists terms \vec{r} such that the $r_i \parallel_{\theta s} \theta w_i$ and $\theta s, \vec{r} \in E$. Using proposition 4.3.11 we have $t \vec{r} =_{\beta_\eta \theta s} t \theta w_1 \dots \theta w_n =_{\beta_\eta} \theta s$ hence $t \vec{r} =_{\beta_\eta \theta s} \theta s$ i.e. $t \vec{r} =_{\beta_\eta} \theta s$. Similarly, we have $t' \vec{r} =_{\beta_\eta \theta s} t' \theta w_1 \dots \theta w_n \neq_{\beta_\eta} \theta s$ hence $t' \vec{r} \neq_{\beta_\eta \theta s} \theta s$ i.e. $t' \vec{r} \neq_{\beta_\eta} \theta s$. Hence t and t' are not the solutions of the same problems. ■

Finally, we can prove the enumeration lemma.

4.3.22. PROPOSITION (Enumeration lemma). *If dual interpolation of rank n is decidable, then, for any closed term u of type o , the sets $\mathcal{T}_T / \parallel_u$ can be enumerated up to rank n .*

PROOF. By induction on the order of T . Let $T = T_1 \rightarrow \dots \rightarrow T_n \rightarrow o$. By the induction hypothesis, for any closed term s of type o , the sets $\mathcal{T}_{T_i} / \parallel_s$ can be enumerated by sets $\mathcal{E}_{T_i}^s$.

We consider all the interpolation problems of definition 4.3.18. Using the decidability of dual interpolation of rank n , we select those of such problems that have a solution. By proposition 4.3.19, we can construct for each such problem a solution not containing the constants $c_1, \dots, c_n, d_1, \dots, d_n$ and by proposition 4.3.20 and 4.3.21, these terms form an enumeration of $\mathcal{T}_T / \parallel_u$. ■

To conclude, we prove the matching lemma (proposition 4.3.14) exactly as in the particular case and then the theorem.

4.3.23. THEOREM. (Padovani) *Fourth-order matching problems can be decided.*

PROOF. Dual interpolation of order 4 is decidable, hence, by proposition 4.3.13, if u is a closed term, then the sets $\mathcal{T} / \parallel_u$ can be enumerated up to order 4, hence, by proposition 4.3.14, we can decide if a problem of the form $t = u$ has a solution. ■

4.4. Decidability of the maximal theory

We prove now that the maximal theory is decidable. The original proof of this result is due to Vincent Padovani [1996]. This proof has later been simplified independently by Schmidt-Schauß and Loader [1997], based on Schmidt-Schauß [1999].

Remember that the maximal theory is

$$\mathcal{T}_{\max}\{M = N \mid M, N \in \Lambda_o^\emptyset(A), A \in \mathbb{T}_o \text{ \& } \mathcal{M}_{\min}^{\vec{c}} \models M = N\},$$

where

$$\mathcal{M}_{\min}^{\vec{c}} = \Lambda_o^\emptyset[\vec{c}] / \approx_{\vec{c}}^{\text{ext}}$$

consists of all terms having the $\vec{c} = c_1, \dots, c_n$, with $n > 1$, of type o as distinct constants and $M \approx_{\vec{c}}^{\text{ext}} N$ on type $A = A_1 \rightarrow \dots \rightarrow A_n \rightarrow o$ is defined by

$$M \approx_{\vec{c}}^{\text{ext}} N \iff \forall t_1 \in \Lambda_o^\emptyset[\vec{c}](A_1) \dots t_n \in \Lambda_o^\emptyset[\vec{c}](A_n). M\vec{t} =_{\beta\eta} N\vec{t}.$$

Theorem 3.5.29 states that $\approx_{\vec{c}}^{\text{ext}}$ is a congruence which we will denote by \approx . Also that theorem implies that \mathcal{T}_{\max} is independent of n .

4.4.1. DEFINITION. For a type $T \in \mathbb{T}_{\mathbb{A}}$ we define its *degree* $\|T\|$ as follows.

$$\begin{aligned} \|o\| &= 2, \\ \|T \rightarrow U\| &= \|T\|! \|U\|, \quad \text{i.e. } \|T\| \text{ factorial times } \|U\|. \end{aligned}$$

4.4.2. PROPOSITION. (i) $\|T_1 \rightarrow \dots \rightarrow T_n \rightarrow o\| = 2\|T_1\|! \dots \|T_n\|!$.

(ii) $\|T_i\| < \|T_1 \rightarrow \dots \rightarrow T_n \rightarrow o\|$.

(iii) $n < \|T_1 \rightarrow \dots \rightarrow T_n \rightarrow o\|$.

(iv) If $p < \|T_i\|$, $\|U_1\| < \|T_i\|$, \dots , $\|U_p\| < \|T_i\|$ then

$$\begin{aligned} &\|T_1 \rightarrow \dots \rightarrow T_{i-1} \rightarrow U_1 \rightarrow \dots \rightarrow U_p \rightarrow T_{i+1} \rightarrow \dots \rightarrow T_n \rightarrow o\| < \\ &< \|T_1 \rightarrow \dots \rightarrow T_n \rightarrow o\| \end{aligned}$$

4.4.3. DEFINITION. Let $M \in \Lambda_o^\emptyset[\vec{c}](T_1 \rightarrow \dots \rightarrow T_n \rightarrow o)$ be a lnf. Then either $M \equiv \lambda x_1 \dots x_n. y$ or $M \equiv \lambda x_1 \dots x_n. x_i U_1 \dots U_p$. In the first case, M is called *constant*, in the second it has *index* i .

The following proposition states that for every type T , the terms $t \in \Lambda_o^\emptyset[\vec{c}](T)$ with a given index can be enumerated by a term $E : \vec{V} \rightarrow T$, where the \vec{V} have degrees lower than T .

4.4.4. PROPOSITION. Let \approx be the equality in the minimal model (the maximal theory). Then for each type T and each natural number i , there exists a natural number $k < \|T\|$, types V_1, \dots, V_k such that $\|V_1\| < \|T\|$, \dots , $\|V_k\| < \|T\|$, a term E of type $V_1 \rightarrow \dots \rightarrow V_k \rightarrow T$ and terms B_1 of type $T \rightarrow V_1$, \dots , B_k of type $T \rightarrow V_k$ such that if t has index i then

$$t \approx E(B_1 t) \dots (B_k t)$$

PROOF. By induction on $\|T\|$. Let us write $T = T_1 \rightarrow \dots \rightarrow T_n \rightarrow o$ and $T_i = U_1 \rightarrow \dots \rightarrow U_m \rightarrow o$. By induction hypothesis, for each j in $\{1, \dots, m\}$ there are

types $W_{j,1}, \dots, W_{j,l_j}$, terms $E_j, B_{j,1}, \dots, B_{j,l_j}$ such that $l_j < \|T_i\|$, $\|W_{j,1}\| < \|T_i\|$, \dots , $\|W_{j,l_j}\| < \|T_i\|$ and if $u \in \Lambda_o^\emptyset[\vec{c}](T)_i$ has index j then

$$u \approx E_j(B_{j,1}u) \dots (B_{j,l_j}u).$$

We take $k = m$,

$$V_1 \equiv T_1 \rightarrow \dots \rightarrow T_{i-1} \rightarrow W_{1,1} \rightarrow \dots \rightarrow W_{1,l_1} \rightarrow T_{i+1} \rightarrow \dots \rightarrow T_n \rightarrow o,$$

\dots

$$V_k \equiv T_1 \rightarrow \dots \rightarrow T_{i-1} \rightarrow W_{k,1} \rightarrow \dots \rightarrow W_{k,l_k} \rightarrow T_{i+1} \rightarrow \dots \rightarrow T_n \rightarrow o,$$

$$\begin{aligned} E &\equiv \lambda f_1 \dots f_k x_1 \dots x_n. x_i (\lambda \vec{c}. f_1 x_1 \dots x_{i-1} (d_{1,1} x_i) \dots (d_{1,l_1} x_i) x_{i+1} \dots x_n) \\ &\quad \dots \\ &\quad (\lambda \vec{c}. f_k x_1 \dots x_{i-1} (d_{k,1} x_i) \dots (d_{k,l_k} x_i) x_{i+1} \dots x_n), \end{aligned}$$

$$B_1 \equiv \lambda g x_1 \dots x_{i-1} \vec{z}_1 x_{i+1} \dots x_n. g x_1 \dots x_{i-1} (E_1 \vec{z}_1) x_{i+1} \dots x_n,$$

\dots

$$B_k \equiv \lambda g x_1 \dots x_{i-1} \vec{z}_k x_{i+1} \dots x_n. g x_1 \dots x_{i-1} (E_k \vec{z}_k) x_{i+1} \dots x_n,$$

where $\vec{z}_i = z_1, \dots, z_{l_i}$ for $1 \leq i \leq k$. We have $k < \|T_i\| < \|T\|$, $\|V_i\| < \|T\|$ for $1 \leq i \leq k$ and for any $t \in \Lambda_o^\emptyset[\vec{c}](T)$

$$\begin{aligned} E(B_1 t) \dots (B_k t) &= \lambda x_1 \dots x_n. x_i \\ &\quad (\lambda \vec{c}. t x_1 \dots x_{i-1} (E_1 (B_{1,1} x_i) \dots (B_{1,l_1} x_i)) x_{i+1} \dots x_n) \\ &\quad \dots \\ &\quad (\lambda \vec{c}. t x_1 \dots x_{i-1} (E_k (B_{k,1} x_i) \dots (B_{k,l_k} x_i)) x_{i+1} \dots x_n) \end{aligned}$$

We want to prove that if t has index i then this term is equal to t . Consider terms $\vec{w} \in \Lambda_o^\emptyset[\vec{c}]$. We want to prove that for the term

$$\begin{aligned} N &= w_i (\lambda \vec{c}. t w_1 \dots w_{i-1} (E_1 (B_{1,1} w_i) \dots (B_{1,l_1} w_i)) w_{i+1} \dots w_n) \\ &\quad \dots \\ &\quad (\lambda \vec{c}. t w_1 \dots w_{i-1} (E_k (B_{k,1} w_i) \dots (B_{k,l_k} w_i)) w_{i+1} \dots w_n) \end{aligned}$$

one has $N \approx (t w_1 \dots w_n)$. If w_i is constant then this is obvious. Otherwise, it has an index j , say, and N reduces to

$$N' = t w_1 \dots w_{i-1} (E_j (B_{j,1} w_i) \dots (B_{j,l_j} w_i)) w_{i+1} \dots w_n.$$

By the induction hypothesis the term $(E_j (B_{j,1} w_i) \dots (B_{j,l_j} w_i)) \approx w_i$ and hence, by Theorem 3.5.29 one has $N = N' \approx (t w_1 \dots w_n)$. ■

4.4.5. THEOREM. Let \mathcal{M} be the minimal model built over $\vec{c}:o$, i.e.

$$\mathcal{M} = \mathcal{M}_{\min} = \Lambda_o^\emptyset[\vec{c}]/\approx.$$

For each type T , we can compute a finite set $\mathcal{R}_T \subseteq \Lambda_o^\emptyset[\vec{c}](T)$ that enumerates $\mathcal{M}(T)$, i.e. such that

$$\forall M \in \mathcal{M}(T) \exists N \in \mathcal{R}_T. M \approx N.$$

PROOF. By induction on $\|T\|$. If $T = o$, then we can take $\mathcal{R}_T = \{\vec{c}\}$. Otherwise write $T = T_1 \rightarrow \dots \rightarrow T_n \rightarrow o$. By Proposition 4.4.4 for each $i \in \{1, \dots, n\}$, there exists a natural number k_i , types $V_{i,1}, \dots, V_{i,k_i}$ smaller than T , a term E_i of type $V_{i,1} \rightarrow \dots \rightarrow V_{i,k_i} \rightarrow T$ such that for each term t of index i , there exists terms v_1, \dots, v_{k_i} such that

$$t \approx (E_i v_1 \dots v_{k_i}).$$

By the induction hypothesis, for each type $V_{i,j}$ we can compute a finite set $R_{V_{i,j}}$ that enumerates $\mathcal{M}(V_{i,j})$. We take for R_T all the terms of the form $(E_i e_1 \dots e_{k_i})$ with e_1 in $R_{V_{i,1}}, \dots, e_{k_i}$ in $R_{V_{i,k_i}}$. ■

4.4.6. COROLLARY (Padovani). *The maximal theory is decidable.*

PROOF. Check equivalence in any minimal model $\mathcal{M}_{\min}^{\vec{c}}$. At type $A = A_1 \rightarrow \dots \rightarrow A_n \rightarrow o$ we have

$$M \approx N \iff \forall P_1 \in \Lambda_o^\emptyset[\vec{c}](A_1) \dots P_n \in \Lambda_o^\emptyset[\vec{c}](A_n). M\vec{t} =_{\beta\eta} N\vec{t},$$

where we can now restrict the \vec{P} to the \mathcal{R}_{A_j} . ■

4.4.7. COROLLARY (Decidability of unification in \mathcal{T}_{\max}). *For terms*

$$M, N \in \Lambda_o^\emptyset[\vec{c}](A \rightarrow B),$$

of the same type, the following unification problem is decidable

$$\exists X \in \Lambda_o^\emptyset[\vec{c}](A). MX \approx NX.$$

PROOF. Working in $\mathcal{M}_{\min}^{\vec{c}}$, check the finitely many enumerating terms as candidates. ■

4.4.8. COROLLARY (Decidability of atomic higher-order matching). (i) *For*

$$M_1 \in \Lambda_o^\emptyset[\vec{c}](A_1 \rightarrow o), \dots, M_n \in \Lambda_o^\emptyset[\vec{c}](A_n \rightarrow o),$$

with $1 \leq i \leq n$, the following problem is decidable

$$\begin{aligned} \exists X_1 \in \Lambda_o^\emptyset[\vec{c}](A_1), \dots, X_n \in \Lambda_o^\emptyset[\vec{c}](A_n). [M_1 X_1 &=_{\beta\eta} c_1 \\ &\dots \\ M_n X_n &=_{\beta\eta} c_n]. \end{aligned}$$

(ii) *For $M, N \in \Lambda_o^\emptyset[\vec{c}](A \rightarrow o)$ the following problem is decidable.*

$$\exists X \in \Lambda_o^\emptyset[\vec{c}](A). MX =_{\beta\eta} NX.$$

PROOF. (i) Since $\beta\eta$ -convertibility at type o is equivalent to \approx , the previous Corollary applies.

(ii) Similarly to (i) or by reducing this problem to the problem in (i). ■

The non-redundancy of the enumeration

We now prove that the enumeration of terms in Proposition is not redundant. We follow the given construction, but actually the proof does not depend on it, see Exercise 4.5.2. We first prove a converse to Proposition 4.4.4.

4.4.9. PROPOSITION. *Let E, B_1, \dots, B_k be the terms constructed in Proposition 4.4.4. Then for any sequence of terms M_1, \dots, M_k , we have*

$$(B_j(EM_1 \dots M_k)) \approx M_j$$

PROOF. By induction on $\|T\|$ where T is the type of $(EM_1 \dots M_k)$. The term

$$N \equiv B_j(EM_1 \dots M_k)$$

reduces to

$$\begin{aligned} & \lambda x_1 \dots x_{i-1} \vec{z}_j x_{i+1} \dots x_n. E_j \vec{z}_j \\ & (\lambda \vec{c}. M_1 x_1 \dots x_{i-1} (B_{1,1}(E_j \vec{z}_j)) \dots (B_{1,l_1}(E_j \vec{z}_j)) x_{i+1} \dots x_n) \\ & \dots \\ & (\lambda \vec{c}. M_k x_1 \dots x_{i-1} (B_{k,1}(E_j \vec{z}_j)) \dots (B_{k,l_k}(E_j \vec{z}_j)) x_{i+1} \dots x_n) \end{aligned}$$

Then, since E_j is a term of index $l_j + j$, the term N continues to reduce to

$$\lambda x_1 \dots x_{i-1} \vec{z}_j x_{i+1} \dots x_n. M_j x_1 \dots x_{i-1} (B_{j,1}(E_j \vec{z}_j)) \dots (B_{j,l_j}(E_j \vec{z}_j)) x_{i+1} \dots x_n.$$

We want to prove that this term is equal to M_j . Consider terms

$$N_1, \dots, N_{i-1}, \vec{L}_j, N_{i+1}, \dots, N_n \in \Lambda_o^\emptyset[\vec{c}].$$

It suffices to show that

$$\begin{aligned} & M_j N_1 \dots N_{i-1} (B_{j,1}(E_j \vec{L}_j)) \dots (B_{j,l_j}(E_j \vec{L}_j)) N_{i+1} \dots N_n \approx \\ & M_j N_1 \dots N_{i-1} \vec{L}_j N_{i+1} \dots N_n. \end{aligned}$$

By the induction hypothesis we have

$$(B_{j,1}(E_j \vec{L}_j)) \approx L_1$$

...

$$(B_{j,l_j}(E_j \vec{L}_j)) \approx L_{l_j}$$

Hence by Theorem 3.5.29 we are done. ■

4.4.10. PROPOSITION. *The enumeration in Theorem 4.4.5 is non-redundant, i.e.*

$$\forall A \in \mathbb{T}_o \forall M, N \in \mathcal{R}_A. M \approx_c N \Rightarrow M \equiv N.$$

PROOF. Consider two terms M and N equal in the enumeration of a type A . We prove, by induction, that these two terms are equal. Since M and N are equal, they must have the same head variables. If this variable is free then they are equal. Otherwise, the terms have the form $M = (E_i M'_1 \dots M'_k)$ and $N = (E_i N'_1 \dots N'_k)$. For all j , we have

$$M'_j \approx (B_j M) \approx (B_j N) \approx N'_j$$

Hence, by induction hypothesis $M'_j = N'_j$ and therefore $M = N$. ■

4.5. Exercises

4.5.1. Let $T_n = 1^n \rightarrow o$ and let c_n be the cardinality of the set \mathcal{M}_{T_n} .

(i) Prove that

$$c_{n+1} = 2 + (n+1)c_n$$

(ii) Prove that

$$c_n = 2n! \sum_{i=0}^n \frac{1}{i!}.$$

The $d_n = n! \sum_{i=0}^n \frac{1}{i!}$ “the number of arrangements of n elements” forms a well-known sequence in combinatorics. See, for instance, Flajolet and Sedgewick [1993].

(iii) Can the cardinality of \mathcal{M}_T be bounded by a function of the form $k^{|T|}$ where $|T|$ is the size of T and k a constant?

4.5.2. Let $\mathcal{C} = \{c^o, d^o\}$. Let \mathcal{E} be a computable function that assigns to each type $A \in \mathbb{T}_o$ a finite set of terms \mathcal{X}_A such that for all

$$\forall M \in \Lambda[\mathcal{C}](A) \exists N \in \mathcal{X}_A. M \approx_{\mathcal{C}} N.$$

Show that not knowing the theory of section 4.4 one can effectively make \mathcal{E} non-redundant, i.e. such that

$$\forall A \in \mathbb{T}_o \forall M, N \in \mathcal{E}_A. M \approx_{\mathcal{C}} N \Rightarrow M \equiv N.$$

4.5.3. (Herbrand’s Problem) Consider sets S of universally quantified equations $\forall x_1 \dots x_n. [t_1 = t_2]$ between first order terms involving constants f, g, h, \dots various arities. Herbrand’s theorem concerns the problem of whether $S \models r = s$ where r, s are closed first order terms. For example the word problem for groups can be represented this way. Now let d be a new quaternary constant i.e. $d : 1_4$ and let a, b be new 0-ary constants i.e. $a, b : o$. We define the set S^+ of simply typed equations by

$$S^+ = \{ (\lambda \vec{x}. t_1 = \lambda \vec{x}. t_2) \mid (\forall \vec{x} [t_1 = t_2]) \in S \}.$$

Show that the following are equivalent

(i) $S \not\models r = s$.

(ii) $S^+ \cup \{\lambda x. dxxab = \lambda x. a, drsab = b\}$ is consistent.

Conclude that the consistency problem for finite sets of equations with constants is Π_1^0 -complete (in contrast to the decidability of finite sets of pure equations).

4.5.4. (Undecidability of second-order unification) Consider the unification problem

$$Fx_1 \dots x_n = Gx_1 \dots x_n,$$

where each x_i has type of rank < 2 . By the theory of reducibility we can assume that $Fx_1 \dots x_n$ has type $(o \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o)$ and so by introducing new constants of types o and $o \rightarrow (o \rightarrow o)$ we can assume $Fx_1 \dots x_n$ has type o . Thus we arrive

at the problem (with constants) in which we consider the problem of unifying 1st order terms built up from 1st and 2nd order constants and variables, The aim of this exercise is to show that it is recursively unsolvable by encoding Hilbert's 10-th problem, Goldfarb [1981]. For this we shall need several constants. Begin with constants

$$\begin{array}{ll} a, b & o \\ s & o \rightarrow o \\ e & o \rightarrow (o \rightarrow (o \rightarrow o)) \end{array}$$

The n th numeral is $s^n a$.

- (i) Let $F o \rightarrow o$. F is said to be affine if $F = \lambda x. s^n x$. N is a numeral iff there exists an affine F such that $Fa = N$. Show that F is affine $\iff F(sa) = s(Fa)$.
- (ii) Next show that $L = N + M$ iff there exist affine F and G such that $N = Fa$, $M = Ga$ & $L = F(Ga)$.
- (iii) We can encode a computation of $n * m$ by

$$e(n * m)m(e(n * (m - 1))(m - 1)(\dots(e(n * 1)11)\dots)).$$

Finally show that $L = N * M \iff \exists c, d, u, v$ affine and $\exists f, w$

$$\begin{aligned} fab &= e(ua)(va)(wab) \\ f(ca)(sa)(e(ca)(sa)b) &= e(u(ca))(v(sa))(fab) \\ L &= ua \\ N &= ca \\ M &= va \\ &= da. \end{aligned}$$

4.5.5. (Sets of unifiers) The intention of this exercise is to prove the following.

THEOREM. *Let A be a simple type and let $\mathcal{S} \subseteq \Lambda^\emptyset(A)$ be recursively enumerable and closed under $=_{\beta\eta}$. Then there exist simple types B and C and $F, G : A \rightarrow (B \rightarrow C)$ such that*

$$M \in \mathcal{S} \iff \exists N B.FMN =_{\beta\eta} GMN.$$

Moreover we can always assume that

$$C = (o \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o) = 1_{2 \rightarrow o \rightarrow o}.$$

In short, every r.e. $\beta\eta$ -closed set of combinators is the projection of the set of solutions to a unification problem.

- (i) Let A be given. Then by the reducibility theorem for simple types, there exists $H : A \rightarrow (o \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o)$ such that

$$\forall M, N A.[M =_{\beta\eta} N \iff HM =_{\beta\eta} HN].$$

Thus it suffices to prove the theorem for $A = 1_2 \rightarrow o \rightarrow o$. For similar reasons in the end it will suffice to have $C = 1_2 \rightarrow o \rightarrow o$. We shall need some special types below

$$\begin{aligned} o^* &= o \rightarrow o; \\ 1^* &= o^* \rightarrow o^*; \\ A &= (1^* \rightarrow (1^* \rightarrow 1^*)) \rightarrow (1^* \rightarrow 1^*); \\ A^+ &= (1^* \rightarrow (1^* \rightarrow 1^*)) \rightarrow ((1^* \rightarrow 1^*) \rightarrow (1^* \rightarrow 1^*)); \\ A^\# &= (1^* \rightarrow (1^* \rightarrow 1^*)) \rightarrow ((1^* \rightarrow 1^*) \rightarrow ((1^* \rightarrow 1^*) \rightarrow (1^* \rightarrow 1^*))). \end{aligned}$$

and we declare the following variables

$$\begin{aligned} f &: 1^* \rightarrow (1^* \rightarrow 1^*) \\ g &: 1^* \rightarrow 1^* \\ h &: 1^* \rightarrow 1^* \\ a &: o^* \rightarrow o^* \end{aligned}$$

where the reader should recognize $o^* \rightarrow o^*$ as the type of Church numerals.

(ii) We define several notions of terms recursively as follows.

- cheap
 a is cheap if r and s are cheap then $f r s$ is cheap
- inexpensive of level n
 a is inexpensive of level o
 if r and s are inexpensive of level n and j, k are among h, g then $f(jr)(ks)$ is inexpensive of level $n + 1$.

Clearly, X is a cheap term of depth n iff there exists an inexpensive term Y of level n such that $X =_{\beta\eta} [g = I, h = Ka]Y$.

Cheap terms are useful because for each $N (o \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o)$ in $\beta\eta$ normal form there exists a unique cheap X such that

$$N =_{\beta\eta} (\lambda b \lambda x y. b(\lambda u v. \lambda w z. x(uwz)(vwz))(\lambda w z. y) I y)(\lambda f a. X).$$

We shall say that M is frugal if $M =_{\beta\eta} \lambda f a. X$ where X is cheap. We shall try to prove our theorem for r.e. $\beta\eta$ closed sets of frugal terms.

(iii) We define

$$M A^\# \text{ is frugal if } M =_{\beta\eta} \lambda f g h a. Y, \text{ where } Y \text{ is inexpensive.}$$

Similarly we say that $M : A^+$ is frugal if $M =_{\beta\eta} \lambda f g a. [h = g]Y$ where Y is inexpensive. This will not introduce any ambiguity because of the difference between A, A^+ , and $A^\#$. For $M A^\#$ define $M^+ A^+$ by $M^+ = \lambda f g a. M f g g a$. We have M is frugal iff M^+ is frugal. Claim:

$$\begin{aligned} M : A^+ \text{ is frugal} &\iff \\ \lambda f g a. f(g(M f g a))(g(M f g a)) &=_{\beta\eta} \lambda f g a. M f g (f(g a)(g a)). \end{aligned}$$

Proof. W.l.o.g. we may assume that M is in long $\beta\eta$ normal form. The proof of \Leftarrow is by induction on the length of M . We let $M = \lambda fga.\lambda uv.Z$.

Case 1. $Z = aUV$. Let $@$ be the substitution $[a = f(ga)(ga)]$. In this case we have

$$\lambda uv.f(ga)(ga)(@U)(@V) =_{\beta\eta} f(g(\lambda uv.aUV))(g(\lambda uv.aUV)).$$

This is only possible if $\lambda uv.Z$ eta reduces to a so M is frugal.

Case 2. $Z = fXYUV$. In this case we have

$$\begin{aligned} f(g(\lambda uv.fXYUV))(g(\lambda uv.fXYUV)) &=_{\beta\eta} \\ \lambda uv.f(@X)(@Y)(@U)(@V) \end{aligned}$$

and as in case 1 this is not possible unless U eta reduce to u and V eta reduces to v . Thus $f(g(fXY))(g(fXY)) =_{\beta\eta} f(@X)(@Y)$ Now by similar reasoning X eta reduces to gP and Y eta reduces to gQ . Thus $f(gP)(gQ) =_{\beta\eta} @P$ and $@Q$; in particular, $@P =_{\beta\eta} @Q$ so $P =_{\beta\eta} Q$ and $f(gP)(gP) =_{\beta\eta} @P f(gQ)(gQ) =_{\beta\eta} @Q$. Thus setting $T = \lambda fga.P$ and

$$L = \lambda fga.Q \lambda fga.f(g(Tfga))(g(Tfga)) =_{\beta\eta} \lambda fga.Tfg(f(ga)(ga)),$$

and $\lambda fga.f(g(Lfga))(g(Lfga)) =_{\beta\eta} \lambda fga.Lfg(f(ga)(ga))$. Now the induction hypothesis applies to the long $\beta\eta$ -normal forms of T and L so these terms are frugal. Thus so is M .

Case 3. $Z = gXUV$. This case is impossible.

Case 4. $Z = uV$ or $Z = v$. These cases are impossible.

This completes the proof of the claim.

- (iv) By Matijasevic's solution to Hilbert's 10th problem for every r.e. set S of natural numbers there exists F and G such that n belongs to S iff there exists N such that $FnN =_{\beta\eta} GnN$ (here N can be taken to be a 12-tuple of Church numerals). Now there exists a bijective polynomial pairing function $p(x,y)$ represented by a lambda term P . If M is frugal then we take as the Gödel number for M the number represented by the $\beta\eta$ -nf of MP_1 . The set of Gödel numbers of members of an r.e. $\beta\eta$ closed set of frugal members of A is itself r.e. and has such F and G . We can now put all of these facts together. Consider the system

$$\begin{aligned} Ha &= \lambda bxy.b(\lambda uvwz.x(uwz)(vwz))(\lambda wz.y)ly) \\ b &= \lambda fa.cfla \\ d &= \lambda fga.cfgga \\ \lambda fga.dfg(f(ga)(ga)) &= \lambda fga.f(g(dfga))(g(dfga)) \\ F(bP_1)e &= G(bP_1)e \end{aligned}$$

is unifiable with $a = M$ iff M belongs to the original r.e. $\beta\eta$ -closed set of closed terms.

- 4.5.6. Consider $\Gamma_{n,m} = \{c_1 o, \dots, c_m o, f_1 1, \dots, f_n o\}$. Show that the unification problem with constants from Γ with several unknowns of type 1 can be reduced to the case where $m = 1$. This is equivalent to the following problem of Markov. Given a finite alphabet $\Sigma = \{a_1, \dots, a_n\}$ consider equations between words over $\Sigma \cup \{X_1, \dots, X_p\}$. The aim is to find for the unknowns \vec{X} words $w_1, \dots, w_p \in \Sigma^*$ such that the equations become syntactic identities. In Makanin [1977] it is proved that this problem is decidable (uniformly in n, p).
- 4.5.7. (Decidability of unification of second-order terms) Consider the unification problem $F\vec{x} = G\vec{x}$ of type A with $\text{rk}(A) = 1$. Here we are interested in the case of pure unifiers of any types. Then $A = 1_m = o^m \rightarrow o$ for some natural number m . Consider for $i = 1, \dots, m$ the systems

$$S_i = \{F\vec{x} = \lambda \vec{y}.y_i, G\vec{x} = \lambda \vec{y}.y_i\}.$$

- (i) Observe that the original unification problem is solvable iff one of the systems S_i is solvable.
- (ii) Show that systems whose equations have the form

$$F\vec{x} = \lambda \vec{y}.y_i$$

where $y_i : 0$ have the same solutions as single equations

$$H\vec{x} = \lambda xy.x$$

where $x, y : 0$

- (iii) Show that provided there are closed terms of the types of the x_i the solutions to a matching equation

$$H\vec{x} = \lambda xy.x$$

are exactly the same as the lambda definable solutions to this equation in the minimal model.

- (iv) Apply the method of Exercise 2.5.7 to the minimal model. Conclude that if there is a closed term of type A then the lambda definable elements of the minimal model of type A are precisely those invariant under the transposition of the elements of the ground domain. Conclude that unification of terms of type of rank 1 is decidable.

Chapter 5

Extensions

5.1. Lambda delta

In this section the simply typed lambda calculus will be extended by constants δ ($= \delta_{A,B}$), for every $A, B \in \mathbb{T}(o)$. Church [1940] used this extension to introduce a logical system called “the simple theory of types”, based on classical logic. (The system is also referred to as “higher order logic”, and denoted by HOL.) We will introduce a variant of this system denoted by Δ . The intuitive idea is that $\delta = \delta_{A,B}$ satisfies for all for all $a, a' : A, b, b' : B$

$$\begin{aligned}\delta aa'bb' &= b && \text{if } a = a'; \\ &= b' && \text{if } a \neq a'.$$

Therefore the type of the new constants is as follows

$$\delta_{A,B} : A \rightarrow A \rightarrow B \rightarrow B \rightarrow B.$$

The set of typed terms *à la Church* with as type atoms only o will be denoted by $\Lambda\delta$; its elements will be called *$\lambda\delta$ -terms*. The classical variant of the theory in which each term and variable carries its unique type will be considered only, but will suppress types whenever there is little danger of confusion.

The theory Δ is a strong logical system, in fact stronger than each of the 1st, 2nd, 3rd, ... order logics. It turns out that because of the presence of δ 's an arbitrary formula of Δ is equivalent to an equation. This fact will be an incarnation of the comprehension principle. It is because of the δ 's that Δ is powerful, less so because of the presence of quantification over elements of arbitrary types. Moreover, the set of equational consequences of Δ can be axiomatized by a finite subset. These are the main results in this section. It is an open question whether there is a natural (decidable) notion of reduction that is confluent and has as convertibility relation exactly these equational consequences. Since the decision problem for (higher order) predicate logic is undecidable, this notion of reduction will be non-terminating.

Higher Order Logic

In the following logical system terms are elements of Λ_o^{CL} . Formulas are built up from equations between terms of the same type using implication (\supset) and typed quantification ($\forall x^A.\varphi$). Absurdity is defined by $\perp \equiv (\mathbf{K} = \mathbf{K}_*)$, where $\mathbf{K} \equiv \lambda x^o y^o.x$, $\mathbf{K}_* \equiv \lambda x^o y^o.y$. and negation by $\neg\varphi \equiv \varphi \supset \perp$. By contrast to other sections in this book Γ stand for a set of formulas. Finally $\Gamma \vdash \varphi$ is defined by the following axioms and rules. Variables always have to be given types such that the terms involved are typable and have the same type if they occur in one equation. Below Γ is a set of formulas, and

$FV(\Gamma) = \{x \mid x \in FV(\varphi), \varphi \in \Gamma\}$. M, N, L, P, Q are terms.

Δ : Higher Order Logic	
$\Gamma \vdash (\lambda x.M)N = M[x = N]$	(beta)
$\Gamma \vdash \lambda x.Mx = M, x \notin FV(M)$	(eta)
$\Gamma \vdash M = M$	(reflexivity)
$\Gamma \vdash M = N$	(symmetry)
$\Gamma \vdash N = M$	
$\Gamma \vdash M = N, \Gamma \vdash N = L$	(transitivity)
$\Gamma \vdash M = L$	
$\Gamma \vdash M = N, \Gamma \vdash P = Q$	(cong-app)
$\Gamma \vdash MP = NQ$	
$\Gamma \vdash M = N$	(cong-abs)
$\Gamma \vdash \lambda x.M = \lambda x.N \quad x \notin FV(\Gamma)$	
$\frac{\varphi \in \Gamma}{\Gamma \vdash \varphi}$	(axiom)
$\Gamma \vdash \varphi \supset \psi \quad \Gamma \vdash \varphi$	$(\supset\text{-elim})$
$\Gamma \vdash \psi$	
$\Gamma, \varphi \vdash \psi$	$(\supset\text{-intr})$
$\Gamma \vdash \varphi \supset \psi$	
$\Gamma \vdash \forall x^A.\varphi$	$(\forall\text{-elim})$
$\Gamma \vdash \varphi[x = M] \quad M \in \Lambda(A)$	
$\Gamma \vdash \varphi$	$(\forall\text{-intr})$
$\Gamma \vdash \forall x^A.\varphi \quad x^A \notin FV(\Gamma)$	
$\Gamma, M \neq N \vdash \perp$	(classical)
$\Gamma \vdash M = N$	
$\Gamma \vdash M = N \supset \delta MNPQ = P$	(delta _L)
$\Gamma \vdash M \neq N \supset \delta MNPQ = Q$	(delta _R)

Provability in this system will be denoted by $\Gamma \vdash_{\Delta} \varphi$.

5.1.1. DEFINITION. The other logical connectives are introduced in the usual classical

manner.

$$\begin{aligned}\varphi \vee \psi &::= \neg\varphi \supset \psi; \\ \varphi\psi &::= \neg(\neg\varphi \vee \neg\psi); \\ \exists x^A.\varphi &::= \neg\forall x^A.\neg\varphi.\end{aligned}$$

5.1.2. LEMMA. *For all formulas of Δ one has*

$$\perp \vdash \varphi.$$

PROOF. By induction on the structure of φ . If $\varphi \equiv (M = N)$, then observe that by (eta)

$$\begin{aligned}M &= \lambda\vec{x}.M\vec{x} = \lambda\vec{x}.\mathbf{K}(M\vec{x})(N\vec{x}), \\ N &= \lambda\vec{x}.N\vec{x} = \lambda\vec{x}.\mathbf{K}_*(M\vec{x})(N\vec{x}),\end{aligned}$$

where the \vec{x} are such that the type of $M\vec{x}$ is o . Hence $\perp \vdash M = N$, since $\perp \equiv (\mathbf{K} = \mathbf{K}_*)$. If $\varphi \equiv (\psi \supset \chi)$ or $\varphi \equiv \forall x^A.\psi$, then the result follows immediately from the induction hypothesis. ■

5.1.3. PROPOSITION. $\delta_{A,B}$ can be defined from $\delta_{A,o}$.

PROOF. Indeed, if we only have $\delta_{A,o}$ (with their properties) and define

$$\delta_{A,B} = \lambda mnpq\vec{x}.\delta_{A,o}mn(p\vec{x})(q\vec{x}),$$

then all $\delta_{A,B}$ satisfy the axioms. ■

The rule (classical) is equivalent to

$$\neg\neg(M = N) \supset M = N.$$

In this rule the terms can be restricted to type o and the same theory Δ will be obtained.

5.1.4. PROPOSITION. *Suppose that in the formulation of Δ one requires*

$$\Gamma, \neg\neg(M = N) \vdash_{\Delta} \perp \Rightarrow \Gamma \vdash_{\delta} M = N \tag{1}$$

only for terms x, y of type o . Then (1) holds for terms of all types.

PROOF. By (1) we have $\neg\neg M = N \supset M = N$ for terms of type o . Assume $\neg\neg(M = N)$, with M, N of arbitrary type, in order to show $M = N$. We have

$$M = N \supset M\vec{x} = N\vec{x},$$

for all fresh \vec{x} such that the type of $M\vec{x}$ is o . By taking the contrapositive twice we obtain

$$\neg\neg(M = N) \supset \neg\neg(M\vec{x} = N\vec{x}).$$

Therefore by assumption and (1) we get $M\vec{x} = N\vec{x}$. But then by (cong-abs) and (eta) it follows that $M = N$. ■

5.1.5. PROPOSITION. *For all formulas φ one has*

$$\vdash_{\Delta} \neg\neg\varphi \supset \varphi.$$

PROOF. Induction on the structure of φ . If φ is an equation, then this is a rule of the system Δ . If $\varphi \equiv \psi \supset \chi$, then by the induction hypothesis one has $\vdash_{\Delta} \neg\neg\chi \supset \chi$ and we have the following derivation

$$\begin{array}{c}
 \frac{[\psi \supset \chi]^1 \quad [\psi]^3}{\chi \quad [\neg\chi]^2} \\
 \frac{\perp}{\neg(\psi \supset \chi)} \quad 1 \quad \frac{[\neg\neg(\psi \supset \chi)]^4}{\vdots} \\
 \frac{\perp}{\neg\neg\chi} \quad 2 \quad \frac{\neg\neg\chi \supset \chi}{3} \\
 \frac{\psi \supset \chi}{\neg\neg(\psi \supset \chi) \supset \psi \supset \chi} \quad 4
 \end{array}$$

for $\neg\neg(\psi \supset \chi) \supset (\psi \supset \chi)$. If $\varphi \equiv \forall x.\psi$, then by the induction hypothesis $\vdash_{\Delta} \neg\neg\psi(x) \supset \psi(x)$. Now we have a similar derivation

$$\begin{array}{c}
 \frac{[\forall x.\psi(x)]^1}{\psi(x) \quad [\neg\psi(x)]^2} \\
 \frac{\perp}{\neg\forall x.\psi(x)} \quad 1 \quad \frac{[\neg\neg\forall x.\psi(x)]^3}{\vdots} \\
 \frac{\perp}{\neg\neg\psi(x)} \quad 2 \quad \frac{\neg\neg\psi(x) \supset \psi(x)}{\vdots} \\
 \frac{\psi(x)}{\forall x.\psi(x)} \quad 3 \\
 \neg\neg\forall x.\psi(x) \supset \forall x.\psi(x)
 \end{array}$$

for $\neg\neg\forall x.\psi(x) \supset \forall x.\psi(x)$. ■

Now we will derive some equations in Δ that happen to be strong enough to provide an equational axiomatization of the equational part of Δ .

5.1.6. PROPOSITION. *The following equations hold universally (for those terms such that*

the equations make sense).

$$\begin{aligned}
\delta MMPQ &= P && (\delta\text{-identity}); \\
\delta MNPP &= P && (\delta\text{-reflexivity}); \\
\delta MNMN &= N && (\delta\text{-hypothesis}); \\
\delta MNPQ &= \delta NMPQ && (\delta\text{-symmetry}); \\
F(\delta MNPQ) &= \delta MN(FP)(FQ) && (\delta\text{-monotonicity}); \\
\delta MN(P(\delta MN))(Q(\delta MN)) &= \delta MN(PK)(QK_*) && (\delta\text{-transitivity}).
\end{aligned}$$

PROOF. We only show δ -reflexivity, the proof of the other assertions being similar. By the δ axioms one has

$$\begin{aligned}
M = N &\vdash \delta MNPP = P; \\
M \neq N &\vdash \delta MNPP = P.
\end{aligned}$$

By the “contrapositive” of the first statement one has $\delta MNPP \neq P \vdash M \neq N$ and hence by the second statement $\delta MNPP \neq P \vdash \delta MNPP = P$. So in fact $\delta MNPP \neq P \vdash \perp$, but then $\vdash \delta MNPP = P$, by the classical rule. ■

5.1.7. DEFINITION. The equational theory δ consists of equations between $\lambda\delta$ -terms of the same type, axiomatized as follows. (As usual the axioms and rules are assumed to hold universally, i.e. the free variables may be replaced by arbitrary terms. In the

following \mathcal{E} denotes a set of equations between $\lambda\delta$ -terms of the same type.

δ : Equational version of Δ	
$\mathcal{E} \vdash (\lambda x.M)N = M[x = N]$	(β)
$\mathcal{E} \vdash \lambda x.Mx = M, x \notin \text{FV}(M)$	(η)
$\mathcal{E} \vdash M = M$	(reflexivity)
$\mathcal{E} \vdash M = N$	(symmetry)
$\mathcal{E} \vdash N = M$	
$\mathcal{E} \vdash M = N, \mathcal{E} \vdash N = L$	(transitivity)
$\mathcal{E} \vdash M = L$	
$\mathcal{E} \vdash M = N, \mathcal{E} \vdash P = Q$	(cong-app)
$\mathcal{E} \vdash MP = NQ$	
$\mathcal{E} \vdash M = N$	(cong-abs)
$\mathcal{E} \vdash \lambda x.M = \lambda x.N \quad x \notin \text{FV}(\mathcal{E})$	
$\mathcal{E} \vdash \delta MMPQ = P$	(δ -identity)
$\mathcal{E} \vdash \delta MNPP = P$	(δ -reflexivity)
$\mathcal{E} \vdash \delta MNMN = N$	(δ -hypothesis)
$\mathcal{E} \vdash \delta MNPQ = \delta NMPQ$	(δ -symmetry)
$\mathcal{E} \vdash F(\delta MNPQ) = \delta MN(FP)(FQ)$	(δ -monotonicity)
$\mathcal{E} \vdash \delta MN(P(\delta MN))(Q(\delta MN)) = \delta MN(PK)(QK_*)$	(δ -transitivity)

The system δ may be given more conventionally by leaving out all occurrences of $\mathcal{E} \vdash_\delta$ and replacing in the rule (cong-abs) the proviso “ $x \notin \text{FV}(\mathcal{E})$ ” by “ x not occurring in any assumption on which $M = N$ depends”.

There is a canonical map from formulas to equations, preserving provability in Δ .

5.1.8. DEFINITION. (i) For an equation $E \equiv (M = N)$ in Δ , write $E.L ::= M$ and $E.R ::= N$.

(ii) Define for a formula φ the corresponding equation φ^+ as follows.

$$\begin{aligned}
(M = N)^+ & ::= M = N; \\
(\psi \supset \chi)^+ & ::= (\delta(\psi^+.L)(\psi^+.R)(\chi^+.L)(\chi^+.R) = \chi^+.R); \\
(\forall x.\psi)^+ & ::= (\lambda x.\psi^+.L = \lambda x.\psi^+.R).
\end{aligned}$$

So, if $\psi^+ \equiv (M = N)$ and $\chi^+ \equiv (P = Q)$, then

$$\begin{aligned}
(\psi \supset \chi)^+ & ::= (\delta MNPQ = Q); \\
(\neg \psi)^+ & ::= (\delta MNKK_* = K_*); \\
(\forall x.\psi)^+ & ::= (\lambda x.M = \lambda x.N).
\end{aligned}$$

(iii) If Γ is a set of formulas, then $\Gamma^+ = \{\varphi^+ \mid \varphi \in \Gamma\}$.

5.1.9. THEOREM. *For every formula φ one has*

$$\vdash_{\Delta} \varphi \leftrightarrow \varphi^+.$$

PROOF. By induction on the structure of φ . If φ is an equation, then this is trivial. If $\varphi \equiv \psi \supset \chi$, then the statement follows from

$$\vdash_{\Delta} (M = N \supset P = Q) \leftrightarrow (\delta MNPQ = Q).$$

If $\varphi \equiv \forall x.\psi$, then this follows from

$$\vdash_{\Delta} \forall x.(M = N) \leftrightarrow (\lambda x.M = \lambda x.N). \blacksquare$$

We will show now that Δ is conservative over δ . The proof occupies 5.1.10-5.1.17

5.1.10. LEMMA. (i) $\vdash_{\delta} \delta MNPQz = \delta MN(Pz)(Qz)$.

(ii) $\vdash_{\delta} \delta MNPQ = \lambda z.\delta MN(Pz)(Qz)$.

(iii) $\vdash_{\delta} \lambda z.\delta MNPQ = \delta MN(\lambda z.P)(\lambda z.Q)$.

PROOF. (i) Use δ -monotonicity $F(\delta MNPQ) = \delta MN(FP)(FQ)$ for $F = \lambda x.xz$.

(ii) By (i) and (η) .

(iii) By (ii) applied with $P := \lambda z.P$ and $Q := \lambda z.Q$. \blacksquare

5.1.11. LEMMA. (i) $\delta MNPQ = Q \vdash_{\delta} \delta MNQP = P$.

(ii) $\delta MNPQ = Q, \delta MNQR = R \vdash_{\delta} \delta MNPR = R$.

(iii) $\delta MNPQ = Q, \delta MNUV = V \vdash_{\delta} \delta MN(PU) = (QV)$.

PROOF. (i) $P = \delta MNPP$

$$= \delta MN(KPQ)(K_*QP)$$

$$= \delta MN(\delta MNPQ)(\delta MNQP), \quad \text{by } (\delta\text{-transitivity}),$$

$$= \delta MNQ(\delta MNQP), \quad \text{by assumption,}$$

$$= \delta MNQ(K_*QP), \quad \text{by } (\delta\text{-transitivity}),$$

$$= \delta MNQP.$$

(ii) $R = \delta MNQR,$ by assumption,

$$= \delta MN(\delta MNPQ)(\delta MNQR), \quad \text{by assumption,}$$

$$= \delta MN(KPQ)(K_*QR), \quad \text{by } (\delta\text{-transitivity}),$$

$$= \delta MNPR.$$

(iii) Assuming $\delta MNPQ = Q$ and $\delta MNUV = V$ we obtain by $(\delta\text{-transitivity})$ applied twice $\delta MN(PU)(QV) = (QV)$ and $\delta MN(PU)(QV) = (QV)$. Hence the result $\delta MN(PU)(QV) = QV$ follows by (ii). \blacksquare

5.1.12. PROPOSITION (Deduction theorem I). *Let \mathcal{E} be a set of equations. Then*

$$\mathcal{E}, M = N \vdash_{\delta} P = Q \Rightarrow \mathcal{E} \vdash_{\delta} \delta MNPQ = Q.$$

PROOF. By induction on the derivation of $\mathcal{E}, M = N \vdash_\delta P = Q$. If $P = Q$ is an axiom of δ or in \mathcal{E} , then $\mathcal{E} \vdash_\delta P = Q$ and hence $\mathcal{E} \vdash_\delta \delta MNPQ = \delta MNQQ = Q$. If $(P = Q) \equiv (M = N)$, then $\mathcal{E} \vdash_\delta \delta MNPQ \equiv \delta MNMN = N \equiv N$. If $P = Q$ follows directly from $\mathcal{E}, N = M \vdash_\delta V = U$, then by the induction hypothesis one has $\mathcal{E} \vdash_\delta \delta MNQP = U$. But then by lemma 5.1.11(i) one has $\mathcal{E} \vdash_\delta \delta MNPQ = Q$. If $P = Q$ follows by (transitivity), (cong-app) or (cong-abs), then the result follows from the induction hypothesis, using lemma 5.1.11(ii), (iii) or lemma 5.1.10(iii) respectively. ■

5.1.13. LEMMA. (i) $\vdash_\delta \delta MN(\delta MNPQ)P = P$.

(ii) $\vdash_\delta \delta MNQ(\delta MNPQ) = Q$.

PROOF. (i) By (δ -transitivity) one has

$$\delta MN(\delta MNPQ)P = \delta MN(KPQ)P = \delta MNPP = P.$$

(ii) Similarly. ■

5.1.14. LEMMA. (i) $\vdash_\delta \delta KK_* = K_*$;
(ii) $\vdash_\delta \delta MNKK_* = \delta MN$;
(iii) $\vdash_\delta \delta(\delta MN)K_*PQ = \delta MNQP$;
(iv) $\vdash_\delta \delta(\delta MNKK_*)K_*(\delta MNPQ)Q = P$.

PROOF. (i) $K_* = \delta KK_*K_*$, by (δ -hypothesis),
 $= \lambda ab. \delta KK_*(Kab)(K_*ab)$, by (η) and lemma 5.1.10(ii),
 $= \lambda ab. \delta KK_*ab$
 $= \delta KK_*$, by (η).
(ii) $\delta MNKK_* = \delta MN(\delta MN)(\delta MN)$, by (δ -transitivity),
 $= \delta MN$, by (δ -reflexivity).
(iii) $\delta MNQP = \delta MN(\delta KK_*PQ)(\delta K_*K_*PQ)$, by (i), (δ -identity),
 $= \delta MN(\delta(\delta MN)K_*PQ)(\delta(\delta MN)K_*PQ)$, by (δ -transitivity),
 $= \delta(\delta MN)K_*PQ$, by (δ -reflexivity).
(iv) By (ii) and (iii) we have

$$\delta(\delta MNKK_*)K_*(\delta MNPQ)Q = \delta(\delta MN)K_*(\delta MNPQ)Q = \delta MNQ(\delta MNPQ).$$

Therefore we are done by lemma 5.1.13(ii). ■

5.1.15. LEMMA. (i) $\delta MN = K \vdash_\delta M = N$;
(ii) $\delta MNK_*K = K_* \vdash_\delta M = N$.
(iii) $\delta(\delta MNKK_*)K_*KK_* = K_* \vdash_\delta M = N$.

PROOF. (i) $M = KMN = \delta MNMN = N$, by assumption and (δ -hypothesis).

(ii) Suppose $\delta MNK_*K = K_*$. Then by lemma 5.1.10(i) and (δ -hypothesis)

$$M = K_*NM = \delta MNK_*KNM = \delta MN(K_*NM)(KNM) = \delta MNMN = N.$$

(iii) By lemma 5.1.14(ii) and (iii)

$$\delta(\delta MNKK_*)K_*KK_* = \delta(\delta MN)K_*KK_* = \delta MNK_*K.$$

Hence by (ii) we are done. ■

5.1.16. EXERCISE. Prove in δ the following equations.

- (i) $\delta MNK_*K = \delta(\delta MN)K_*$.
- (ii) $\delta(\lambda z.\delta(Mz)(Nz))(\lambda z.K) = \delta MN$.

[Hint. Start observing that $\delta(Mz)(Nz)(Mz)(Nz) = Nz$.]

Now we are able to prove the conservativity of Δ over δ .

5.1.17. THEOREM. For equations \mathcal{E}, E and formulas Γ, φ of Δ one has the following.

- (i) $\Gamma \vdash_{\Delta} \varphi \iff \Gamma^+ \vdash_{\delta} \varphi^+$.
- (ii) $\mathcal{E} \vdash_{\Delta} E \iff \mathcal{E} \vdash_{\delta} E$.

PROOF. (i) (\Rightarrow) Suppose $\Gamma \vdash_{\Delta} \varphi$. By induction on this proof in Δ we show that $\Gamma^+ \vdash_{\delta} \varphi^+$. Case 1. φ is in Γ . Then $\varphi^+ \in \Gamma^+$ and we are done. Case 2. φ is an equational axiom. Then the result holds since δ has more equational axioms than Δ . Case 3. φ follows from an equality rule in Δ . Then the result follows from the induction hypothesis and the fact that δ has the same equational deduction rules. Case 4. φ follows from $\Gamma \vdash_{\Delta} \psi$ and $\Gamma \vdash_{\Delta} \psi \supset \varphi$. By the induction hypothesis $\Gamma^+ \vdash_{\delta} (\psi \supset \varphi)^+ \equiv (\delta MNPQ = Q)$ and $\Gamma^+ \vdash_{\delta} \psi^+ \equiv (M = N)$, where $\psi^+ \equiv (M = N)$ and $\varphi^+ \equiv (P = Q)$. Then $\Gamma^+ \vdash_{\delta} U = \delta MMPQ = Q$, i.e. $\Gamma^+ \vdash_{\delta} \varphi^+$. Case 5. $\varphi \equiv (\chi \supset \psi)$ and follows by an $(\supset$ -intro) from $\Gamma, \chi \vdash_{\Delta} \psi$. By the induction hypothesis $\Gamma^+, \chi^+ \vdash_{\delta} \psi^+$ and we can apply the deduction theorem 5.1.12. In the cases that φ is introduced by a $(\forall$ -elim) or $(\forall$ -intro), the result follows easily from the induction hypothesis and axiom (β) or the rule (cong-abs). One needs that $FV(\Gamma) = FV(\Gamma^+)$. Case 8. $\varphi \cong (M = N)$ and follows from $\Gamma, M \neq N \vdash_{\Delta} \perp$ using the rule (classical). By the induction hypothesis $\Gamma^+, (M \neq N)^+ \vdash_{\delta} K = K_*$. By the deduction theorem it follows that $\Gamma^+ \vdash_{\delta} \delta(\delta MNKK_*)K_*KK_* = K_*$. Hence we are done by lemma 5.1.15(iii). Case 9. φ is the axiom $(M = N \supset \delta MNPQ = P)$. Then φ^+ is provable in δ by lemma 5.1.13(i). Case 10. φ is the axiom $(M \neq N \supset \delta MNPQ = Q)$. Then φ^+ is provable in δ by lemma 5.1.14(iv).

(\Leftarrow) By the fact that δ is a subtheory of Δ and theorem 5.1.9.

- (ii) By (i) and the fact that $E^+ \equiv E$. ■

n-th Order Logic

In this subsection some results will be sketched but not (completely) proved.

5.1.18. DEFINITION. (i) The system Δ without the two delta rules is denoted by Δ^- .

(ii) $\Delta(n)$ is Δ^- extended by the two delta rules restricted to $\delta_{A,B}$'s with $\text{rank}(A) \leq n$.

(iii) Similarly $\delta(n)$ is the theory δ in which only terms $\delta_{A,B}$ are used with $\text{rank}(A) \leq n$.

(iv) The rank of a formula φ is $\mathbf{rank}(\varphi) = \max\{\mathbf{rank}(\delta) \mid \delta \text{ occurs in } \varphi\}$.

In the applications section we will show that $\Delta(n)$ is essentially n -th order logic.

The relation between Δ and δ that we have seen also holds level by level. We will only state the relevant results, the proofs being similar, but using as extra ingredient the proof-theoretic normalization theorem for Δ . This is necessary, since a proof of a formula of rank n may use *a priori* formulas of arbitrarily high rank. By the normalization theorem this is not the case.

A natural deduction is called *normal* if there is no (\forall -intro) immediately followed by a (\forall -elim), nor a (\supset -intro) immediately followed by a (\supset -elim). If a reduction is not normal, then one can make Prawitz deductions reduction as follows.

$$\frac{\frac{\frac{\vdots \Sigma}{\varphi}}{\forall x.\varphi}}{\varphi[x := M]} \Rightarrow \frac{\vdots \Sigma[x := M]}{\varphi[x := M]}$$

$$\frac{\frac{\vdots \Sigma_2}{\varphi} \quad \frac{\frac{[\varphi] \vdots \Sigma_1}{\psi}}{\varphi \supset \psi}}{\psi} \Rightarrow \frac{[\varphi] \vdots \Sigma_2}{\psi}$$

5.1.19. THEOREM. *Every reduction sequence of Δ deductions terminates in a unique normal form.*

PROOF. This has been proved essentially in Prawitz [1965]. The higher order quantifiers pose no problems. ■

NOTATION. (i) Let Γ_δ be the set of universal closures of

$$\begin{aligned} \delta mmpq &= p, \\ \delta mnpp &= p, \\ \delta mnmn &= n, \\ \delta mn pq &= \delta nmpq, \\ f(\delta mn pq) &= \delta mn(fp)(fq), \\ \delta mn(p(\delta mn))(q(\delta mn)) &= \delta mn(pK)(qK_*). \end{aligned}$$

(ii) Let $\Gamma_{\delta(n)} = \{\varphi \in \Gamma_\delta \mid \mathbf{rank}(\varphi) \leq n\}$.

5.1.20. PROPOSITION (Deduction theorem II). *Let \mathcal{S} be a set of equations or negations of equations in Δ . Then for every n*

- (i) $\mathcal{S}, \Gamma_{\delta(n)}, M = N \vdash_{\Delta(n)} P = Q \Rightarrow \mathcal{S}, \Gamma_{\delta(n)} \vdash_{\Delta(n)} \delta MNPQ = Q.$
- (ii) $\mathcal{S}, \Gamma_{\delta(n)}, M \neq N \vdash_{\Delta(n)} P = Q \Rightarrow \mathcal{S}, \Gamma_{\delta(n)} \vdash_{\Delta(n)} \delta MNPQ = P.$

PROOF. In the same style as the proof of proposition 5.1.12, but now using the normalization theorem 5.1.19. ■

5.1.21. LEMMA. *Let \mathcal{S} be a set of equations or negations of equations in Δ . Let \mathcal{S}^* be \mathcal{S} with each $M \neq N$ replaced by $\delta MNKK_* = K_*$. Then we have the following.*

- (i) $\mathcal{S}, M = N \vdash_{\Delta(n)} P = Q \Rightarrow \mathcal{S}^* \vdash_{\delta(n)} \delta MNPQ = Q.$
- (ii) $\mathcal{S}, M \neq N \vdash_{\Delta(n)} P = Q \Rightarrow \mathcal{S}^* \vdash_{\delta(n)} \delta MNPQ = P.$

PROOF. By induction on derivations. ■

5.1.22. THEOREM. $\mathcal{E} \vdash_{\Delta(n)} E \iff \mathcal{E} \vdash_{\delta(n)} E.$

PROOF. (\Rightarrow) By taking $\mathcal{S} = \mathcal{E}$ and $M \equiv N \equiv x$ in lemma 5.1.21(i) one obtains $\mathcal{E} \vdash_{\delta(n)} \delta xxPQ = Q$. Hence $\mathcal{E} \vdash_{\delta(n)} P = Q$, by (δ -identity). (\Leftarrow) Trivial. ■

5.1.23. THEOREM. (i) *Let $\text{rank}(\mathcal{E}, M = N) \leq 1$. Then*

$$\mathcal{E} \vdash_{\Delta} M = N \iff \mathcal{E} \vdash_{\delta(1)} M = N.$$

(ii) *Let Γ, A be a first-order sentences. Then*

$$\Gamma \vdash_{\Delta} A \iff \Gamma \vdash_{\delta(1)} A^+.$$

PROOF. See Statman [2000]. ■

In Statman [2000] it is also proved that $\Delta(0)$ is decidable. Since $\Delta(n)$ for $n \geq 1$ is at least first order predicate logic, these systems are undecidable. It is observed in Gödel [1931] that the consistency of $\Delta(n)$ can be proved in $\Delta(n+1)$.

5.2. Surjective pairing 21.8.2006:951

A *pairing* on a set X consists of three maps π, π_1, π_2 such that

$$\begin{aligned} \pi &: X \rightarrow X \rightarrow X \\ \pi_i &: X \rightarrow X \end{aligned}$$

and for all $x_1, x_2 \in X$ one has

$$\pi_i(\pi x_1 x_2) = x_i.$$

Using a pairing one can pack two or more elements of X into one element:

$$\begin{aligned} \pi xy &\in X, \\ \pi x(\pi yz) &\in X. \end{aligned}$$

A pairing on X is called *surjective* if one also has for all $x \in X$

$$\pi(\pi_1 x)(\pi_2 x) = x.$$

This is equivalent to saying that every element of X is a pair. Using a (surjective) pairing datastructures can be encoded.

The main results in this section are the following. 1. After adding a surjective pairing to λ_{\rightarrow}^o , the resulting system λ_{SP} becomes Hilbert-Post complete. This means that an equation between terms is either provable or inconsistent. 2. Every recursively enumerable set \mathcal{X} of terms that is closed under provable equality is Diophantine, i.e. satisfies for some terms F, G

$$M \in \mathcal{X} \iff \exists N FMN = GMN.$$

Both results will be proved by introducing Cartesian monoids and studying freely generated ones.

The system λ_{SP}

We define λ_{SP} as an extension of the simply typed lambda calculus λ_{\rightarrow}^o .

5.2.1. DEFINITION. (i) The set of *types* of λ_{SP} , notation $\mathbb{T} = \mathbb{T}(\lambda_{SP})$, is the same as $\mathbb{T}(\lambda_{\rightarrow}^o)$: $\mathbb{T} = o \mid \mathbb{T} \rightarrow \mathbb{T}$.

(ii) The *terms* of λ_{SP} , notation Λ_{SP} (or $\Lambda_{SP}(A)$ for terms of a certain type A or Λ^{\emptyset} , $\Lambda_{SP}^{\emptyset}(A)$ for closed terms), are obtained from λ_{\rightarrow}^o by adding to the formation of terms the constants $\pi : 1_2 = o^2 \rightarrow o$, $\pi_1 : 1$, $\vdash : 1$.

(iii) Equality for λ_{SP} is axiomatized by β , η and the following scheme. For all $M, M_1, M_2 : o$

$$\begin{aligned} \pi_i(\pi M_1 M_2) &= M_i; \\ \pi(\pi_1 M)(\pi_2 M) &= M. \end{aligned}$$

(iv) A notion of reduction SP is introduced on λ_{SP} -terms by the following contraction rules: for all $M, M_1, M_2 : o$

$$\begin{aligned} \pi_i(\pi M_1 M_2) &\rightarrow M_i; \\ \pi(\pi_1 M)(\pi_2 M) &\rightarrow M. \end{aligned}$$

Usually we will consider SP in combination of $\beta\eta$, obtaining $\beta\eta SP$.

5.2.2. THEOREM. *The conversion relation $=_{\beta\eta SP}$, generated by the notion of reduction $\beta\eta SP$, coincides with that of the theory λ_{SP} .*

PROOF. As usual. ■

For objects of higher type pairing can be defined in terms of π, π_1, \vdash in the following way.

5.2.3. DEFINITION. For every type $A \in \mathbb{T}$ we define $\pi^A : A \rightarrow A \rightarrow A$, $\pi_i : A \rightarrow A$ as follows, cf. definition ??.

$$\begin{aligned}\pi^o &\equiv \pi; \\ \pi_i^o &\equiv \pi_i; \\ \pi^{A \rightarrow B} &\equiv \lambda xy (A \rightarrow B) \lambda z A. \pi^B(xz)(yz); \\ \pi_i^{A \rightarrow B} &\equiv \lambda x (A \rightarrow B) \lambda z A. \pi_i^B(xz).\end{aligned}$$

Sometimes we may suppress type annotations in π^A, π_1^A, \vdash^A , but the types can always and unambiguously be reconstructed from the context.

The defined constants for higher type pairing can easily be shown to be a surjective pairing also.

5.2.4. PROPOSITION. Let $\pi = \pi^A, \pi_i = \pi_i^A$. Then for $M, M_1, M_2 \in \Lambda_{SP}(A)$

$$\begin{aligned}\pi(\pi_1 M)(\vdash M) &\rightarrow_{\beta\eta SP} M; \\ \pi_i(\pi M_1 M_2) &\rightarrow_{\beta\eta SP} M_i, \quad (i = 1, 2).\end{aligned}$$

PROOF. By induction on the type A . ■

Note that the above reductions may involve more than one step, typically additional $\beta\eta$ -steps.

Now we will show that the notion of reduction $\beta\eta SP$ is confluent.

5.2.5. LEMMA. The notion of reduction $\beta\eta SP$ satisfies WCR.

PROOF. By the critical pair lemma of Mayr and Nipkow [1998]. But a simpler argument is possible, since SP reductions only reduce to terms of type o that did already exist and hence cannot create any redexes. ■

5.2.6. LEMMA. (i) The notion of reduction SP is SN.

(ii) If $M \rightarrow_{\beta\eta SP} N$, then there exists P such that $M \rightarrow_{\beta\eta} P \rightarrow_{SP} N$.

(iii) The notion of reduction $\beta\eta SP$ is SN.

PROOF. (i) Since SP -reductions are strictly decreasing.

(ii) Show $M \rightarrow_{SP} L \rightarrow_{\beta\eta} N \Rightarrow \exists L' M \rightarrow_{\beta\eta} L' \rightarrow_{\beta\eta SP} N$. Then (ii) follows by a staircase diagram chase.

(iii) By (i), the fact that $\beta\eta$ is SN and a staircase diagram chase, possible by (ii). ■

5.2.7. PROPOSITION. $\beta\eta SP$ is CR.

PROOF. By lemma 5.2.6(iii) and Newman's Lemma 5.3.14. ■

5.2.8. DEFINITION. (i) An SP -retraction pair from A to B is a pair of terms $M A \rightarrow B$ and $N B \rightarrow A$ such that $N \circ M =_{\beta\eta SP} \vdash_A$.

(ii) A is a *SP-retract* of B , notation $A \triangleleft_{SP} B$, if there is an *SP-retraction* pair between A and B .

The proof of the following result is left as an exercise to the reader.

5.2.9. PROPOSITION. *Define types N_n as follows. $N_0 \equiv o$ and $N_{n+1} \equiv N_n \rightarrow N_n$. Then for every type A , one has $A \triangleleft_{SP} N_{rank(A)}$.*

Cartesian monoids

We start with the definition of a Cartesian monoid, introduced in Scott [1980] and, independently, in Lambek [1980].

5.2.10. DEFINITION. (i) A *Cartesian monoid* is a structure

$$\mathcal{C} = \langle \mathcal{M}, *, I, L, R, \langle \cdot, \cdot \rangle \rangle$$

such that $(\mathcal{M}, *, I)$ is a monoid ($*$ is associative and I is a two sided unit), $L, R \in \mathcal{M}$ and $\langle \cdot, \cdot \rangle : \mathcal{M}^2 \rightarrow \mathcal{M}$ and satisfy

$$\begin{aligned} L * \langle x, y \rangle &= x \\ R * \langle x, y \rangle &= y \\ \langle x, y \rangle * z &= \langle x * z, y * z \rangle \\ \langle L, R \rangle &= I \end{aligned}$$

(ii) \mathcal{M} is called *trivial* if $L = R$.

Note that if \mathcal{M} is trivial, then it consists of only one element: for all $x, y \in \mathcal{M}$

$$x = L \langle x, y \rangle = R \langle x, y \rangle = y.$$

5.2.11. LEMMA. *The last axiom of the Cartesian monoids can be replaced equivalently by the surjectivity of the pairing:*

$$\langle L * x, R * x \rangle = x.$$

PROOF. First suppose $\langle L, R \rangle = I$. Then $\langle L * x, R * x \rangle = \langle L, R \rangle * x = I * x = x$. Conversely suppose $\langle L * x, R * x \rangle = x$, for all x . Then $\langle L, R \rangle = \langle L * \mathbf{l}, R * \mathbf{l} \rangle = \mathbf{l}$. ■

5.2.12. LEMMA. *Let \mathcal{M} be a Cartesian monoid. Then for all $x, y \in \mathcal{M}$*

$$L * x = L * y \ \& \ R * x = R * y \Rightarrow x = y.$$

PROOF. $x = \langle L * x, R * x \rangle = \langle L * y, R * y \rangle = y$. ■

A first example of a Cartesian monoid has as carrier set the closed $\beta\eta SP$ -terms of type $1 = o \rightarrow o$.

5.2.13. DEFINITION. Write for $M, N \in \Lambda_{SP}^\emptyset(1)$

$$\begin{aligned}\langle M, N \rangle &\equiv \pi^1 MN; \\ M \circ N &\equiv \lambda x \circ. M(Nx); \\ \mathbf{I} &\equiv \lambda x \circ. x; \\ \mathbf{L} &\equiv \pi_1^o; \\ \mathbf{R} &\equiv \vdash^o.\end{aligned}$$

Define

$$\mathcal{C}^0 = \langle \Lambda_{SP}^\emptyset(1) / \equiv_{\beta\eta SP}, \circ, \mathbf{I}, \mathbf{L}, \mathbf{R}, \langle \cdot, \cdot \rangle \rangle.$$

The reason to call this structure \mathcal{C}^0 and not \mathcal{C}^1 is that we will generalize it to \mathcal{C}^n being based on terms of the type $1^n \rightarrow 1$.

5.2.14. PROPOSITION. \mathcal{C}^0 is a non-trivial Cartesian monoid.

PROOF. For $x, y, z \vdash 1$ the following equations are valid in λ_{SP} .

$$\begin{aligned}\mathbf{I} \circ x &= x; \\ x \circ \mathbf{I} &= x; \\ \mathbf{L} \circ \langle x, y \rangle &= x; \\ \mathbf{R} \circ \langle x, y \rangle &= y; \\ \langle x, y \rangle \circ z &= \langle x \circ z, y \circ z \rangle; \\ \langle \mathbf{L}, \mathbf{R} \rangle &= \mathbf{I}. \blacksquare\end{aligned}$$

The third equation is intuitively right, if we remember that the pairing on type 1 is lifted pointwise from a pairing on type o ; that is, $\langle f, g \rangle = \lambda x. \pi(fx)(gx)$.

5.2.15. EXAMPLE. Let $[\cdot, \cdot]$ be any surjective pairing of natural numbers, with left and right projections $l, r : \mathbb{N} \rightarrow \mathbb{N}$. For example, we can take Cantor's well-known bijection¹ from \mathbb{N}^2 to \mathbb{N} . We can lift the pairing function to the level of functions by putting $\langle f, g \rangle(x) = [f(x), g(x)]$ for all $x \in \mathbb{N}$. Let I be the identity function and let \circ denote function composition. Then

$$\mathcal{N}^1 = \langle \mathbb{N} \rightarrow \mathbb{N}, I, \circ, l, r, \langle \cdot, \cdot \rangle \rangle.$$

is a non-trivial Cartesian monoid.

Now we will show that the equalities in the theory of Cartesian monoids are generated by a confluent rewriting system.

¹A variant of this function is used in Section 5.3 as a non-surjective pairing function $[x, y] + 1$, such that, deliberately, 0 does not encode a pair. This variant is specified in detail and explained in Figure 5.2.

5.2.16. DEFINITION. (i) Let T_{CM} be the terms in the signature of Cartesian monoids. Let T_{CM}^o be the closed such terms.

(ii) Consider the rewrite system CM on T_{CM} defined as follows.

$$\begin{aligned}
 L * \langle x, y \rangle &\rightarrow x \\
 R * \langle x, y \rangle &\rightarrow y \\
 \langle x, y \rangle * z &\rightarrow \langle x * z, y * z \rangle \\
 \langle L, R \rangle &\rightarrow I \\
 \langle L * x, R * x \rangle &\rightarrow x \\
 I * x &\rightarrow x \\
 x * I &\rightarrow x
 \end{aligned}$$

modulo the associativity axioms (i.e. terms like $f * (g * h)$ and $(f * g) * h$ are considered as the same).

5.2.17. PROPOSITION. (i) CM is WCR.

(ii) CM is SN.

(iii) CM is CR.

PROOF. (i) Examine all critical pairs. Modulo associativity there are many such pairs, but they all converge. Consider, as an example, the following reductions:

$$x * z \leftarrow (L * \langle x, y \rangle) * z = L * (\langle x, y \rangle * z) \rightarrow L * \langle x * z, y * z \rangle \rightarrow x * z$$

(ii) Interpret CM as integers by putting

$$\begin{aligned}
 \llbracket x \rrbracket_\rho &= \rho(x); \\
 \llbracket e \rrbracket_\rho &= 2, & \text{if } e \text{ is } L, R \text{ or } I; \\
 \llbracket e_1 * e_2 \rrbracket_\rho &= \llbracket e_1 \rrbracket_\rho \cdot \llbracket e_2 \rrbracket_\rho; \\
 \llbracket \langle e_1, e_2 \rangle \rrbracket_\rho &= \llbracket e_1 \rrbracket_\rho + \llbracket e_2 \rrbracket_\rho + 1.
 \end{aligned}$$

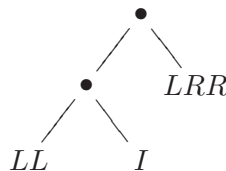
Then $\llbracket \cdot \rrbracket_\rho$ preserves associativity and

$$e \rightarrow_{CM} e' \Rightarrow \llbracket e \rrbracket_\rho > \llbracket e' \rrbracket_\rho.$$

Therefore CM is SN.

(iii) By (i), (ii) and Newmans lemma 5.3.14. ■

Closed terms in CM -nf can be represented as binary trees with strings of L, R (the empty string becomes I) at the leaves, for example



represents $\langle \langle L * L, I \rangle, L * R * R \rangle$. In such trees the subtree corresponding to $\langle L, R \rangle$ will not occur, since this term reduces to I .

The free Cartesian monoids $\mathcal{F}[x_1, \dots, x_n]$

5.2.18. DEFINITION. (i) The closed term model of the theory of Cartesian monoids consists of T_{CM} modulo $=_{\text{CM}}$ and is denoted by \mathcal{F} . It is the *free Cartesian monoid* with no generators.

(ii) The *free Cartesian monoid* over the generators \vec{x} , notation $\mathcal{F}[\vec{x}]$, is generated from I, L, R and the indeterminates \vec{x} using $*$ and $\langle -, - \rangle$. Usually $\vec{x} = x_1, \dots, x_n$ is finite.

5.2.19. PROPOSITION. (i) For all $a, b \in \mathcal{F}$ one has

$$a \neq b \Rightarrow \exists c, d \in \mathcal{F} [c * a * d = L \ \& \ c * b * d = R].$$

(ii) \mathcal{F} is simple: every homomorphism $g : \mathcal{F} \rightarrow \mathcal{M}$ to a non-trivial Cartesian monoid \mathcal{M} is injective.

PROOF. (i) We can assume that a, b are in normal form. The binary tree part of the normal form is called Δ . The Δ 's of a, b can be made to be congruent by expansions of the form $x \leftarrow \langle L * x, R * x \rangle$. The expanded trees are distinct in some leaf, which can be reached by a string of L 's and R 's joined by $*$. Thus there is such a string, say c , such that $c * a \neq c * b$ and both of these reduce to $\langle \rangle$ -free strings of L 's and R 's joined by $*$. We can also assume that neither of these strings is a suffix of the other, since c could be replaced by $L * c$ or $R * c$ (depending on an R or an L just before the suffix). Thus there are $\langle \rangle$ -free a', b' and integers k, l such that

$$\begin{aligned} c * a * \langle I, I \rangle^k * \langle R, L \rangle^l &= a' * L \quad \text{and} \\ c * b * \langle I, I \rangle^k * \langle R, L \rangle^l &= b' * R \end{aligned}$$

and there exist integers n and m , being the length of a' and of b' , respectively, such that

$$\begin{aligned} a' * L * \langle \langle I, I \rangle^n * L, \langle I, I \rangle^m * R \rangle &= L \quad \text{and} \\ b' * R * \langle \langle I, I \rangle^n * L, \langle I, I \rangle^m * R \rangle &= R \end{aligned}$$

Therefore we can set $d = \langle I, I \rangle^k * \langle R, L \rangle^l * \langle \langle I, I \rangle^n * L, \langle I, I \rangle^m * R \rangle$.

(ii) By (i) and the fact that \mathcal{M} is non-trivial. ■

Finite generation of $\mathcal{F}[x_1, \dots, x_n]$

Now we will show that $\mathcal{F}[x_1, \dots, x_n]$ is finitely generated as a monoid, i.e. from finitely many of its elements using the operation $*$ only.

5.2.20. NOTATION. In a monoid \mathcal{M} we define list-like left-associative and right-associative iterated $\langle \rangle$ -expressions of length > 0 as follows. Let the elements of \vec{x} range over \mathcal{M} .

$$\begin{aligned} \langle\langle x \rangle\rangle &\equiv x; \\ \langle\langle x_1, \dots, x_{n+1} \rangle\rangle &\equiv \langle\langle x_1, \dots, x_n \rangle, x_{n+1} \rangle, \quad n > 0; \\ \langle x \rangle &\equiv x; \\ \langle x_1, \dots, x_{n+1} \rangle &\equiv \langle x_1, \langle x_2, \dots, x_{n+1} \rangle \rangle, \quad n > 0. \end{aligned}$$

5.2.21. DEFINITION. Define $\mathcal{G} \subseteq \mathcal{F}$ as follows.

$$\mathcal{G} = \{\langle X * L, Y * L * R, Z * R * R \rangle \mid X, Y, Z \in \{L, R, I\}\} \cup \{\langle I, I, I \rangle\}.$$

Let $[\mathcal{G}]$ be the set generated from \mathcal{G} using $*$, i.e. the least subset of \mathcal{F} containing \mathcal{G} and closed under $*$. We will show that $[\mathcal{G}] = \mathcal{F}$.

5.2.22. LEMMA. Define a string to be an expression of the form $X_1 * \dots * X_n$, with $X_i \in \{L, R, I\}$. Then for all strings s, s_1, s_2, s_3 one has the following.

- (i) $\langle s_1, s_2, s_3 \rangle \in [\mathcal{G}]$.
- (ii) $s \in [\mathcal{G}]$.

PROOF. (i) Note that

$$\langle X * L, Y * L * R, Z * R * R \rangle * \langle s_1, s_2, s_3 \rangle = \langle X * s_1, Y * s_2, Z * s_3 \rangle.$$

Hence, starting from $\langle I, I, I \rangle \in \mathcal{G}$ every triple of strings can be generated because the X, Y, Z range over $\{L, R, I\}$.

(ii) Notice that

$$\begin{aligned} s &= \langle L, R \rangle * s \\ &= \langle L * s, R * s \rangle \\ &= \langle L * s, \langle L, R \rangle * R * s \rangle \\ &= \langle L * s, L * R * s, R * R * s \rangle, \end{aligned}$$

which is in $[\mathcal{G}]$ by (i). ■

5.2.23. LEMMA. Suppose $\langle s_1, \dots, s_n \rangle \in [\mathcal{G}]$. Then

- (i) $s_i \in [\mathcal{G}]$, for $1 \leq i \leq n$.
- (ii) $\langle s_1, \dots, s_n, \langle s_i, s_j \rangle \rangle \in [\mathcal{G}]$ for $0 \leq i, j \leq n$.
- (iii) $\langle s_1, \dots, s_n, X * s_i \rangle \in [\mathcal{G}]$ for $X \in \{L, R, I\}$.

PROOF. (i) By lemma 5.2.22(ii) one has $F_1 \equiv L^{(n-1)} \in [\mathcal{G}]$ and $F_i \equiv R * L^{(n-i)} \in [\mathcal{G}]$. Hence

$$\begin{aligned} s_1 &= F_1 * \langle s_1, \dots, s_n \rangle \in [\mathcal{G}]; \\ s_i &= F_i * \langle s_1, \dots, s_n \rangle \in [\mathcal{G}], \quad \text{for } i = 2, \dots, n. \end{aligned}$$

(ii) By lemma 5.2.22(i) one has $\langle I, \langle F_i, F_j \rangle \rangle = \langle I, F_i, F_j \rangle \in [\mathcal{G}]$. Hence

$$\langle s_1, \dots, s_n, \langle s_i, s_j \rangle \rangle = \langle I, \langle F_i, F_j \rangle \rangle * \langle s_1, \dots, s_n \rangle \in [\mathcal{G}].$$

(iii) Similarly $\langle s_1, \dots, s_n, X * s_i \rangle = \langle I, X * F_i \rangle * \langle s_1, \dots, s_n \rangle \in [\mathcal{G}]$. ■

5.2.24. THEOREM. As a monoid, \mathcal{F} is finitely generated. In fact $\mathcal{F} = [\mathcal{G}]$.

PROOF. Let $e \in \mathcal{F}$. Then there is a formation sequence $e_1 \equiv L, e_2 \equiv R, e_3 \equiv I, \dots, e_n \equiv e$ such that for each $4 \leq k \leq n$ there are $i, j < k$ such that $e_k \equiv \langle e_i, e_j \rangle$ or $e_k \equiv X * e_i$, with $X \in \{L, R, I\}$.

We have $I = \langle L, R \rangle = \langle\langle L, R \rangle \in [\mathcal{G}]$. By lemma 5.2.23(ii), (iii) it follows that

$$\langle\langle e_1, e_2, e_3, \dots, e_n \rangle \in [\mathcal{G}].$$

Therefore by (i) of that lemma $e \equiv e_n \in [\mathcal{G}]$.

The following corollary is similar to a result of Böhm, who showed that the monoid of untyped lambda terms has two generators, see Barendregt [1984].

5.2.25. COROLLARY. (i) *Let \mathcal{M} be a finitely generated cartesian monoid. Then \mathcal{M} is generated by two of its elements.*

(ii) *$\mathcal{F}[x_1, \dots, x_n]$ is generated by two elements.*

PROOF. (i) Let $\mathcal{G} = \{g_1, \dots, g_n\}$ be the set of generators of \mathcal{M} . Then \mathcal{G} and hence \mathcal{M} is generated by R and $\langle\langle g_1, \dots, g_n, L \rangle$.

(ii) $\mathcal{F}[\vec{x}]$ is generated by \mathcal{G} and the \vec{x} , hence by (i) by two elements. ■

Invertibility in \mathcal{F}

5.2.26. DEFINITION. (i) Let $\mathcal{L}(\mathcal{R})$ be the submonoid of the right (left) invertible elements of \mathcal{F}

$$\begin{aligned} \mathcal{L} &= \{a \in \mathcal{F} \mid \exists b \in \mathcal{F} \, b * a = I\}; \\ \mathcal{R} &= \{a \in \mathcal{F} \mid \exists b \in \mathcal{F} \, a * b = I\}. \end{aligned}$$

(ii) Let \mathcal{I} be the subgroup of \mathcal{F} consisting of invertible elements

$$\mathcal{I} = \{a \in \mathcal{F} \mid \exists b \in \mathcal{F} \, a * b = b * a = I\}.$$


It is easy to see that $\mathcal{I} = \mathcal{L} \cap \mathcal{R}$.

5.2.27. EXAMPLES. (i) $L, R \in \mathcal{L}$, since both have the right inverse $\langle I, I \rangle$.

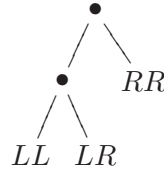
(ii) $a = \langle\langle L, R \rangle, L \rangle$ having as ‘tree’



has as left inverse $b = \langle R, RL \rangle$, where we do not write the $*$ in strings.

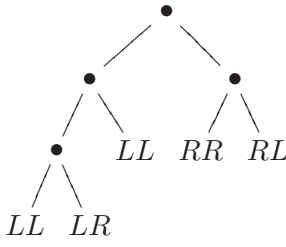
- (iii)  has no left inverse, since “ R cannot be obtained”.

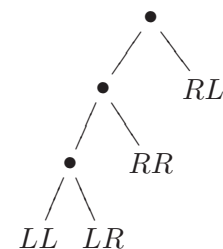
- (iv) $a = \langle \langle RL, LL \rangle, RR \rangle$ having the following tree



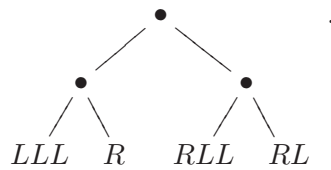
has the following right inverses $b = \langle \langle \langle RL, LL \rangle, \langle c, R \rangle \rangle, R \rangle$. Indeed

$$a * b = \langle \langle RLb, LLb \rangle, RRb \rangle = \langle \langle LL, RL \rangle, R \rangle = \langle L, R \rangle = I.$$

- (v)  has no right inverse, as “ LL occurs twice”.

- (vi)  has a two-sided inverse, as “all strings of two letters” occur

exactly once, the inverse being



For normal forms $f \in \mathcal{F}$ we have the following characterizations.

5.2.28. PROPOSITION. (i) f has a right inverse if and only if f can be expanded (by replacing x by $\langle Lx, Rx \rangle$) so that all of its strings at the leaves have the same length and none occurs more than once.

(ii) f has a left inverse if and only if f can be expanded so that all of its strings at the leaves have the same length, say n , and each of the possible 2^n strings of this length actually occurs.

(iii) f is doubly invertible if and only if f can be expanded so that all of its strings at the leaves have the same length, say n , and each of the possible 2^n strings of this length occurs exactly once.

PROOF. This is clear from the examples. ■

The following terms are instrumental to generate \mathcal{I} and \mathcal{R} .

5.2.29. DEFINITION.

$$\begin{aligned} B_n &= \langle LR^0, \dots, LR^{n-1}, LLR^n, RLR^n, RR^n \rangle; \\ C_0 &= \langle R, L \rangle \\ C_{n+1} &= \langle LR^0, \dots, LR^{n-1}, LRR^n, LR^n, RRR^n \rangle. \end{aligned}$$

5.2.30. PROPOSITION. (i) $\mathcal{I} = [\{B_n \mid n \in \mathbb{N}\} \cup \{C_n \mid n \in \mathbb{N}\}]$.

(ii) $\mathcal{R} = L * \mathcal{I} = R * \mathcal{I}$.

5.2.31. REMARK. The B_n alone generate the so-called Thompson-Freyd-Heller group, see exercise 5.6.25.

5.2.32. PROPOSITION. If $f(\vec{x})$ and $g(\vec{x})$ are distinct members of $\mathcal{F}[\vec{x}]$, then there exists $\vec{h} \in \mathcal{F}$ such that $f(\vec{h}) \neq g(\vec{h})$. We say that $\mathcal{F}[\vec{x}]$ is separable.

PROOF. Suppose that $f(\vec{x})$ and $g(\vec{x})$ are distinct normal members of $\mathcal{F}[\vec{x}]$. We shall find \vec{h} such that $f(\vec{h}) \neq g(\vec{h})$. First remove subexpressions of the form $L * x_i * h$ and $R * x_j * h$ by substituting $\langle y, z \rangle$ for x_i, x_j and renormalizing. This process terminates, and is invertible by substituting $L * x_i$ for y and $R * x_j$ for z . Thus we can assume that $f(\vec{x})$ and $g(\vec{x})$ are distinct normal and without subexpressions of the two forms above. Indeed, expressions like this can be recursively generated as a string of x_i 's followed by a string of L 's and R 's, or as a string of x_i 's followed by a single $\langle \rangle$ of expressions of the same form. Let m be a large number relative to $f(\vec{x}), g(\vec{x})$ ($> \#f(\vec{x}), \#g(\vec{x})$, where $\#t$ is the number of symbols in t .) For each positive integer i , set

$$h_i = \langle \langle R^m, \dots, R^m, I \rangle, R^m \rangle$$

where the right-associative $\langle \rangle$ -expression contains i times R^m . We claim that both $f(\vec{x})$ and $g(\vec{x})$ can be reconstructed from the normal forms of $f(\vec{h})$ and $g(\vec{h})$, so that $f(\vec{h}) \neq g(\vec{h})$.

Define $d_r(t)$, for a normal $t \in \mathcal{F}$, as follows.

$$\begin{aligned} d_r(\vec{w}) &= 0 && \text{if } \vec{w} \text{ is a string of } L, R\text{'s;} \\ d_r(\langle t, s \rangle) &= d_r(s). \end{aligned}$$

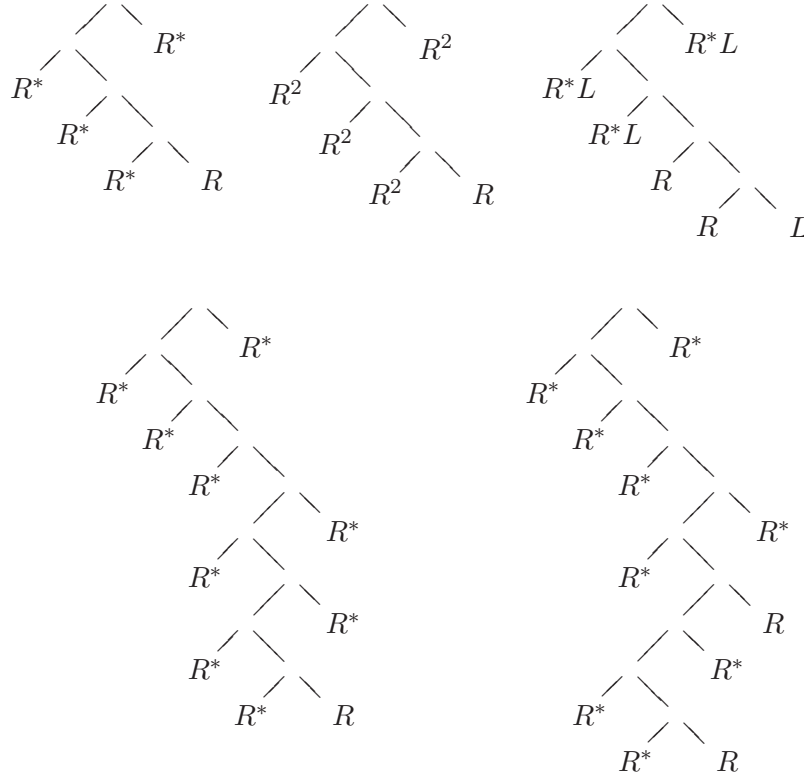
Note that if t is a normal member of \mathcal{F} and $d_r(t) < m$, then

$$h_i * t =_{\text{CM}} \langle \langle t', \dots, t', t \rangle, t' \rangle,$$

where $t' \equiv R^m t$ is $\langle \rangle$ -free. Also note that if s is the CM-nf of $h_i * t$, then $d_r(s) = 1$. The normal form of, say, $f(\vec{h})$ can be computed recursively bottom up as in the computation of the normal form of $h_i * t$ above. In order to compute back $f(\vec{x})$ we consider several examples.

$$\begin{aligned} f_1(\vec{x}) &= x_3 R; \\ f_2(\vec{x}) &= \langle \langle R^2, R^2, R^2, R \rangle, R^2 \rangle; \\ f_3(\vec{x}) &= \langle R, R, L \rangle \\ f_4(\vec{x}) &= x_3 x_1 x_2 R; \\ f_5(\vec{x}) &= x_3 x_1 \langle x_2 R, R \rangle. \end{aligned}$$

Then $f_1(\vec{h}), \dots, f_5(\vec{h})$ have as trees respectively



In these trees the R^* denote long sequences of R 's of possibly different lengths. ■

Cartesian monoids inside λ_{SP}

Remember $\mathcal{C}^0 = \langle \Lambda_{SP}^\emptyset(1) / =_{\beta\eta SP}, \circ, \mathsf{l}, \mathsf{L}, \mathsf{R}, \langle \cdot, \cdot \rangle \rangle$.

5.2.33. PROPOSITION. *There is a surjective homomorphism $h : \mathcal{F} \rightarrow \mathcal{C}^0$.*

PROOF. Now if $M : 1$ is a closed term and in long $\beta\eta SP$ normal form, then M has one of the following shapes: $\lambda a.a$, $\lambda a.\pi X_1 X_2$, $\lambda a.\pi_i X$ for $i = 1$ or $i = 2$. Then we have $M \equiv I$, $M = \langle \lambda a.X_1, \lambda a.X_2 \rangle$, $M = L \circ (\lambda a.X)$ or $M = R \circ (\lambda a.X)$, respectively. Since the terms $\lambda a.X_i$ are smaller than M , this yields an inductive definition of the set of closed terms of λ_{SP} modulo \equiv in terms of the combinators $I, L, R, \langle \cdot, \cdot \rangle, \circ$. Thus the elements of \mathcal{C}^0 are generated from $\{I, \circ, L, R, \langle \cdot, \cdot \rangle\}$ in an algebraic way. Now define

$$\begin{aligned} h(I) &= I; \\ h(L) &= L; \\ h(R) &= R; \\ h(\langle a, b \rangle) &= \langle h(a), h(b) \rangle; \\ h(a * b) &= h(a) \circ h(b). \end{aligned}$$

Then h is a surjective homomorphism. ■

Now we will show in two different ways that this homomorphism is in fact injective and hence an isomorphism.

5.2.34. THEOREM. $\mathcal{F} \cong \mathcal{C}^0$.

PROOF 1. We will show that the homomorphism h in proposition 5.2.33 is injective. By a careful examination of CM -normal forms one can see the following. Each expression can be rewritten uniquely as a binary tree whose nodes correspond to applications of $\langle \cdot, \cdot \rangle$ with strings of L 's and R 's joined by $*$ at its leaves (here I counts as the empty string) and no subexpressions of the form $\langle L * e, R * e \rangle$. Thus

$$a \neq b \Rightarrow a^{\text{nf}} \neq b^{\text{nf}} \Rightarrow h(a^{\text{nf}}) \neq h(b^{\text{nf}}) \Rightarrow h(a) \neq h(b),$$

so h is injective. ■₁

PROOF 2. By proposition 5.2.19. ■₂

The structure \mathcal{C}^0 will be generalized as follows.

5.2.35. DEFINITION. Consider the type $1^n \rightarrow 1 = (o \rightarrow o)^n \rightarrow o \rightarrow o$. Define

$$\mathcal{C}^n = \langle \Lambda_{SP}^\emptyset(1^n \rightarrow 1) / \equiv_{\beta\eta SP}, I_n, L_n, R_n, \circ_n, \langle -, - \rangle_n \rangle,$$

where writing $\vec{x} = x_1, \dots, x_n$

$$\begin{aligned} \langle M, N \rangle_n &= \lambda \vec{x}. \langle M \vec{x}, N \vec{x} \rangle; \\ M \circ_n N &= \lambda \vec{x}. (M \vec{x}) \circ (N \vec{x}); \\ I_n &= \lambda \vec{x}. I; \\ L_n &= \lambda \vec{x}. L; \\ R_n &= \lambda \vec{x}. R. \end{aligned}$$

5.2.36. PROPOSITION. \mathcal{C}^n is a non-trivial Cartesian monoid.

PROOF. Easy. ■

5.2.37. PROPOSITION. $\mathcal{C}^n \cong \mathcal{F}[x_1, \dots, x_n]$.

PROOF. As before, let $h_n : \mathcal{F}[\vec{x}] \rightarrow \mathcal{C}^n$ be induced by

$$\begin{aligned} h_n(x_i) &= \lambda \vec{x} \lambda z \ o.x_i z &= \lambda \vec{x}.x_i; \\ h_n(I) &= \lambda \vec{x} \lambda z \ o.z &= I_n; \\ h_n(L) &= \lambda \vec{x} \lambda z \ o.\pi_1 z &= L_n; \\ h_n(R) &= \lambda \vec{x} \lambda z \ o.\pi_2 z &= R_n; \\ h_n(\langle s, t \rangle) &= \lambda \vec{x} \lambda z \ o.\pi(s\vec{x}z)(t\vec{x}z) &= \langle h_n(s), h_n(t) \rangle_n. \end{aligned}$$

As before one can show that this is an isomorphism. ■

In the sequel an important case is $n = 1$, i.e. $\mathcal{C}^{1 \rightarrow 1} \cong \mathcal{F}[x]$.

Hilbert-Post completeness of $\lambda_{\rightarrow} SP$

The claim that an equation $M = N$ is either a $\beta\eta SP$ convertibility or inconsistent is proved in two steps. First it is proved for the type $1 \rightarrow 1$ by the analysis of $\mathcal{F}[x]$; then it follows for arbitrary types by reducibility of types in λ_{SP} .

Remember that $M \#_T N$ means that $T \cup \{M = N\}$ is inconsistent.

5.2.38. PROPOSITION. (i) Let $M, N \in \Lambda_{SP}^\emptyset(1)$. Then

$$M \neq_{\beta\eta SP} N \Rightarrow M \#_{\beta\eta SP} N.$$

(ii) The same holds for $M, N \in \Lambda_{SP}^\emptyset(1 \rightarrow 1)$.

PROOF. (i) Since $\mathcal{F} \cong \mathcal{C}^0 = \Lambda_{SP}^\emptyset(1)$, by theorem 5.2.34, this follows from proposition 5.2.19(i).

(ii) If $M, N \in \Lambda_{SP}^\emptyset(1 \rightarrow 1)$, then

$$\begin{aligned} M \neq N &\Rightarrow M = \lambda f \ 1.M_1[f] \neq \lambda f \ 1.N_1[f] = N && \text{with } M_1[f], N_1[f] \\ &\Rightarrow M_1[f] \neq N_1[f] \\ &\Rightarrow M_1[F] \neq N_1[F] && \text{for some } F \in \Lambda_{SP}^\emptyset(1), \\ &&& \text{by 5.2.32,} \\ &\Rightarrow M_1[F] \# N_1[F] \\ &\Rightarrow M \# N. \blacksquare \end{aligned}$$

We now want to generalize this last result for all types by using type reducibility in the context of λ_{SP} .

5.2.39. DEFINITION. Let $A, B \in \mathbb{T}(\lambda_{\rightarrow})$. We say that A is $\beta\eta SP$ -reducible to B , notation $A \leq_{\beta\eta SP} B$, iff there exists $\Phi : A \rightarrow B$ such that for any closed $N_1, N_2 : A$

$$N_1 = N_2 \iff \Phi N_1 = \Phi N_2.$$

5.2.40. PROPOSITION. For each type A one has $A \leq_{\beta\eta SP} 1 \rightarrow 1$.

PROOF. We can copy the proof 3.4.7 to obtain $A \leq_{\beta\eta SP} 1_2 \rightarrow o \rightarrow o$. Moreover, by

$$\lambda u x a. u(\lambda z_1 z_2. x(\pi(x z_1)(x z_2)))a$$

one has $1_2 \rightarrow o \rightarrow o \leq_{\beta\eta SP} 1 \rightarrow 1$. ■

5.2.41. COROLLARY. Let $A \in \mathbb{T}(\lambda_{\rightarrow})$ and $M, N \in \Lambda_{SP}^\emptyset$. Then

$$M \neq_{\beta\eta SP} N \Rightarrow M \#_{\beta\eta SP} N.$$

PROOF. Let $A \leq_{\beta\eta SP} 1 \rightarrow 1$ using Φ . Then

$$\begin{aligned} M \neq N &\Rightarrow \Phi M \neq \Phi N \\ &\Rightarrow \Phi M \# \Phi N, \text{ by corollary 5.2.38(ii),} \\ &\Rightarrow M \# N. \blacksquare \end{aligned}$$

We obtain the following Hilbert-Post completeness theorem.

5.2.42. THEOREM. Let \mathcal{M} be a model of λ_{SP} . For any type A and closed terms $M, N \in \Lambda^\emptyset(A)$ the following are equivalent.

- (i) $M =_{\beta\eta SP} N$;
- (ii) $\mathcal{M} \models M = N$;
- (iii) $\lambda_{SP} \cup \{M = N\}$ is consistent.

PROOF. ((i) \Rightarrow (ii)) By soundness. ((ii) \Rightarrow (iii)) Since truth implies consistency. ((iii) \Rightarrow (i)) By corollary 5.2.41. ■

The result also holds for equations between open terms (consider their closures). The moral is that every equation is either provable or inconsistent. Or that every model of λ_{SP} has the same (equational) theory.

Diophantine relations

5.2.43. DEFINITION. Let $R \subseteq \Lambda^\emptyset(A_1) \times \dots \times \Lambda^\emptyset(A_n)$ be an n -ary relation.

(i) R is called *equational* iff

$$\exists B \in \Pi(o) \exists M, N \in \Lambda^\emptyset(A_1 \rightarrow \dots \rightarrow A_n \rightarrow B) \forall \vec{F}$$

$$R(F_1, \dots, F_n) \iff MF_1 \dots F_n = NF_1 \dots F_n. \quad (1)$$

Here $=$ is taken in the sense of the theory of λ_{SP} .

(ii) R is called the *projection* of the $n + m$ -ary relation S if

$$R(\vec{F}) \iff \exists \vec{G} S(\vec{F}, \vec{G})$$

(iii) R is called *Diophantine* if it is the projection of an equational relation.

Note that syntactic relations are closed coordinate wise under $=$ and are recursive (since λ_{SP} is CR and SN). A Diophantine relation is clearly closed under $=$ (coordinate wise) and recursively enumerable. Our main result will be the converse.

5.2.44. THEOREM. *A relation R on closed Λ_{SP} terms is Diophantine if and only if R is closed coordinate wise under $=$ and recursively enumerable.*

The rest of this section is devoted to the proof of this claim.

5.2.45. PROPOSITION. (i) *Equational relations are closed under substitution of lambda definable functions. This means that if R is equational and R' is defined by*

$$R'(\vec{F}) \iff R(H_1\vec{F}, \dots, H_n\vec{F}),$$

then R' is equational.

(ii) *Equational relations are closed under conjunction.*

(iii) *Equational relations are Diophantine.*

(iv) *Diophantine relations are closed under substitution of lambda definable functions, conjunction and projection.*

PROOF. (i) Easy.

(ii) Use (simple) pairing.

(iii) By dummy projections.

(iv) By some easy logical manipulations. ■

5.2.46. LEMMA. *Let $R \subseteq \Pi_{i=1}^n \Lambda^\emptyset(A_i)$. Suppose that $\Phi_i : A_i \leq_{\beta\eta SP} 1 \rightarrow 1$. Define $R^\Phi \subseteq \Lambda^\emptyset(1 \rightarrow 1)^n$ by*

$$R^\Phi(G_1, \dots, G_n) \iff \exists F_1 \dots F_n [\Phi_1 F_1 = G_1 \& \dots \& \Phi_n F_n = G_n \& R(F_1, \dots, F_n)].$$

Then

(i) *R is Diophantine iff R^Φ is Diophantine.*

(ii) R is $=$ -closed and re iff R^Φ is $=$ -closed and re.

PROOF. (i) (\Rightarrow) By proposition 5.2.45.

(\Leftarrow) By noting that $R(F_1, \dots, F_n) \iff R^\Phi(\Phi F_1, \dots, \Phi F_n)$.

(ii) Similarly. ■

By pairing we can assume without loss of generality that $n = 1$.

5.2.47. LEMMA. Let $R \subseteq \Lambda^\emptyset(1 \rightarrow 1)^n$. Define $R^\wedge \subseteq \Lambda^\emptyset(1 \rightarrow 1)$ by

$$R^\wedge(F) \iff R(\pi_1^{1 \rightarrow 1}(F), \dots, \pi_n^{1 \rightarrow 1}(F)).$$

Then

(i) R is Diophantine iff R^\wedge is Diophantine.

(ii) R is $=$ -closed and re iff R^\wedge is $=$ -closed and re.

PROOF. By proposition 5.2.45(i) and the pairing functions. ■

5.2.48. COROLLARY. In order to prove that every RE relation $R \subseteq \Pi_{i=1}^n \Lambda^\emptyset(A_i)$ that is closed under $=_{\beta\eta SP}$ is Diophantine, it suffices to do this just for such $R \subseteq \Lambda^\emptyset(1 \rightarrow 1)$.

PROOF. By the previous two lemmas. ■

So now we are interested in recursively enumerable subsets of $\Lambda^\emptyset(1 \rightarrow 1)$ closed under $=_{\beta\eta SP}$. Since

$$\exp(\mathbf{CM} \cup \{\mathbf{x}\}) / =_{\mathbf{CM}} = \mathcal{F}[x] \cong \mathcal{C}^1 = \Lambda^\emptyset(1 \rightarrow 1) / =_{\beta\eta SP}$$

one can shift attention to relations on $\exp(\mathbf{CM} \cup \{\mathbf{x}\})$ closed under $=_{\mathbf{CM}}$. We say loosely that such relations are on $\mathcal{F}[x]$. The definition of Such relations to be equational (Diophantine) is slightly different (but completely in accordance with the isomorphism $\mathcal{C}^1 \cong \mathcal{F}[x]$).

5.2.49. DEFINITION. A k -ary relation R on $\mathcal{F}[\vec{x}]$ is called *Diophantine* if there exist $s(u_1, \dots, u_k, \vec{v}), t(u_1, \dots, u_k, \vec{v}) \in \mathcal{F}[\vec{u}, \vec{v}]$ such that

$$R(f_1[\vec{x}], \dots, f_k[\vec{x}]) \iff \exists \vec{v} \in \mathcal{F}[\vec{x}]. s(f_1[\vec{x}], \dots, f_k[\vec{x}], \vec{v}) = t(f_1[\vec{x}], \dots, f_k[\vec{x}], \vec{v}).$$

Diophantine relations on \mathcal{F} are closed under conjunction as before.

5.2.50. PROPOSITION (Transfer lemma). (i) Let $X \subseteq (\mathcal{F}[x_1, \dots, x_n])^k$ be equational (Diophantine). Then $h_n(X) \subseteq (\mathcal{C}^n)^k$ is equational (Diophantine), respectively.

(ii) Let $X \subseteq (\mathcal{C}^n)^k$ be RE and closed under $=_{\beta\eta SP}$. Then $h_n^{-1}(X) \subseteq (\mathcal{F}[x_1, \dots, x_n])^k$ is RE and closed under $=_{\mathbf{CM}}$.

5.2.51. COROLLARY. In order to prove that every RE relation on \mathcal{C}^1 closed under $=_{\beta\eta SP}$ is Diophantine it suffices to show that every RE relation on $\mathcal{F}[x]$ closed under $=_{\mathbf{CM}}$ is Diophantine.

Before proving that every recursively enumerable relation on $\mathcal{F}[x]$ is Diophantine, for the sake of clarity we shall give the proof first for \mathcal{F} . It consists of two steps: first we encode Matijasevic's solution to Hilbert's 10th problem into this setting; then we give a Diophantine coding of \mathcal{F} in \mathcal{F} , and finish the proof for \mathcal{F} . Since the coding of \mathcal{F} can easily be extended to $\mathcal{F}[x]$ the result then holds also for this structure and we are done.

5.2.52. DEFINITION. Write $s_n = R^n \in \mathcal{F}$. The set of *numerals* in \mathcal{F} is defined by $\mathcal{N} = \{s_n \mid n \in \mathbb{N}\}$.

We have the following.

5.2.53. PROPOSITION. $f \in \mathcal{N} \iff f * R = R * f$.

PROOF. This is because if f is normal and $f * R = R * f$, then f the binary tree part of f must be trivial, that is, f must be a string of L 's and R 's, thus only R 's. ■

5.2.54. DEFINITION. A sequence of k -ary relations on $R_n \subseteq \mathcal{F}$ is called *Diophantine uniformly in n* iff there is a $k + 1$ -ary Diophantine relation $P \subseteq \mathcal{F}^{k+1}$ such that

$$R_n(\vec{u}) \iff P(s_n, \vec{u}).$$

Now we build up a toolkit of Diophantine relations on \mathcal{F} .

1. \mathcal{N} is equational (hence Diophantine).

PROOF. In 5.2.53 is was proved that

$$f \in \mathcal{N} \iff f * R = R * f. \blacksquare$$

2. The sets $\mathcal{F} * L$, $\mathcal{F} * R \subseteq \mathcal{F}$ and $\{L, R\}$ are equational. In fact one has

$$\begin{aligned} f \in \mathcal{F} * L &\iff f * \langle L, L \rangle = f. \\ f \in \mathcal{F} * R &\iff f * \langle R, R \rangle = f. \\ f \in \{L, R\} &\iff f * \langle I, I \rangle = f. \end{aligned}$$

PROOF. To see that the first equivalence holds, notice that if $f \in \mathcal{F} * L$, then $f = g * L$, for some $g \in \mathcal{F}$ hence $f * \langle L, L \rangle = f$. Conversely, if $f = f * \langle L, L \rangle$, then $f = f * \langle l, l \rangle * L \in \mathcal{F} * L$.

The second equivalence is proved similarly.

In the third equivalence (\Leftarrow) follows by induction on the nf of f . ■

3. Notation
$$\begin{aligned} [] &= R; \\ [f_0, \dots, f_{n-1}] &= \langle f_0 * L, \dots, f_{n-1} * L, R \rangle, \quad \text{if } n > 0. \end{aligned}$$

One easily sees that $[f_0, \dots, f_{n-1}] * [I, f_n] = [f_0, \dots, f_n]$. Write

$$\text{Aux}_n(f) = [f, f * R, \dots, f * R^{n-1}].$$

Then the relation $h = \text{Aux}_n(f)$ is Diophantine uniformly in n .

PROOF. Indeed,

$$h = \text{Aux}_n(f) \iff R^n * h = R \ \& \ h = R * h * \langle \langle L, L \rangle, \langle f * R^{n-1} * L, R \rangle \rangle.$$

To see (\Rightarrow) , assume $h = [f, f * R, \dots, f * R^{n-1}]$, then

$h = \langle f * L, f * R * L, \dots, f * R^{n-1} * L, R \rangle$, so $R^n * h = R$ and

$$\begin{aligned} R * h &= [f * R, \dots, f * R^{n-1}] \\ R * h * \langle \langle L, L \rangle, \langle f * R^{n-1} * L, R \rangle \rangle &= [f, f * R, \dots, f * R^{n-1}] \\ &= h. \end{aligned}$$

To see (\Leftarrow) , note that we always can write $h = \langle h_0, \dots, h_n \rangle$. By the assumptions $h = R * h * \langle \langle L, L \rangle, \langle f * R^{n-1} * L, R \rangle \rangle = R * h * \text{---}$, say. So by reading the following equality signs in the correct order (first the left $=$'s top to bottom; then the right $=$'s bottom to top) it follows that

$$\begin{aligned} h_0 &= h_1 * \text{---} = f * L \\ h_1 &= h_2 * \text{---} = f * R * L \\ &\dots \\ h_{n-2} &= h_{n-1} * \text{---} = f * R^{n-2} * L \\ h_{n-1} &= f * R^{n-1} * L \\ h_n &= R. \end{aligned}$$

Therefore $h = \text{Aux}_n(f)$ ■.

4. Write $\text{Seq}_n(f) \iff f = [f_0, \dots, f_{n-1}]$, for some f_0, \dots, f_{n-1} . Then Seq_n is Diophantine uniformly in n .

PROOF. One has $\text{Seq}_n(f)$ iff

$$R^n * f = R \ \& \ \text{Aux}_n(L) * \langle I, L \rangle * f = \text{Aux}_n(L) * \langle I, L \rangle * f * \langle L, L \rangle,$$

as can be proved similarly (use 2(i)). ■

5. Define

$$\text{Cp}_n(f) = [f, \dots, f], \text{ (} n \text{ times } f \text{)}.$$

(By default $\text{Cp}_0(f) = [] = R$.) Then Cp_n is Diophantine uniformly in n . Then Cp_n is Diophantine uniformly in n .

PROOF. $\text{Cp}_n(f) = g$ iff

$$\text{Seq}_n(g) \ \& \ g = R * g * \langle L, f * L, R \rangle. \quad \blacksquare$$

6. Let $\text{Pow}_n(f) = f^n$. Then Pow_n is Diophantine uniformly in n .

PROOF. One has $\text{Pow}_n(f) = g$ iff

$$\exists h [\text{Seq}_n(h) \ \& \ h = R * h * \langle f * L, f * L, R \rangle \ \& \ L * h = g].$$

This can be proved in a similar way (it helps to realize that h has to be of the form $h = [f^n, \dots, f^1]$). ■

Now we can show that the operations $+$ and \times on \mathcal{N} are ‘Diophantine’.

7. There are Diophantine ternary relations P_+, P_\times such that for all n, m, k

$$(1) P_+(s_n, s_m, s_k) \iff n + m = k.$$

$$(2) P_\times(s_n, s_m, s_k) \iff n \cdot m = k.$$

PROOF. (i) Define $P_+(x, y, z) \iff x * y = z$. This relation is Diophantine and works: $R^n * R^m = R^k \iff R^{n+m} = R^k \iff n + m = k$.

(ii) Let $\text{Pow}_n(f) = g \iff P(s_n, f, g)$, with P Diophantine. Then $P_\times(x, y, z) \iff P(x, y, z)$. ■

8. Let $X \subseteq \mathbb{N}$ be a recursively enumerable set of natural numbers. Then $\{s_n \mid n \in X\}$ is Diophantine.

PROOF. By 7 and the famous theorem of Matijasevič [1971]. ■

9. Define $\text{Seq}_n^{\mathcal{N}} = \{[s_{m_0}, \dots, s_{m_{n-1}}] \mid m_0, \dots, m_{n-1} \in \mathbb{N}\}$. Then the relation $f \in \text{Seq}_n^{\mathcal{N}}$ is Diophantine uniformly in n .

PROOF. Indeed, $f \in \text{Seq}_n^{\mathcal{N}}$ iff

$$\text{Seq}_n(f) \ \& \ f * \langle R * L, R \rangle = \text{Aux}_n(R * L) * \langle I, R^n \rangle * f. \quad \blacksquare$$

10. Let $f = [f_0, \dots, f_{n-1}]$ and $g = [g_0, \dots, g_{n-1}]$. We write

$$f \# g = [f_0 * g_0, \dots, f_{n-1} * g_{n-1}].$$

Then there exists a Diophantine relation P such that for arbitrary n and $f, g \in \text{Seq}_n$ one has

$$P(f, g, h) \iff h = f \# g.$$

PROOF. Let

$$\text{Cmp}_n(f) = [L * f, L * R * f * R, \dots, L * R^{n-1} * f * R^{n-1}].$$

Then $g = \text{Cmp}_n(f)$ is Diophantine uniformly in n .

This requires some work. One has by the by now familiar technique

$$\begin{aligned} \text{Cmp}_n(f) = g \iff & \\ \exists h_1, h_2, h_3 \quad [& \\ \text{Seq}_n(h_1) \quad \& \quad & f = h_1 * \langle I, R^n * f \rangle \\ \text{Seq}_{n^2}(h_2) \quad \& \quad & h_2 = R^n * h_2 * \langle \langle L, L \rangle, h_1 * \langle R^{n-1} * L, R \rangle \rangle \\ \text{Seq}_n^{\mathcal{N}}(h_3) \quad \& \quad & h_3 = R * h_3 * \langle \langle I, I \rangle^{n+1} * L, \langle R^{n^2-1} * L, R \rangle \rangle \\ & \& \quad & g = \text{Aux}_n(L^2) * \langle h_3, R^n \rangle * \langle h_2, R \rangle \\ &] . \end{aligned}$$

For understanding it helps to identify the h_1, h_2, h_3 . Suppose $f = \langle f_0, \dots, f_{n-1}, f_n \rangle$. Then

$$\begin{aligned} h_1 &= [f_0, f_1, \dots, f_{n-1}]; \\ h_2 &= [f_0, f_1, \dots, f_{n-1}, \\ &\quad f_0 * R, f_1 * R, \dots, f_{n-1} * R, \\ &\quad \dots, \\ &\quad f_0 * R^{n-1}, f_1 * R^{n-1}, \dots, f_{n-1} * R^{n-1}]; \\ h_3 &= [I, R^{n+1}, R^{2(n+1)}, \dots, R^{(n-1)(n+1)}]. \end{aligned}$$

Now define

$$P(f, g, h) \iff \exists n [\text{Seq}_n(f) \ \& \ \text{Seq}_n(g) \ \& \ \text{Cmp}_n(f * L) * \langle I, R^n \rangle * g = h].$$

Then P is Diophantine and for arbitrary n and $f, g \in \text{Seq}_n$ one has

$$h = f \# g \iff P(f, g, h). \blacksquare$$

11. For $f = [f_0, \dots, f_{n-1}]$ define $\Pi(f) = f_0 * \dots * f_{n-1}$. Then there exists a Diophantine relation P such that for all $n \in \mathbb{N}$ and all $f \in \text{Seq}_n$ one has

$$P(f, g) \iff \Pi(f) = g.$$

PROOF. Define $P(f, g)$ iff

$$\begin{aligned} &\exists n, h \ [\\ &\quad \text{Seq}_n(f) \ \& \\ &\quad \text{Seq}_{n+1}(h) \ \& \ h = ((f * \langle I, R \rangle) \# (R * h)) * \langle L, I * L, R \rangle \\ &\quad \& \ g = L * h * \langle I, R \rangle \\ &\quad]. \end{aligned}$$

Then P works as can be seen realizing h has to be

$$[f_0 * \dots * f_{n-1}, f_1 * \dots * f_{n-1}, \dots, f_{n-2} * f_{n-1}, f_{n-1}, I]. \blacksquare$$

12. Define $\text{Byte}_n(f) \iff f = [b_0, \dots, b_{n-1}]$, for some $b_i \in \{L, R\}$. Then Byte_n is Diophantine uniformly in n .

PROOF. Using 2 one has $\text{Byte}_n(f)$ iff

$$\text{Seq}_n(f) \ \& \ f * \langle \langle I, I \rangle, R \rangle = \text{Cp}_n(I). \blacksquare$$

13. Let $m \in \mathbb{N}$ and let $[m]_2$ be its binary notation of length n . Let $[m]_{\text{Byte}} \in \text{Seq}_n^{\mathcal{N}}$ be the corresponding element, where L corresponds to a 1 and R to a 0 and the most

significant bit is written last. For example $[6]_2 = 110$, hence $[6]_{\text{Byte}} = [R, L, L]$. Then there exists a Diophantine relation Bin such that for all $m \in \mathbb{N}$

$$\text{Bin}(s_m, f) \iff f = [m]_{\text{Byte}}.$$

PROOF. We need two auxiliary maps.

$$\begin{aligned} \text{Pow2}(n) &= [R^{2^{n-1}}, \dots, R^{2^0}]; \\ \text{Pow2}I(n) &= [\langle R^{2^{n-1}}, I \rangle, \dots, \langle R^{2^0}, I \rangle]. \end{aligned}$$

For these the relations $\text{Pow2}(n) = g$ and $\text{Pow2}I(n) = g$ are Diophantine uniformly in n . Indeed, $\text{Pow2}(n) = g$ iff

$$\text{Seq}_n(g) \ \& \ g = ((R * g) \# (R * g)) * [I, R];$$

and $\text{Pow2}I(n) = g$ iff

$$\begin{aligned} \text{Seq}_n(g) \ \& \ \text{Cp}_n(L) \# g = \text{Pow2}(n); \\ \& \ \text{Cp}_n(R) \# g = \text{Cp}_n(I). \end{aligned}$$

It follows that Bin is Diophantine since $\text{Bin}(m, f)$ iff

$$m \in \mathcal{N} \ \& \ \exists n [\text{Byte}_n(f) \ \& \ \Pi(f \# \text{Pow2}I(n)) = m]. \blacksquare$$

14. We now define a surjection $\varphi : \mathbb{N} \rightarrow \mathcal{F}$. Remember that \mathcal{F} is generated by two elements $\{e_0, e_1\}$ using only $*$. One has $e_1 = L$. Define

$$\varphi(n) = e_{i_0} * \dots * e_{i_{m-1}},$$

where $[n]_2 = i_{m-1} \dots i_0$. We say that n is a *code* of $\varphi(n)$. Since every $f \in \mathcal{F}$ can be written as $L * \langle I, I \rangle * f$ the map φ is surjective indeed.

15. $\text{Code}(n, f)$ defined by $\varphi(n) = f$ is Diophantine uniformly in n .

PROOF. Indeed, $\text{Code}(n, f)$ iff

$$\exists g [\text{Bin}(n, g) \ \& \ \Pi(g * \langle \langle e_0, e_1 \rangle, R \rangle) = f]. \blacksquare$$

16. Every RE subset $\mathcal{X} \subseteq \mathcal{F}$ is Diophantine.

PROOF. Since the word problem for \mathcal{F} is decidable, $\# \mathcal{X} = \{m \mid \exists f \in \mathcal{X} \ \varphi(m) = f\}$ is also RE. By (8), $\# \mathcal{X} \subseteq \mathbb{N}$ is Diophantine. Hence by (15) \mathcal{X} is Diophantine via

$$g \in \mathcal{X} \iff \exists f \ f \in \# \mathcal{X} \ \& \ \text{Code}(f, g). \blacksquare$$

17. Every RE subset $\mathcal{X} \subseteq \mathcal{F}[\vec{x}]$ is Diophantine.

PROOF. Similarly, since also $\mathcal{F}[\vec{x}]$ is generated by two of its elements. We need to know that all the Diophantine relations $\subseteq \mathcal{F}$ are also Diophantine $\subseteq \mathcal{F}[x]$. This follows from exercise 5.6.23 and the fact that such relations are closed under intersection. \blacksquare

PROOF OF 5.2.44. By 17 and corollaries 5.2.48 and 5.2.51. \blacksquare

5.3. Gödel's system \mathcal{T} : higher-order primitive recursion

The set of primitive recursive functions is the smallest set containing zero, successor and projection functions which is closed under composition and the following schema of first-order primitive recursion:

$$\begin{aligned} F(0, \vec{x}) &= G(\vec{x}) \\ F(n+1, \vec{x}) &= H(F(n, \vec{x}), n, \vec{x}) \end{aligned}$$

This schema defines F from G and H by stating that $F(0) = G$ and by expressing $F(n+1)$ in terms of $F(n)$, H and n . The parameters \vec{x} range over the natural numbers.

Thus defined, the primitive recursive functions were conjectured by Skolem to cover all computable functions, early this century. This conjecture was shown to be false in Ackermann [1928], who gave a computable function that is not primitive recursive. A few years later the class of computable functions was shown to be much larger by Church and Turing. Nevertheless the primitive recursive functions include almost all functions that one encounters 'in practice', such as addition, multiplication, exponentiation, and many more.

Besides the existence of computable functions that are not primitive recursive, there is another reason to generalize the above schema, namely the existence of computable objects that are not number theoretic functions. For example, given a number theoretic function F and a number n , compute the maximum that F takes on arguments less than n . Other examples of computations where inputs and/or outputs are functions: compute the function that coincides with F on arguments less than n and zeroes otherwise, compute the n -th iterate of F , and so on. These computations define mappings that are commonly called functionals, to stress that they are more general than number theoretic functions.

Consider the full typestructure over the natural numbers, that is, sets $\mathbb{N}_{\mathbb{N}} = \mathbb{N}$ and $\mathbb{N}_{A \rightarrow B} = \mathbb{N}_A \rightarrow \mathbb{N}_B$, the set of all mappings from \mathbb{N}_A to \mathbb{N}_B . Application of $F \in \mathbb{N}_{A \rightarrow B}$ to $G \in \mathbb{N}_A$ is denoted by FG or $F(G)$. We allow a liberal use of currying, so the following denotations are all identified:

$$FGH \equiv (FG)H \equiv F(G, H) \equiv F(G)H \equiv F(G)(H)$$

Application is left associative, so $F(GH)$ is notably different from the above denotations.

The abovementioned interest in higher-order computations leads to the following schema of higher-order primitive recursion proposed in Gödel [1958]:²

$$\begin{aligned} RMN0 &= M \\ RMN(n+1) &= N(RMNn)n \end{aligned}$$

Here M need not be a natural number, but can have any $A \in \mathbb{T}_o$ as type (see Section 1.1). The corresponding type of N is $A \rightarrow \mathbb{N} \rightarrow A$, where \mathbb{N} is the type of the natural numbers.

²For the purpose of a translation of intuitionistic arithmetic into the quantifier free theory of primitive recursive functionals of finite type, yielding a consistency proof for arithmetic.

We make some further observations with respect to this schema. First, the dependency of F from G and H in the first-order schema is made explicit by defining RMN , which is to be compared to F . Second, the parameters \vec{x} from the first-order schema are left out above since they are no longer necessary: we can have higher-order objects as results of computations. Third, the type of R depends on the type of the result of the computation. In fact we have a family of *recursors* $R_A : A \rightarrow (A \rightarrow \mathbb{N} \rightarrow A) \rightarrow \mathbb{N} \rightarrow A$ for every type A .

5.3.1. EXERCISE. Show that the first-order schema of primitive recursion is subsumed by the higher-order schema, by expressing F in terms of R , G and H .

The set of primitive recursive functionals is the smallest set of functionals containing 0, the successor function and functionals R of all appropriate types, which is closed under explicit λ_{\rightarrow}^o -definition. This definition implies that the primitive recursive functionals include projection functions and are closed under application, composition and the above schema of higher-order primitive recursion.

We shall now exhibit a number of examples of primitive recursive functionals. First, let K, K^* be defined explicitly by $K(x, y) = x$, $K^*(x, y) = y$ for all $x, y \in \mathbb{N}$, that is, the first and the second projection. Obviously, K and K^* are primitive recursive functionals, as they come from λ_{\rightarrow}^o -terms. Now consider $P \equiv R0K^*$. Then we have $P0 = 0$ and $P(n+1) = R0K^*(n+1) = K^*(R0K^*)n = n$ for all $n \in \mathbb{N}$, so that we call P the predecessor function. Now consider $x \dot{-} y \equiv Rx(P * K)y$. Here $P * K$ is the composition of P and K , that is, $(P * K)xy = P(K(x, y)) = P(x)$. We have $x \dot{-} 0 = x$ and $x \dot{-} (y+1) = Rx(P * K)(y+1) = (P * K)(Rx(P * K)y)y = P(Rx(P * K)y) = P(x \dot{-} y)$. Thus we have defined cut-off subtraction $\dot{-}$ as primitive recursive functional.

5.3.2. EXERCISE. Which function is computed if we replace P in $Rx(P * K)y$ by the successor function? Define multiplication, exponentiation and division with remainder as primitive recursive functionals.

In the previous paragraph, we have only used $R_{\mathbb{N}}$ in order to define some functions that are, in fact, already definable with first-order primitive recursion. In this paragraph we are going to use $R_{\mathbb{N} \rightarrow \mathbb{N}}$ as well. Given a functions F, F' and natural numbers x, y , define explicitly the functional G by $G(F, F', x, y) = F'(F(y))$ and abbreviate $G(F)$ by G_F . Now consider RIG_F , where R is actually $R_{\mathbb{N} \rightarrow \mathbb{N}}$ and I is the identity function on the natural numbers. We calculate $RIG_F 0 = I$ and $RIG_F(n+1) = G_F(RIG_F n)n$, which is a function assigning $G(F, RIG_F n, n, m) = RIG_F n(Fm)$ to every natural number m . In other words, $RIG_F n$ is a function which iterates F precisely n times, and we denote this function by F^n .

We finish this paragraph with an example of a computable function A that is not first-order primitive recursive, a result due to Ackermann. The essential difficulty of the function A is the nested recursion in the third clause below.

$$\begin{aligned} A(0, m) &= m + 1 \\ A(n + 1, 0) &= A(n, 1) \\ A(n + 1, m + 1) &= A(n, A(n + 1, m)) \end{aligned}$$

In other words, $A(0)$ is the successor function, and by using the the last two equations we see $A(n+1, m) = A(n)^{m+1}(1)$. Thus we can obtain $A = RSH$, where S is the successor function and $H(F, x, y) = F^{y+1}1$. As examples we calculate $A(1, m) = H(A(0), 1, m) = A(0)^{m+1}1 = m+2$ and $A(2, m) = H(A(1), 1, m) = A(1)^{m+1}1 = 2m+3$.

5.3.3. EXERCISE. Calculate $A(3, m)$ and verify that $A(4, 0) = 13$ and $A(4, 1) = 65533$.

5.3.4. EXERCISE. With one occurrence hidden in H , RSH contains $R_{\mathbb{N} \rightarrow \mathbb{N}}$ twice. Define A using $R_{\mathbb{N}}$ and $R_{\mathbb{N} \rightarrow \mathbb{N}}$ only once. Is it possible to define A with $R_{\mathbb{N}}$ only, possibly with multiple occurrences?

course of value recursion
multiple recursion

5.3.5. EXERCISE (simultaneous primitive recursion). Assume G_i, H_i ($i = 1, 2$) have been given and define F_i ($i = 1, 2$) as follows.

$$\begin{aligned} F_i(0, \vec{x}) &= G_i(\vec{x}) \\ F_i(n+1, \vec{x}) &= H_i(F_1(n, \vec{x}), (F_2(n, \vec{x}), n, \vec{x})) \end{aligned}$$

Show that F_i ($i = 1, 2$) can be defined by first-order primitive recursion. Hint: use a pairing function such as in Figure 5.2.

5.3.6. EXERCISE (nested recursion, Péter [1967]). Define

$$\begin{aligned} \psi(n, m) &= 0 \quad \text{if } m \cdot n = 0 \\ \psi(n+1, m+1) &= \beta(m, n, \psi(m, \gamma(m, n, \psi(m+1, n))), \psi(m+1, n)) \end{aligned}$$

Show that ψ can be defined from β, γ using higher-order primitive recursion.

5.3.7. EXERCISE (Dialectica translation). We closely follow Troelstra [1973], Section 3.5; the solution can be found there. Let \mathbf{HA}^ω be the theory of higher-order primitive recursive functionals equipped with many-sorted intuitionistic predicate logic with equality for natural numbers and axioms for arithmetic, in particular the schema of arithmetical induction:

$$(\varphi(0) \wedge \forall x (\varphi(x) \Rightarrow \varphi(x+1))) \Rightarrow \forall x \varphi(x)$$

The *Dialectica interpretation* of Gödel [1958], D-interpretation for short, assigns to every formula φ in the language of \mathbf{HA}^ω a formula $\varphi^D \equiv \exists \vec{x} \forall \vec{y} \varphi_D(\vec{x}, \vec{y})$ in the same language. The types of \vec{x}, \vec{y} depend on the logical structure of φ only. We define φ_D and φ^D by induction on φ :

1. If φ is prime, that is, an equation of lowest type, then $\varphi^D \equiv \varphi_D \equiv \varphi$.

For the binary connectives, assume $\varphi^D \equiv \exists \vec{x} \forall \vec{y} \varphi_D(\vec{x}, \vec{y})$, $\psi^D \equiv \exists \vec{u} \forall \vec{v} \psi_D(\vec{u}, \vec{v})$.

2. $(\varphi \wedge \psi)^D \equiv \exists \vec{x}, \vec{u} \forall \vec{y}, \vec{v} (\varphi \wedge \psi)_D$, with $(\varphi \wedge \psi)_D \equiv (\varphi_D(\vec{x}, \vec{y}) \wedge \psi_D(\vec{u}, \vec{v}))$.

3. $(\varphi \vee \psi)^D \equiv \exists z, \vec{x}, \vec{u} \forall \vec{y}, \vec{v} (\varphi \vee \psi)_D$, with $(\varphi \vee \psi)_D \equiv ((z = 0 \Rightarrow \varphi_D(\vec{x}, \vec{y})) \wedge (z \neq 0 \Rightarrow \psi_D(\vec{u}, \vec{v})))$.
4. $(\varphi \Rightarrow \psi)^D \equiv \exists \vec{u}', \vec{y}' \forall \vec{x}, \vec{v} (\varphi \Rightarrow \psi)_D$, with $(\varphi \Rightarrow \psi)_D \equiv (\varphi_D(\vec{x}, \vec{y}' \vec{x} \vec{v}) \Rightarrow \psi_D(\vec{u}' \vec{x}, \vec{v}))$.

Note that the clause for $\varphi \Rightarrow \psi$ introduces quantifications over higher types than those used for the formulas φ, ψ . This is also the case for formulas of the form $\forall z \varphi(z)$, see the sixth case below. For both quantifier clauses, assume $\varphi^D(z) \equiv \exists \vec{x} \forall \vec{y} \varphi_D(\vec{x}, \vec{y}, z)$.

5. $(\exists z \varphi(z))^D \equiv \exists z, \vec{x} \forall \vec{y} (\exists z \varphi(z))_D$, with $(\exists z \varphi(z))_D \equiv \varphi_D(\vec{x}, \vec{y}, z)$.
6. $(\forall z \varphi(z))^D \equiv \exists \vec{x}' \forall z, \vec{y} (\forall z \varphi(z))_D$, with $(\forall z \varphi(z))_D \equiv \varphi_D(\vec{x}' z, \vec{y}, z)$.

With φ, ψ as in the case of a binary connective, determine $(\varphi \Rightarrow (\varphi \vee \psi))^D$ and give a sequence \vec{t} of higher-order primitive recursive functionals such that $\forall \vec{y} (\varphi \Rightarrow (\varphi \vee \psi))_D(\vec{t}, \vec{y})$. We say that thus the D-interpretation of $(\varphi \Rightarrow (\varphi \vee \psi))^D$ is validated by higher-order primitive recursive functionals. Validate the D-interpretation of $(\varphi \Rightarrow (\varphi \wedge \psi))^D$. Validate the D-interpretation of induction. The result of Gödel [1958] can now be rendered as: the D-interpretation of every theorem of HA^ω can be validated by higher-order primitive recursive functionals. This yields a consistency proof for HA^ω , since $0 = 1$ cannot be validated. Note that the D-interpretation and the successive validation translates arbitrarily quantified formulas into universally quantified propositional combinations of equations.

Syntax of λ_T

In this section we formalize Gödel's \mathcal{T} as an extension of the simply typed lambda calculus λ_{\rightarrow}^o , called λ_T .

5.3.8. DEFINITION. The *types* of λ_T are the types of λ_{\rightarrow}^o over a base type \mathbf{N} . The *terms* of λ_T are obtained by adding to the term formation rules of λ_{\rightarrow}^o the constants $0 : \mathbf{N}$, $S^+ : \mathbf{N} \rightarrow \mathbf{N}$ and $R_A : A \rightarrow (A \rightarrow \mathbf{N} \rightarrow A) \rightarrow \mathbf{N} \rightarrow A$ for all types A . We denote the set of (closed) terms of type A by $\Lambda_T(A)$ ($\Lambda_T^\emptyset(A)$) and put $\Lambda_T = \bigcup_A \Lambda_T(A)$ ($\Lambda_T^\emptyset = \bigcup_A \Lambda_T^\emptyset(A)$). Terms constructed from 0 and S^+ only are called *numerals*, with 1 abbreviating $S^+(0)$, 2 abbreviating $S^+(S^+(0))$, and so on. An arbitrary numeral will be denoted by n . We define inductively $n^{A \rightarrow B} \equiv \lambda x^A. n^B$, with $n^{\mathbf{N}} \equiv n$.

The *formulas* of λ_T are equations between terms (of the same type). The *theory* of λ_T is axiomatized by equality axioms and rules, β -conversion and the schema of higher-order primitive recursion from the previous section. The *reduction relation* \rightarrow_T of λ_T is the compatible closure of the β -rules and the following (schematic) rules for the constants R_A :

$$\begin{aligned} R_A M N 0 &\rightarrow_T M \\ R_A M N (S^+ P) &\rightarrow_T N(R_A M N P) \end{aligned}$$

5.3.9. THEOREM. *The conversion relation generated by \rightarrow_T coincides with the theory λ_T .*

PROOF. By an easy extension of the proof of this result in untyped lambda calculus, see B[1984] Proposition 3.2.1. ■

5.3.10. LEMMA. *Every closed normal form of type \mathbf{N} is a numeral.*

PROOF. Consider the leftmost symbol of a closed normal form of type \mathbf{N} . This symbol cannot be a variable since the term is closed. The leftmost symbol cannot be a λ , since abstraction terms are not of type \mathbf{N} and a redex is not a normal form. If the leftmost symbol is 0 , then the term is the numeral 0 . If the leftmost symbol is S^+ , then the term must be of the form S^+P , with P a closed normal form of type \mathbf{N} . If the leftmost term is R , then for typing reasons the term must be $RMNP\vec{Q}$, with P a closed normal form of type \mathbf{N} . In the latter two cases we can complete the argument by induction, since P is a smaller term. Hence P is a numeral, so also S^+P . The case $RMNP$ with P a numeral can be excluded, as $RMNP$ should be a normal form. ■

We now prove SN and CR for λ_T , two results that could be proved independently from each other. However, the proof of CR can be simplified by using SN, which we prove first by an extension of the proof of SN for λ_{ω}^o , Theorem 2.2.2.

5.3.11. THEOREM. *Every $M \in \Lambda_T$ is SN with respect to \rightarrow_T .*

PROOF. We recall the notion of computability from the proof of Theorem 2.2.2, generalised to terms of λ_T . We shall frequently use that computable terms are SN, see formula (2) in the proof of Theorem 2.2.2. In view of the definition of computability it suffices to prove that the constants $0, S^+, R_A$ of λ_T are computable. The constant $0 : \mathbf{N}$ is computable since it is SN. Consider S^+P with computable $P : \mathbf{N}$, so P is SN and hence S^+P . It follows that S^+ is computable. In order to prove that R_A is computable, assume that M, N, P are computable and of appropriate type such that R_AMNP is of type A . Since $P : \mathbf{N}$ is computable, it is SN. Since \rightarrow_T is finitely branching, P has only finitely many normal forms, which are numerals by Lemma 5.3.10. Let $\#P$ be the largest of those numerals. We shall prove by induction on $\#P$ that R_AMNP is computable. Let \vec{Q} be computable such that $R_AMNP\vec{Q}$ is of type \mathbf{N} . We have to show that $R_AMNP\vec{Q}$ is SN. If $\#P = 0$, then every reduct of $R_AMNP\vec{Q}$ passes through a reduct of $M\vec{Q}$, and SN follows since $M\vec{Q}$ is computable. If $\#P = S^+n$, then every reduct of $R_AMNP\vec{Q}$ passes through a reduct of $N(R_AMNP')P'\vec{Q}$, where P' is such that S^+P' is a reduct of P . Then we have $\#P' = n$ and by induction it follows that R_AMNP' is computable. Now SN follows since all terms involved are computable. We have proved that R_AMNP is computable whenever M, N, P are, and hence R_A is computable. ■

5.3.12. THEOREM. *Every $M \in \Lambda_T$ is WCR with respect to \rightarrow_T .*

PROOF. Different redexes in the same term are either completely disjoint, or one redex is included in the other. In the first case the order of the reduction steps is irrelevant, and in the second case a common reduct can be obtained by reducing (possibly multiplied) included redexes. ■

5.3.13. THEOREM. Every $M \in \Lambda_T$ is CR with respect to \rightarrow_T .

PROOF. By Newman's Lemma 5.3.14, using Theorem 5.3.11. ■

5.3.14. LEMMA (Newman, localized). Let S be a set and \rightarrow a binary relation on S that is WCR. For every $a \in S$ we have: if $a \in \text{SN}$, then $a \in \text{CR}$.

PROOF. Call an element *ambiguous* if it reduces to two (or more) distinct normal forms. Assume $a \in \text{SN}$, then a reduces to at least one normal form and all reducts of a are SN. It suffices for $a \in \text{CR}$ to prove that a is not ambiguous, i.e. that a reduces to exactly one normal form. Assume by contradiction that a is ambiguous, reducing to different normal forms n_1, n_2 , say $a \rightarrow b \rightarrow \cdots \rightarrow n_1$ and $a \rightarrow c \rightarrow \cdots \rightarrow n_2$. Applying WCR to the diverging reduction steps yields a common reduct d such that $b \twoheadrightarrow d$ and $c \twoheadrightarrow d$. Since $d \in \text{SN}$ reduces to a normal form, say n , distinct of at least one of n_1, n_2 , it follows that at least one of b, c is ambiguous. See Figure 5.1. Hence a has a one-step reduct which is again ambiguous and SN. Iterating this argument yields an infinite reduction sequence contradicting $a \in \text{SN}$, so a cannot be ambiguous. ■

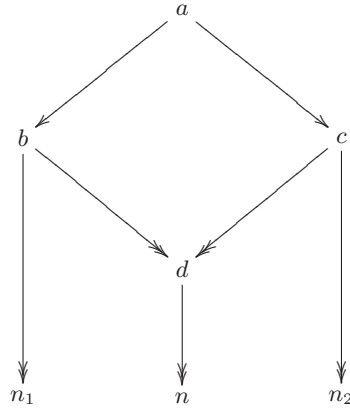


Figure 5.1: Ambiguous a has ambiguous reduct b or c .

If one considers λ_T also with η -reduction, then the above results can also be obtained. For SN it simply suffices to strengthen the notion of computability for the base case to SN with also η -reductions included. WCR and hence CR are harder to obtain and require techniques like η -postponement, see Barendregt [1984], Section 15.1.6.

Semantics of λ_T

In this section we give some interpretations of Gödel's \mathcal{T} , preceded by a general model definition of λ_T building on that of λ_{\rightarrow}^o .

5.3.15. DEFINITION. A model of λ_T is a model of λ_{\rightarrow}^o with interpretations of the constants 0 , S^+ and R_A for all A , such that the schema of higher-order primitive recursion is valid.

5.3.16. **EXAMPLE.** Recall the full typestructure over the natural numbers, that is, sets $\mathbb{N}_{\mathbb{N}} = \mathbb{N}$ and $\mathbb{N}_{A \rightarrow B} = \mathbb{N}_A \rightarrow \mathbb{N}_B$, with set-theoretic application. The full typestructure becomes the canonical model of λ_T by interpreting 0 as 0 , S^+ as the successor function, and the constants R_A as primitive recursors of the right type. The proof that $\llbracket R_A \rrbracket$ is well-defined goes by induction.

5.3.17. **EXERCISE.** Consider for any type B the set of closed terms of type B modulo convertibility. Prove that this yields a model for Gödel's \mathcal{T} . This model is called the *closed term model* of Gödel's \mathcal{T} .

Let $*$ be *Kleene application*, that is, $i * n$ stands for applying the i -th partial recursive function to the input n . If this yield a result, then we flag $i * n \downarrow$, otherwise $i * n \uparrow$. Equality between expressions with Kleene application is taken to be strict, that is, equality does only hold if left and right hand sides do yield a result and the results are equal. Similarly, $i * n \in S$ should be taken in the strict sense of $i * n$ actually yielding a result in S .

5.3.18. **EXERCISE.** By induction we define for every type B a set $HRO_B \subseteq \mathbb{N}$:

$$\begin{aligned} HRO_{\mathbb{N}} &= \mathbb{N} \\ HRO_{B \rightarrow B'} &= \{x \in \mathbb{N} \mid x * y \in HRO_{B'} \text{ for all } y \in HRO_B\} \end{aligned}$$

Prove that HRO with Kleene application constitutes a model for Gödel's \mathcal{T} .

5.3.19. **EXERCISE.** By simultaneous induction we define for every type B a set $HEO_B \subseteq \mathbb{N}$ equipped with an equivalence relation $=_B$ by

$$\begin{aligned} HEO_{\mathbb{N}} &= \mathbb{N} \\ x =_{\mathbb{N}} y &\iff x = y \\ HEO_{B \rightarrow B'} &= \{x \in \mathbb{N} \mid x * y \in HEO_{B'} \text{ for all } y \in HEO_B \text{ and} \\ &\quad x * y =_{B'} x * y' \text{ for all } y, y' \in HEO_B \text{ with } y =_B y'\} \\ x =_{B \rightarrow B'} x' &\iff x, x' \in HEO_{B \rightarrow B'} \text{ and } x * y =_{B'} x' * y \text{ for all } y \in HEO_B \end{aligned}$$

Prove that HEO with Kleene application constitutes a model for Gödel's \mathcal{T} .

5.3.20. **EXERCISE.** Recall that extensionality essentially means that objects having the same applicative behaviour can be identified. Which of the above models of λ_T , the full type structure, the closed term model, HRO and HEO , is extensional?

Computational strength

Brief review of ordinal theory

Here are some ordinal numbers, simply called *ordinals*, in increasing order:

$$0, \underline{1}, \underline{2}, \dots, \omega, \omega + \underline{1}, \omega + \underline{2}, \dots, \omega + \omega = \omega \cdot \underline{2}, \dots, \omega \cdot \omega = \omega^2, \dots, \omega^\omega, \dots, \omega^{(\omega^\omega)}, \dots$$

Apart from ordinals, also some basic operations of ordinal arithmetic are visible, namely addition, multiplication and exponentiation, denoted in the same way as in highschool algebra. The dots ... stand for many more ordinals in between, produced by iterating the previous construction process.

The most important structural property of ordinals is that $<$ is a well-order, that is, an order such that every non-empty subset contains a smallest element. This property leads to the principle of (transfinite) induction for ordinals, stating that $P(\alpha)$ holds for all ordinals α whenever P is *inductive*, that is, $P(\alpha)$ follows from $\forall \beta < \alpha. P(\beta)$ for all α .

In fact the arithmetical operations are defined by means of two more primitive operations on ordinals, namely the *successor* operation $+1$ and the *supremum* operation \bigcup . The supremum $\bigcup a$ of a set of ordinals a is the least upper bound of a , which is equal to the smallest ordinal greater than all ordinals in the set a . A typical example of the latter is the ordinal ω , the first infinite ordinal, which is the supremum of the sequence of the finite ordinals \underline{n} produced by iterating the successor operation on $\underline{0}$.

These primitive operations divide the ordinals in three classes: the *successor* ordinals of the form $\alpha + 1$, the *limit* ordinals $\lambda = \bigcup \{\alpha \mid \alpha < \lambda\}$, i.e. ordinals which are the supremum of the set of smaller ordinals, and the *zero* ordinal $\underline{0}$. (In fact $\underline{0}$ is the supremum of the empty set, but is not considered to be a limit ordinal.) Thus we have zero, successor and limit ordinals.

Addition, multiplication and exponentiation are now defined according to Table 5.1. Ordinal arithmetic has many properties in common with ordinary arithmetic, but there are some notable exceptions. For example, addition and multiplication are associative but not commutative: $\underline{1} + \omega = \omega \neq \omega + \underline{1}$ and $\underline{2} \cdot \omega = \omega \neq \omega \cdot \underline{2}$. Furthermore, multiplication is left distributive over addition, but not right distributive: $(\underline{1} + \underline{1}) \cdot \omega = \omega \neq \underline{1} \cdot \omega + \underline{1} \cdot \omega$. The sum $\alpha + \beta$ is weakly increasing in α and strictly increasing in β . Similarly for the product $\alpha \cdot \beta$ with $\alpha > \underline{0}$. The only exponentiations we shall use, 2^α and ω^α , are strictly increasing in α .

Addition	Multiplication	Exponentiation ($\alpha > \underline{0}$)
$\alpha + \underline{0} = \alpha$	$\alpha \cdot \underline{0} = \underline{0}$	$\alpha^{\underline{0}} = \underline{1}$
$\alpha + (\beta + \underline{1}) = (\alpha + \beta) + \underline{1}$	$\alpha \cdot (\beta + \underline{1}) = \alpha \cdot \beta + \alpha$	$\alpha^{\beta + \underline{1}} = \alpha^\beta \cdot \alpha$
$\alpha + \lambda = \bigcup \{\alpha + \beta \mid \beta < \lambda\}$	$\alpha \cdot \lambda = \bigcup \{\alpha \cdot \beta \mid \beta < \lambda\}$	$\alpha^\lambda = \bigcup \{\alpha^\beta \mid \beta < \lambda\}$

Table 5.1: Ordinal arithmetic (with λ limit ordinal in the third row).

The operations of ordinal arithmetic as defined above provide examples of a more general phenomenon called transfinite iteration, to be defined below.

5.3.21. DEFINITION. Let f be an ordinal function. Define by induction $f^{\underline{0}}(\alpha) = \alpha$, $f^{\beta + \underline{1}}(\alpha) = f(f^\beta(\alpha))$ and $f^\lambda(\alpha) = \bigcup \{f^\beta(\alpha) \mid \beta < \lambda\}$ for every limit ordinal λ . We call f^β the β -th *transfinite iteration* of f .

As examples we redefine the arithmetical operations above: $\alpha + \beta = f^\beta(\alpha)$ with f the successor function; $\alpha \cdot \beta = g_\alpha^\beta(\underline{0})$ with $g_\alpha(\gamma) = \gamma + \alpha$; $\alpha^\beta = h_\alpha^\beta(\underline{1})$ with $h_\alpha(\gamma) = \gamma \cdot \alpha$.

5.3.22. EXERCISE. Verify the redefinition of the ordinal arithmetic is correct.

We proceed with the canonical construction for finding the least fixpoint of a weakly increasing ordinal function if there exists one.

5.3.23. LEMMA. *Let f be a weakly increasing ordinal function. Then:*

- (i) $f^{\alpha+1}(\underline{0}) \geq f^\alpha(\underline{0})$ for all α ;
- (ii) $f^\alpha(\underline{0})$ is weakly increasing in α ;
- (iii) $f^\alpha(\underline{0})$ does not surpass any fixpoint of f ;
- (iv) $f^\alpha(\underline{0})$ is strictly increasing (and hence $f^\alpha(\underline{0}) \geq \alpha$), until a fixpoint of f is reached, after which $f^\alpha(\underline{0})$ becomes constant.

5.3.24. EXERCISE. Prove the above lemma. More precisely:

- (i) To be proved by induction on α ;
- (ii) Prove $\alpha \leq \beta \Rightarrow f^\alpha(\underline{0}) \leq f^\beta(\underline{0})$ by induction on β ;
- (iii) Assume $f(\beta) = \beta$ and prove $f^\alpha(\underline{0}) \leq \beta$ by induction on α ;
- (iv) Prove $\alpha < \beta \Rightarrow f^\alpha(\underline{0}) < f^\beta(\underline{0})$ for all α, β such that $f^\alpha(\underline{0})$ is below any fixpoint, by induction on β .

If a weakly increasing ordinal function f has a fixpoint, then it has a smallest fixpoint and Lemma 5.3.23 above guarantees that this so-called *least fixpoint* is of the form $f^\alpha(\underline{0})$, that is, can be obtained by transfinite iteration of f starting at $\underline{0}$. This justifies the following definition.

5.3.25. DEFINITION. Let f be a weakly increasing ordinal function having a least fixpoint which we denote by $\text{lfp}(f)$. The *closure ordinal* of f is the smallest ordinal α such that $f^\alpha(\underline{0}) = \text{lfp}(f)$.

Closure ordinals can be arbitrarily large, or may not even exist. The following lemma gives a condition under which the closure ordinal exists and does not surpass ω .

5.3.26. LEMMA. *If f is a weakly increasing ordinal function such that $f(\lambda) = \bigcup\{f(\alpha) \mid \alpha < \lambda\}$ for every limit ordinal λ , then the closure ordinal exists and is at most ω .*

PROOF. Let conditions be as in the lemma. Consider the sequence of finite iterations of f : $\underline{0}, f(\underline{0}), f(f(\underline{0}))$ and so on. If this sequence becomes constant, then the closure ordinal is finite. If the sequence is strictly increasing, then the supremum must be a limit ordinal, say λ . Then we have $f(\lambda) = \bigcup\{f(\alpha) \mid \alpha < \lambda\} = f^\omega(\underline{0}) = \lambda$, so the closure ordinal is ω . ■

5.3.27. EXERCISE. Justify the equation $f(\lambda) = \lambda$ in the proof above.

For example, $f(\alpha) = \underline{1} + \alpha$ has $\text{lfp}(f) = \omega$, and $f(\alpha) = (\omega + \underline{1}) \cdot \alpha$ has $\text{lfp}(f) = \underline{0}$. In contrast, $f(\alpha) = \alpha + \underline{1}$ has no fixpoint (note that the latter f is weakly increasing, but

the condition on limit ordinals is not satisfied). Finally, $f(\alpha) = \underline{2}^\alpha$ has $\text{lfp}(f) = \omega$, and the least fixpoint of $f(\alpha) = \omega^\alpha$ is denoted by ϵ_0 , being the supremum of the sequence:

$$0, \omega^0 = \underline{1}, \omega^1 = \omega, \omega^\omega, \omega^{\omega^\omega}, \omega^{\omega^{\omega^\omega}}, \dots$$

In the following proposition we formulate some facts about ordinals that we need in the sequel.

5.3.28. PROPOSITION. (i) *Every ordinal $\alpha < \epsilon_0$ can be written uniquely as $\alpha = \omega^{\alpha_1} + \omega^{\alpha_2} + \dots + \omega^{\alpha_n}$ with $n \geq 0$ and $\alpha_1, \alpha_2, \dots, \alpha_n$ a weakly decreasing sequence of ordinals smaller than α .*

(ii) *For all α, β we have $\omega^\alpha + \omega^\beta = \omega^\beta$ if and only if $\alpha < \beta$. if α is a limit and $\beta < \gamma + \underline{2}^\alpha$, then there is an $\alpha' < \alpha$ such that $\beta < \gamma + \underline{2}^{\alpha'}$.*

PROOF. (i) This is a special case of Cantor normal forms with base ω , the generalization of the position system for numbers to ordinals, where terms of the form $\omega^\alpha \cdot \underline{n}$ are written as $\omega^\alpha + \dots + \omega^\alpha$ (n summands). The fact that the exponents in the Cantor normal form are *strictly* less than α comes from the assumption that $\alpha < \epsilon_0$.

(ii) The proof of this so-called absorption property goes by a induction on β . The case $\alpha \geq \beta$ can be dealt with by using Cantor normal forms. ■

From now on *ordinal* will mean *ordinal less than ϵ_0* , unless explicitly stated otherwise. This also applies to $\forall\alpha, \exists\alpha, f(\alpha)$ and so on.

Encoding ordinals in the natural numbers

Systematic enumeration of grid points in the plane, such as shown in Figure 5.2, yields an encoding of pairs $\langle x, y \rangle$ of natural numbers x, y as given in Definition 5.3.29.

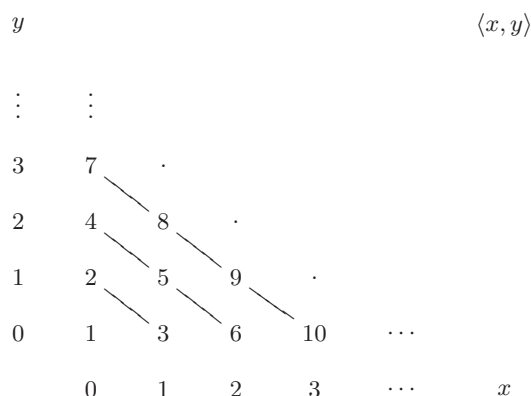


Figure 5.2: $\langle x, y \rangle$ -values for $x + y \leq 3$

Finite sequences $[x_1, \dots, x_k]$ of natural numbers, also called *lists*, can now be encoded by iterating the pairing function. The number 0 does not encode a pair and can hence

be used to encode the empty list $[]$. All functions and relations involved, including projection functions to decompose pairs and lists, are easily seen to be primitive recursive.

5.3.29. DEFINITION. Recall that $1+2+\dots+n = \frac{1}{2}n(n+1)$ gives the number of grid points satisfying $x+y < n$. The function $-$ below is to be understood as cut-off subtraction, that is, $x-y = 0$ whenever $y \geq x$. Define the following functions:

$$\begin{aligned}\langle x, y \rangle &= \frac{1}{2}(x+y)(x+y+1) + x + 1 \\ \text{sum}(p) &= \min\{n \mid p \leq \frac{1}{2}n(n+1)\} - 1 \\ x(p) &= p - \langle 0, \text{sum}(p) \rangle \\ y(p) &= \text{sum}(p) - x(p)\end{aligned}$$

Now let $[] = 0$ and, for $k > 0$, $[x_1, \dots, x_k] = \langle x_1, [x_2, \dots, x_k] \rangle$ encode lists. Define $\text{1th}(0) = 0$ and $\text{1th}(p) = 1 + \text{1th}(y(p))$ ($p > 0$) to compute the length of a list.

The following lemma is a straightforward consequence of the above definition.

5.3.30. LEMMA. *For all $p > 0$ we have $p = \langle x(p), y(p) \rangle$. Moreover, $\langle x, y \rangle > x$, $\langle x, y \rangle > y$, $\text{1th}([x_1, \dots, x_k]) = k$ and $\langle x, y \rangle$ is strictly increasing in both arguments. Every natural number encodes a unique list of smaller natural numbers. Every natural number encodes a unique list of lists of lists and so on, ending with the empty list.*

Based on the Cantor normal form and the above encoding of lists we can represent ordinals below ϵ_0 as natural numbers in the following way. We write $\bar{\alpha}$ for the natural number representing the ordinal α .

5.3.31. DEFINITION. Let $\alpha < \epsilon_0$ have Cantor normal form $\omega^{\alpha_1} + \dots + \omega^{\alpha_k}$. We encode α by putting $\bar{\alpha} = [\bar{\alpha}_1, \bar{\alpha}_2, \dots, \bar{\alpha}_n]$. This representation is well-defined since every α_i ($1 \leq i \leq n$) is strictly smaller than α . The zero ordinal 0 , having the empty sum as Cantor normal form, is thus represented by the empty list $[]$, so by the natural number 0 .

Examples are $\bar{0} = []$, $\bar{1} = [[]]$, $\bar{2} = [[], []]$, \dots and $\bar{\omega} = [[[]]]$, $\overline{\omega+1} = [[[]], []]$ and so on. Observe that $[[[]], [[]]]$ does not represent an ordinal as $\omega^0 + \omega^1$ is not a Cantor normal form. The following lemmas allow one to identify which natural numbers represent ordinals and to compare them.

5.3.32. LEMMA. *Let \prec be the lexicographic ordering on lists. Then \prec is primitive recursive and $\bar{\alpha} \prec \bar{\beta} \iff \alpha < \beta$ for all $\alpha, \beta < \epsilon_0$.*

PROOF. Define $\langle x, y \rangle \prec \langle x', y' \rangle \iff (x \prec x') \vee (x = x' \wedge y \prec y')$ and $x \not\prec 0$, $0 \prec \langle x, y \rangle$. The primitive recursive relation \prec is the lexicographic ordering on pairs, and hence also on lists. Now the lemma follows using Cantor normal forms. (Note that \prec is not a well-order itself, as $[1] \prec [0, 1] \prec [0, 0, 1], \dots$ has no smallest element.) ■

5.3.33. LEMMA. *For all $x \in \mathbb{N}$, define:*

- (i) $\text{Ord}(x)$ if and only if $x = \bar{\alpha}$ for some ordinal $\alpha < \epsilon_0$.
- (ii) $\text{Succ}(x)$ if and only if $x = \bar{\alpha}$ for some successor ordinal $< \epsilon_0$.
- (iii) $\text{Lim}(x)$ if and only if $x = \bar{\alpha}$ for some limit ordinal $< \epsilon_0$.
- (iv) $\text{Fin}(x)$ if and only if $x = \bar{\alpha}$ for some ordinal $\alpha < \omega$.

Then Ord , Fin , Succ and Lim are primitive recursive predicates.

PROOF.

- (i) Put $\text{Ord}(0)$ and $\text{Ord}(\langle x, y \rangle) \iff (\text{Ord}(x) \wedge \text{Ord}(y) \wedge (y > 0 \Rightarrow x(y) \preceq x))$.
- (ii) Put $\neg \text{Succ}(0)$ and $\text{Succ}(\langle x, y \rangle) \iff (\text{Ord}(\langle x, y \rangle) \wedge (x > 0 \Rightarrow \text{Succ}(y)))$.
- (iii) Put $\text{Lim}(x) \iff (\text{Ord}(x) \wedge \neg \text{Succ}(s) \wedge x \neq [])$.
- (iv) Put $\text{Fin}(x) \iff (x = [] \vee (x = \langle 0, y \rangle \wedge \text{Fin}(y)))$. ■

5.3.34. LEMMA. *There exist primitive recursive functions exp (base ω exponentiation), succ (successor), pred (predecessor), plus (addition), exp2 (base $\underline{2}$ exponentiation) such that for all α, β : $\text{exp}(\bar{\alpha}) = \bar{\omega}^\alpha$, $\text{succ}(\bar{\alpha}) = \bar{\alpha} + \underline{1}$, $\text{pred}(\underline{0}) = \underline{0}$, $\text{pred}(\bar{\alpha} + \underline{1}) = \bar{\alpha}$, $\text{plus}(\bar{\alpha}, \bar{\beta}) = \bar{\alpha} + \bar{\beta}$, $\text{exp2}(\bar{\alpha}) = \underline{2}^\alpha$.*

PROOF. Put $\text{exp}(x) = [x]$. Put $\text{succ}(0) = \langle 0, 0 \rangle$ and $\text{succ}(\langle x, y \rangle) = \langle x, \text{succ}(y) \rangle$, then $\text{succ}([x_1, \dots, x_k]) = [x_1, \dots, x_k, 0]$. Put $\text{pred}(0) = 0$, $\text{pred}(\langle x, 0 \rangle) = x$ and $\text{pred}(\langle x, y \rangle) = \langle x, \text{pred}(y) \rangle$ for $y > 0$. For plus , use the absorption property in adding the Cantor normal forms of α and β . For exp2 we use $\omega^\beta = \underline{2}^{\omega \cdot \beta}$. Let α have Cantor normal form $\omega^{\alpha_1} + \dots + \omega^{\alpha_k}$. Then $\omega \cdot \alpha = \omega^{1+\alpha_1} + \dots + \omega^{1+\alpha_k}$. By absorption, $\underline{1} + \alpha_i = \alpha_i$ whenever $\alpha_i \geq \omega$. It follows that we have $\alpha = \omega \cdot (\omega^{\alpha_1} + \dots + \omega^{\alpha_i} + \omega^{\underline{n}_1} + \dots + \omega^{\underline{n}_p}) + \underline{n}$ for suitable \underline{n}_j, n with $\alpha_1 \geq \dots \geq \alpha_i \geq \omega$, $\underline{n}_j + \underline{1} = \alpha_{i+j} < \omega$ for $1 \leq j \leq p$ and $n = k - i - p$ with $\alpha_{k'} = \underline{0}$ for all $i + p < k' \leq k$. Using $\omega^\beta = \underline{2}^{\omega \cdot \beta}$ we can calculate $\underline{2}^\alpha = \omega^\beta \cdot \underline{2}^{\underline{n}}$ with $\beta = \omega^{\alpha_1} + \dots + \omega^{\alpha_i} + \omega^{\underline{n}_1} + \dots + \omega^{\underline{n}_p}$ and n as above. If $\bar{\alpha} = [x_1, \dots, x_i, \dots, x_j, \dots, 0, \dots, 0]$, then $\bar{\beta} = [x_1, \dots, x_i, \dots, \text{pred}(x_j), \dots]$ and we can obtain $\text{exp2}(\bar{\alpha}) = \underline{2}^\alpha = \omega^\beta \cdot \underline{2}^{\underline{n}}$ by doubling n times $\bar{\omega}^\beta = \text{exp}(\bar{\beta})$ using plus . ■

5.3.35. LEMMA. *There exist primitive recursive functions num , mun such that $\text{num}(n) = \underline{n}$ and $\text{mun}(\underline{n}) = n$ for all n . In particular we have $\text{mun}(\text{num}(n)) = n$ and $\text{num}(\text{mun}(\underline{n})) = \underline{n}$ for all n . In other words, num is the order isomorphism between $(\mathbb{N}, <)$ and $(\{\underline{n} \mid n \in \mathbb{N}\}, <)$ and mun is the inverse order isomorphism.*

PROOF. Put $\text{num}(0) = 0 = []$ and $\text{num}(n+1) = \text{succ}(\text{num}(n))$ and $\text{mun}(0) = 0$ and $\text{mun}(\langle x, y \rangle) = \text{mun}(y) + 1$. ■

5.3.36. LEMMA. *There exists a primitive recursive function p such that $p(\bar{\alpha}, \bar{\beta}, \bar{\gamma}) = \bar{\alpha}'$ with $\alpha' < \alpha$ and $\beta < \gamma + \underline{2}^{\alpha'}$, provided that α is a limit and $\beta < \gamma + \underline{2}^\alpha$.*

PROOF. Let conditions be as above. The existence of α' follows directly from the definition of the operations of ordinal arithmetic on limit ordinals. The interesting point, however, is that $\bar{\alpha}'$ can be computed from $\bar{\alpha}, \bar{\beta}, \bar{\gamma}$ in a primitive recursive way, as will become clear by the following argument. If $\beta \leq \gamma$, then we can simply take

$\alpha' = 0$. Otherwise, let $\beta = \omega^{\beta_1} + \dots + \omega^{\beta_n}$ and $\gamma = \omega^{\gamma_1} + \dots + \omega^{\gamma_m}$ be Cantor normal forms. Now $\gamma < \beta$ implies that $\gamma_i < \beta_i$ for some smallest index $i \leq m$, or no such index exists. In the latter case we have $m < n$ and $\gamma_j = \beta_j$ for all $1 \leq j \leq m$, and we put $i = m + 1$. Since α is a limit, we have $\alpha = \omega \cdot \xi$ (see ...) for suitable ξ , and hence $2^\alpha = \omega^\xi$. Since $\beta < \gamma + 2^\alpha$ it follows by absorption that $\omega^{\beta_i} + \dots + \omega^{\beta_n} < \omega^\xi$. Hence $\beta_i + 1 \leq \xi$, so $\omega^{\beta_i} + \dots + \omega^{\beta_n} \leq \omega^{\beta_i} \cdot \underline{n} < \omega^{\beta_i} \cdot 2^{\underline{n}} = 2^{\omega \cdot \beta_i + \underline{n}}$. Now take $\alpha' = \omega \cdot \beta_i + \underline{n} < \omega \cdot (\beta_i + 1) \leq \omega \cdot \xi = \alpha$ and observe $\beta < \gamma + 2^{\alpha'}$. ■

From now on we will freely use ordinals in the natural numbers instead of their codes. This includes uses like α is finite instead of $\text{Fin}(\bar{\alpha})$, $\alpha < \beta$ instead of $\bar{\alpha} < \bar{\beta}$, and so on. Note that we avoid using $<$ for ordinals now, as it would be ambiguous. Phrases like $\forall \alpha P(\alpha)$ and $\exists \alpha P(\alpha)$ should be taken as relativized quantifications over natural numbers, that is, $\forall x (\text{Ord}(x) \Rightarrow P(x))$, and $\exists x (\text{Ord}(x) \wedge P(x))$, respectively. Finally, functions defined in terms of ordinals are assumed to take value 0 for arguments that do not encode any ordinal.

Transfinite induction and recursion

Transfinite induction (TI) is a principle of proof that generalizes the usual schema of structural induction from natural numbers to ordinals. Define:

$$\begin{aligned} \text{Ind}(P) &\equiv \forall \alpha ((\forall \beta < \alpha P(\beta)) \Rightarrow P(\alpha)) \\ \text{TI}_\alpha &\equiv \text{Ind}(P) \Rightarrow \forall \beta < \alpha P(\beta) \end{aligned}$$

Here $\text{Ind}(P)$ expresses that P is *inductive*, that is, $\forall \beta < \alpha P(\beta)$ induces $P(\alpha)$ for all ordinals α . For proving a property P to be inductive it suffices to prove $(\forall \beta < \alpha P(\beta)) \Rightarrow P(\alpha)$ for limit ordinals α only, in addition to $P(0)$ and $P(\alpha) \Rightarrow P(\alpha + 1)$ for all α . If a property is inductive then TI_γ implies that every ordinal up to γ has this property. (For the latter conclusion, in fact inductivity up to γ suffices. Note that ordinals may exceed ϵ_0 in this paragraph.)

By Lemma 5.3.35, TI_ω is equivalent to structural induction on the natural numbers. Obviously, the strength of TI_α increases with α . Therefore TI_α can be used to measure the proof theoretic strength of theories. Given a theory T , for which α can we prove TI_α ? We shall show that TI_α is provable in Peano Arithmetic for all ordinals $\alpha < \epsilon_0$ by a famous argument due to Gentzen.

The computational counterpart of transfinite induction is transfinite recursion TR , a principle of definition which can be used to measure computational strength. By a translation of Gentzen's argument we shall show that every function which can be defined by TR_α for some ordinal $\alpha < \epsilon_0$, is definable in Gödel's \mathcal{T} . Thus we have established a lower bound to the computational strength of Gödel's \mathcal{T} .

5.3.37. LEMMA. *The schema TI_ω is provable in Peano Arithmetic.*

PROOF. Observe that TI_ω is structural induction on an isomorphic copy of the natural numbers by Lemma 5.3.35. ■

5.3.38. LEMMA. *The schema $TI_{\omega \cdot 2}$ is provable in Peano Arithmetic with the schema TI_ω .*

PROOF. Assume TI_ω and $\text{Ind}(P)$ for some P . In order to prove $\forall \alpha < \omega \cdot 2 \ P(\alpha)$ define $P'(\alpha) \equiv \forall \beta < \omega + \alpha \ P(\beta)$. By TI_ω we have $P'(\underline{0})$. Also $P'(\alpha) \Rightarrow P'(\alpha + \underline{1})$, as $P'(\alpha)$ implies $P(\omega + \alpha)$ by $\text{Ind}(P)$. If $\text{Lim}(\alpha)$, then $\beta < \omega + \alpha$ implies $\beta < \omega + \alpha'$ for some $\alpha' < \alpha$, and hence $P'(\alpha') \Rightarrow P(\beta)$. It follows that P' is inductive, which can be combined with TI_ω to conclude $P'(\omega)$, so $\forall \beta < \omega + \omega \ P(\beta)$. This completes the proof of $TI_{\omega \cdot 2}$. ■

5.3.39. LEMMA. *The schema TI_{2^α} is provable in Peano Arithmetic with the schema TI_α , for all $\alpha < \epsilon_0$.*

PROOF. Assume TI_α and $\text{Ind}(P)$ for some P . In order to prove $\forall \alpha' < 2^\alpha \ P(\alpha')$ define $P'(\alpha') \equiv \forall \beta (\forall \beta' < \beta \ P(\beta') \Rightarrow \forall \beta' < \beta + 2^{\alpha'} \ P(\beta'))$. The intuition behind $P'(\alpha')$ is: if P holds on an arbitrary initial segment, then we can prolong this segment with $2^{\alpha'}$. The goal will be to prove $P'(\alpha)$, since we can then prolong the empty initial segment on which P vacuously holds to one of length 2^α . We prove $P'(\alpha)$ by proving first that P' is inductive and then combining this with TI_α , similar to the proof of the previous lemma. We have $P'(\underline{0})$ as P is inductive and $2^{\underline{0}} = \underline{1}$. The argument for $P'(\alpha) \Rightarrow P'(\alpha + \underline{1})$ amounts to applying $P'(\alpha)$ twice, relying on $2^{\alpha+1} = 2^\alpha + 2^\alpha$. Assume $P'(\alpha)$ and $\forall \beta' < \beta \ P(\beta')$ for some β . By $P'(\alpha)$ we have $\forall \beta' < \beta + 2^\alpha \ P(\beta')$. Hence again by $P'(\alpha)$, but now with $\beta + 2^\alpha$ instead of β , we have $\forall \beta' < \beta + 2^\alpha + 2^\alpha \ P(\beta')$. We conclude $P'(\alpha + \underline{1})$. The limit case is equally simple as in the previous lemma. It follows that P' is inductive, and the proof can be completed as explained above. ■

The general idea of the above proofs is that the stronger axiom schema is proved by applying the weaker schema to more complicated formulas (P' as compared to P). This procedure can be iterated as long as the more complicated formulas remain well-formed. In the case of Peano arithmetic we can iterate this procedure finitely many times. This yields the following result.

5.3.40. LEMMA (Gentzen). *TI_α is provable in Peano Arithmetic for every ordinal $\alpha < \epsilon_0$.*

PROOF. Use $\omega^\beta = 2^{\omega \cdot \beta}$, so $2^{\omega \cdot 2} = \omega^2$ and $2^{\omega^2} = \omega^\omega$. From ω^ω on, iterating exponentiation with base 2 yields the same ordinals as with base ω . We start with Lemma 5.3.37 to obtain TI_ω , continue with Lemma 5.3.38 to obtain $TI_{\omega \cdot 2}$, and surpass TI_α for every ordinal $\alpha < \epsilon_0$ by iterating Lemma 5.3.39 a sufficient number of times. ■

We now translate the Gentzen argument from transfinite induction to transfinite recursion, closely following the development of Terlouw [1982].

5.3.41. DEFINITION. For any functional F of type $0 \rightarrow A$ and ordinals α, β we define primitive recursively

$$[F]_\beta^\alpha(\beta') = \begin{cases} F(\beta') & \text{if } \beta' \prec \beta \preceq \alpha, \\ 0^A & \text{otherwise.} \end{cases}$$

By convention, ‘otherwise’ includes the cases in which α, β, β' are not ordinals, and the case in which $\alpha \prec \beta$. Furthermore, we define $[F]_\alpha = [F]_\alpha^\alpha$, that is, the functional F restricted to an initial segment of ordinals smaller than α .

5.3.42. DEFINITION. The class of functionals *definable by* TR_α is the smallest class of functionals which contains all primitive recursive functionals and is closed under the definition schema TR_α , defining F from G (of appropriate types) in the following way:

$$F(\beta) = G([F]_\beta^\alpha, \beta)$$

Note that, by the above definition, $F(\beta) = G(0^{0 \rightarrow A}, \beta)$ if $\alpha \prec \beta$ or if the argument of F does not encode an ordinal.

The following lemma is to be understood as the computational counterpart of Lemma 5.3.38, with the primitive recursive functionals taking over the role of Peano Arithmetic.

5.3.43. LEMMA. *Every functional definable by the schema TR_ω is \mathcal{T} -definable.*

PROOF. Let $F_0(\alpha) = G([F_0]_\alpha^\omega, \alpha)$ be defined by TR_ω . We have to show that F_0 is \mathcal{T} -definable. Define primitive recursively F_1 by $F_1(0) = 0^{0 \rightarrow A}$ and

$$F_1(n+1, \alpha) = \begin{cases} F_1(n, \alpha) & \text{if } \alpha < \underline{n} \\ G([F_1(n)]_\alpha^\omega, \alpha) & \text{otherwise} \end{cases}$$

By induction one shows $[F_0]_\alpha^\omega = [F_1(n)]_\alpha^\omega$ for all n . Define primitive recursively F_2 by $F_2(\underline{n}) = F_1(n+1, \underline{n})$ and $F_2(\alpha) = 0^A$ if α is not a finite ordinal, then $F_2 = [F_0]_\omega^\omega$. Now it is easy to define F_0 explicitly in F_2

$$F_0(\alpha) = \begin{cases} F_2(\alpha) & \text{if } \alpha < \omega \\ G(F_2, \omega) & \text{if } \alpha = \omega \\ G(0^{0 \rightarrow A}, \alpha) & \text{otherwise} \end{cases}$$

Note that we used both num and mun implicitly in the definition of F_2 . ■

The general idea of the proofs below is that the stronger schema is obtained by applying the weaker schema to functionals of more complicated types.

5.3.44. LEMMA. *Every functional definable by the schema $\text{TR}_{\omega \cdot 2}$ is definable by the schema TR_ω .*

PROOF. Put $\omega \cdot 2 = \alpha$ and let $F_0(\beta) = G([F_0]_\beta^\alpha, \beta)$ be defined by TR_α . We have to show that F_0 is definable by TR_ω (applied with functionals of more complicated types). First define $F_1(\beta) = G([F_1]_\beta^\omega, \beta)$ by TR_ω . Then we can prove $F_1(\beta) = F_0(\beta)$ for all $\beta < \omega$ by TI_ω . So we have $[F_1]_\omega = [F_0]_\omega$, which is to be compared to $P'(\underline{0})$ in the proof of Lemma 5.3.38. Now define H of type $0 \rightarrow (0 \rightarrow A) \rightarrow (0 \rightarrow A)$ by TR_ω as follows. The

more complicated type of H as compared to the type $0 \rightarrow A$ of F is the counterpart of the more complicated formula P' as compared to P in the proof of Lemma 5.3.38.

$$\begin{aligned} H(0, F) &= [F_1]_\omega \\ H(\beta + 1, F, \beta') &= \begin{cases} H(\beta, F, \beta') & \text{if } \beta' < \omega + \beta \\ G(H(\beta, F), \beta') & \text{if } \beta' = \omega + \beta \\ 0^A & \text{otherwise} \end{cases} \end{aligned}$$

This definition can easily be casted in the form $H(\beta) = G'([H]_\beta^\omega, \beta)$ for suitable G' , so that H is actually defined by TR_ω . We can prove $H(\beta, 0^{0 \rightarrow A}) = [F_0]_{\omega+\beta}^\alpha$ for all $\beta < \omega$ by TI_ω . Finally we define

$$F_2(\beta') = \begin{cases} F_1(\beta') & \text{if } \beta' < \omega \\ G(H(\beta, 0^{0 \rightarrow A}), \beta') & \text{if } \beta' = \omega + \beta < \alpha \\ G(0^{0 \rightarrow A}, \beta') & \text{otherwise} \end{cases}$$

Note that F_2 is explicitly defined in G and H and therefore defined by TR_ω only. One easily shows that $F_2 = F_0$, which completes the proof of the lemma. ■

5.3.45. LEMMA. *Every functional definable by the schema TR_{2^α} is definable by the schema TR_α , for all $\alpha < \epsilon_0$.*

PROOF. Let $F_0(\beta) = G([F_0]_\beta^{2^\alpha}, \beta)$ be defined by TR_{2^α} . We have to show that F_0 is definable by TR_α (applied with functionals of more complicated types). Like in the previous proof, we will define by TR_α an auxiliary functional H in which F_0 can be defined explicitly. The complicated type of H compensates for the weaker definition principle. The following property satisfied by H is to be understood in the same way as the property P' in the proof of Lemma 5.3.39, namely that we can prolong initial segments with $\underline{2}^\alpha$.

$$\text{prop}H(\alpha') \equiv \forall \beta, F \ ([F]_\beta^{2^\alpha} = [F_0]_\beta^{2^\alpha} \Rightarrow [H(\alpha', \beta, F)]_{\beta+\underline{2}^{\alpha'}}^{2^\alpha} = [F_0]_{\beta+\underline{2}^{\alpha'}}^{2^\alpha})$$

To make $\text{prop}H$ come true, define H of type $0 \rightarrow 0 \rightarrow (0 \rightarrow A) \rightarrow (0 \rightarrow A)$ as follows.

$$H(0, \beta, F, \beta') = \begin{cases} F(\beta') & \text{if } \beta' < \beta \leq \underline{2}^\alpha \\ G([F]_\beta^{2^\alpha}, \beta) & \text{if } \beta' = \beta \leq \underline{2}^\alpha \\ 0^A & \text{otherwise} \end{cases}$$

$$H(\alpha' + 1, \beta, F) = H(\alpha', \beta + \underline{2}^{\alpha'}, H(\alpha', \beta, F))$$

If α' is a limit ordinal, then we use the function p from Lemma 5.3.36.

$$H(\alpha', \beta, F, \beta') = \begin{cases} H(p(\alpha', \beta', \beta), \beta, F, \beta') & \text{if } \beta' < \beta + \underline{2}^{\alpha'} \\ 0^A & \text{otherwise} \end{cases}$$

This definition can easily be casted in the form $H(\beta) = G'([H]_\beta^\alpha, \beta)$ for suitable G' , so that H is in fact defined by TR_α . We shall prove that $\text{prop}H(\alpha')$ is inductive, and

conclude $\text{prop}H(\alpha')$ for all $\alpha' \leq \alpha$ by TI_α . This implies $[H(\alpha', \underline{0}, 0^{0 \rightarrow A})]_{\underline{2}^{\alpha'}}^{\underline{2}^\alpha} = [F_0]_{\underline{2}^{\alpha'}}^{\underline{2}^\alpha}$ for all $\alpha' \leq \alpha$, so that one could manufacture F_0 from H in the following way:

$$F_0(\beta) = \begin{cases} H(\alpha, \underline{0}, 0^{0 \rightarrow A}, \beta) & \text{if } \beta < \underline{2}^\alpha \\ G(H(\alpha, \underline{0}, 0^{0 \rightarrow A}), \beta) & \text{if } \beta = \underline{2}^\alpha \\ G(0^{0 \rightarrow A}, \beta) & \text{otherwise} \end{cases}$$

It remains to show that $\text{prop}H(\alpha')$ is inductive up to and including α . For the case $\alpha' = \underline{0}$ we observe that $H(\underline{0}, \beta, F)$ follows F up to β , applies G to the initial segment of $[F]_\beta^{\underline{2}^\alpha}$ in β , and zeroes after β . This entails $\text{prop}H(\underline{0})$, as $\underline{2}^{\underline{0}} = \underline{1}$. Analogous to the successor case in the proof of Lemma 5.3.39, we prove $\text{prop}H(\alpha + \underline{1})$ by applying $\text{prop}H(\alpha)$ twice, once with β and once with $\beta + \underline{2}^\alpha$. Given β and F we infer:

$$\begin{aligned} [F]_\beta^{\underline{2}^\alpha} = [F_0]_\beta^{\underline{2}^\alpha} &\Rightarrow [H(\alpha', \beta, F)]_{\beta + \underline{2}^{\alpha'}}^{\underline{2}^\alpha} = [F_0]_{\beta + \underline{2}^{\alpha'}}^{\underline{2}^\alpha} \Rightarrow \\ [H(\alpha', \beta + \underline{2}^{\alpha'}, H(\alpha', \beta, F))]_{\beta + \underline{2}^{\alpha'} + \underline{1}}^{\underline{2}^\alpha} &= [F_0]_{\beta + \underline{2}^{\alpha'} + \underline{1}}^{\underline{2}^\alpha} \end{aligned}$$

For the limit case, assume $\alpha' \leq \alpha$ is a limit ordinal such that $\text{prop}H$ holds for all smaller ordinals. Recall that, according to Lemma 5.3.36 and putting $\alpha'' = p(\alpha', \beta', \beta)$, $\alpha'' < \alpha'$ and $\beta' < \beta + \underline{2}^{\alpha''}$ whenever $\beta' < \beta + \underline{2}^{\alpha'}$. Now assume $[F]_\beta^{\underline{2}^\alpha} = [F_0]_\beta^{\underline{2}^\alpha}$ and $\beta' < \beta + \underline{2}^{\alpha'}$, then $[H(\alpha'', \beta, F)]_{\beta + \underline{2}^{\alpha''}}^{\underline{2}^\alpha} = [F_0]_{\beta + \underline{2}^{\alpha''}}^{\underline{2}^\alpha}$ by $\text{prop}H(\alpha'')$, so $H(\alpha'', \beta, F, \beta') = F_0(\beta')$. It follows that $[H(\alpha', \beta, F)]_{\beta + \underline{2}^{\alpha'}}^{\underline{2}^\alpha} = [F_0]_{\beta + \underline{2}^{\alpha'}}^{\underline{2}^\alpha}$. ■

5.3.46. LEMMA. *Every functional definable by the schema TR_α for some ordinal $\alpha < \epsilon_0$ is T -definable.*

PROOF. Analogous to the proof of Lemma 5.3.40. ■

Lemma 5.3.46 establishes ϵ_0 as a lower bound for the computational strength of Gödel's \mathcal{T} . It can be shown that ϵ_0 is a sharp bound for \mathcal{T} , see Tait [1965], Howard [1970] and Schwichtenberg [1975]. In the next section we will introduce Spector's system \mathcal{B} . It is also known that \mathcal{B} is much stronger than \mathcal{T} , lower bounds have been established for subsystems of \mathcal{B} , but the computational strength of \mathcal{B} in terms of ordinals remains one of the great open problems in this field.

5.4. Spector's system \mathcal{B} : bar recursion

Spector [1962] extends Gödel's \mathcal{T} with a definition schema called bar recursion.³ Bar recursion is a principle of definition by recursion on a well-founded tree of finite sequences of functionals of the same type. For the formulation of bar recursion we need finite sequences of functionals of type A . These can conveniently be encoded by pairs consisting

³For the purpose of characterizing the provably recursive functions of analysis, yielding a consistency proof of analysis.

of a functional of type \mathbf{N} and one of type $\mathbf{N} \rightarrow A$. The intuition is that x, C encode the sequence of the first x values of C , that is, $C(0), \dots, C(x-1)$. We need auxiliary functionals to extend finite sequences of any type. A convenient choice is the primitive recursive functional $\text{Ext}_A : (\mathbf{N} \rightarrow A) \rightarrow \mathbf{N} \rightarrow A \rightarrow \mathbf{N} \rightarrow A$ defined by:

$$\text{Ext}_A(C, x, A, y) = \begin{cases} C(y) & \text{if } y < x, \\ A & \text{otherwise.} \end{cases}$$

We shall often omit the type subscript in Ext_A , and abbreviate $\text{Ext}(C, x, A)$ by $C *_x A$ and $\text{Ext}(C, x, 0^A)$ by $[C]_x$. We are now in a position to formulate the schema of bar recursion:⁴

$$\varphi(x, C) = \begin{cases} G(x, C) & \text{if } Y[C]_x < x, \\ H(\lambda a^A. \varphi(x+1, C *_x a), x, C) & \text{otherwise.} \end{cases}$$

The case distinction is governed by $Y[C]_x < x$, the so-called *bar condition*. The base case of bar recursion is the case in which the bar condition holds. In the other case φ is recursively called on all extensions of the (encoded) finite sequence.

A key feature of bar recursion is its proof theoretic strength as established by Spector[1962]. As a consequence, some properties of bar recursion are hard to prove, such as SN and the existence of a model. As an example of the latter phenomenon we shall show that the full set theoretic model of Gödel's \mathcal{T} is not a model of bar recursion.

Consider functionals Y, G, H defined by $G(x, C) = 0$, $H(Z, x, C) = 1 + Z(1)$ and

$$Y(F) = \begin{cases} 0 & \text{if } F(m) = 1 \text{ for all } m, \\ n & \text{otherwise, where } n = \min\{m \mid F(m) \neq 1\}. \end{cases}$$

Let $1^{\mathbf{N} \rightarrow \mathbf{N}}$ be the constant 1 function. The crux of Y is that $Y[1^{\mathbf{N} \rightarrow \mathbf{N}}]_x = x$ for all x , so that the bar recursion is not well-founded. We calculate

$$\varphi(0, 1^{\mathbf{N} \rightarrow \mathbf{N}}) = 1 + \varphi(1, 1^{\mathbf{N} \rightarrow \mathbf{N}}) = \dots = n + \varphi(n, 1^{\mathbf{N} \rightarrow \mathbf{N}}) = \dots$$

which shows that φ is not well-defined.

Syntax of λ_B

In this section we formalize Spector's \mathcal{B} as an extension of Gödel's \mathcal{T} called λ_B .

5.4.1. DEFINITION. The *types* of λ_B are the types of λ_T . We use $A^{\mathbf{N}}$ as shorthand for the type $\mathbf{N} \rightarrow A$. The *terms* of λ_B are obtained by adding constants

$$\begin{aligned} \mathbf{B}(A, B) & : (A^{\mathbf{N}} \rightarrow \mathbf{N}) \rightarrow (\mathbf{N} \rightarrow A^{\mathbf{N}} \rightarrow B) \rightarrow ((A \rightarrow B) \rightarrow \mathbf{N} \rightarrow A^{\mathbf{N}} \rightarrow B) \rightarrow \mathbf{N} \rightarrow A^{\mathbf{N}} \rightarrow B \\ \mathbf{B}_{A,B}^c & : (A^{\mathbf{N}} \rightarrow \mathbf{N}) \rightarrow (\mathbf{N} \rightarrow A^{\mathbf{N}} \rightarrow B) \rightarrow ((A \rightarrow B) \rightarrow \mathbf{N} \rightarrow A^{\mathbf{N}} \rightarrow B) \rightarrow \mathbf{N} \rightarrow A^{\mathbf{N}} \rightarrow \mathbf{N} \rightarrow B \end{aligned}$$

⁴Spector uses $[C]_x$ instead of C as last argument of G and H . Both formulations are easily seen to be equivalent since they are schematic in G, H (as well as in Y).

for all types A, B to the constants of λ_T . The set of (closed) terms of λ_B (of type A) is denoted with $\Lambda_B^{(0)}(A)$. The *formulas* of λ_B are equations between terms of λ_B (of the same type). The *theory* of λ_B extends the theory of λ_T with the above schema of bar recursion (with φ abbreviating BYGH). The *reduction relation* \rightarrow_B of λ_B extends \rightarrow_T by adding the following (schematic) rules for the constants B, B^c (omitting type annotations A, B):

$$\begin{aligned} \text{BYGHXC} &\rightarrow_B B^c\text{YGHXC}(X \dot{-} [C]_X) \\ B^c\text{YGHXC}(S^+N) &\rightarrow_B \text{GXC} \\ B^c\text{YGHXC}0 &\rightarrow_B H(\lambda a.\text{BYGH}(S^+X)(C *_X a))\text{XC} \end{aligned}$$

The reduction rules for B, B^c require some explanation. First observe that $x \dot{-} Y[C]_x = 0$ iff $Y[C]_x \geq x$, so that testing $x \dot{-} Y[C]_x = 0$ amounts to evaluating the (negation) of the bar condition. Consider a primitive recursive functional If_B satisfying $\text{If}_B 0 M_1 M_0 = M_0$ and $\text{If}_B(S^+P)M_1 M_0 = M_1$. A straightforward translation of the definition schema of bar recursion into a reduction rule:

$$\text{BYGHXC} \rightarrow \text{If}(X \dot{-} [C]_X)(\text{GXC})(H(\lambda x.\text{BYGH}(S^+X)(C *_X x))\text{XC})$$

would lead to infinite reduction sequences (the innermost B can be reduced again and again). It turns out to be necessary to evaluate the boolean first. This has been achieved by the interplay between B and B^c .

Theorem 5.3.9, Lemma 5.3.10 and Theorem 5.3.12 carry over from λ_T to λ_B with proofs that are easy generalizations. We now prove SN for λ_B and then obtain CR for λ_B using Newman's Lemma 5.3.14. The proof of SN for λ_B is considerably more difficult than for λ_T , which reflects the metamathematical fact that λ_B corresponds to analysis (see Spector [1962]), whereas λ_T corresponds to arithmetic. We start with defining hereditary finiteness for *sets* of terms, an analytical notion which plays a similar role as the arithmetical notion of computability for *terms* in the case of λ_T . Both are logical relations in the sense of Section 3.3, although hereditary finiteness is defined on the power set. Both computability and hereditary finiteness strengthen the notion of strong normalization, both are shown to hold by induction on terms. For metamathematical reasons, notably the consistency of analysis, it should not come as a surprise that we need an analytical induction loading in the case of λ_B .

5.4.2. DEFINITION. For every set $\mathcal{X} \subseteq \Lambda_B$, let $nf(\mathcal{X})$ denote the set of normal forms of terms from \mathcal{X} . For all $\mathcal{X} \subseteq \Lambda_B(A \rightarrow B)$ and $\mathcal{Y} \subseteq \Lambda_B(A)$, let $\mathcal{X}\mathcal{Y}$ denote the set of all applications of terms from \mathcal{X} to terms from \mathcal{Y} . Furthermore, if $M(x_1, \dots, x_k)$ is a term with free variables x_1, \dots, x_k , and $\mathcal{X}_1, \dots, \mathcal{X}_k$ are sets of terms such that every term from \mathcal{X}_i has the same type as x_i ($1 \leq i \leq k$), then we denote the set of all corresponding substitution instances by $M(\mathcal{X}_1, \dots, \mathcal{X}_k)$.

By induction on the type A we define $\mathcal{X} \in \mathcal{HF}_A$, expressing that the set \mathcal{X} of closed terms of type A is *hereditarily finite*.

$$\begin{aligned} \mathcal{X} \in \mathcal{H}\mathcal{F}_{\mathbf{N}} &\iff \mathcal{X} \subseteq \Lambda_B^\emptyset(\mathbf{N}) \cap \text{SN and } nf(\mathcal{X}) \text{ is finite} \\ \mathcal{X} \in \mathcal{H}\mathcal{F}_{A \rightarrow B} &\iff \mathcal{X} \subseteq \Lambda_B^\emptyset(A \rightarrow B) \text{ and } \mathcal{X}\vec{y} \in \mathcal{H}\mathcal{F}_B \text{ whenever } \vec{y} \in \mathcal{H}\mathcal{F}_A \end{aligned}$$

A closed term M is hereditarily finite, denoted by $M \in \text{HF}^0$, if $\{M\} \in \mathcal{H}\mathcal{F}$. If $M(x_1, \dots, x_k)$ is a term all whose free variables occur among x_1, \dots, x_k , then $M(x_1, \dots, x_k)$ is hereditarily finite, denoted by $M(x_1, \dots, x_k) \in \text{HF}$, if $M(\mathcal{X}_1, \dots, \mathcal{X}_k)$ is hereditarily finite for all $\mathcal{X}_i \in \mathcal{H}\mathcal{F}$ of appropriate types ($1 \leq i \leq k$).

Some basic properties of hereditary finiteness are summarized in the following lemmas. We use vector notation to abbreviate sequences of arguments of appropriate types both for terms and for sets of terms. For example, $M\vec{N}$ abbreviates $MN_1 \dots N_k$ and $\mathcal{X}\vec{y}$ stands for $\mathcal{X}y_1 \dots y_k$. The first two lemmas are instrumental for proving hereditary finiteness.

5.4.3. LEMMA. $\mathcal{X} \subseteq \Lambda_B^\emptyset(A_1 \rightarrow \dots \rightarrow A_n \rightarrow \mathbf{N})$ is hereditarily finite if and only if $\mathcal{X}\vec{y} \in \mathcal{H}\mathcal{F}_{\mathbf{N}}$ for all $y_1 \in \mathcal{H}\mathcal{F}_{A_1}, \dots, y_n \in \mathcal{H}\mathcal{F}_{A_n}$.

PROOF. By induction on n , applying Definition 5.4.2. ■

5.4.4. DEFINITION. Given two sets of terms $\mathcal{X}, \mathcal{X}' \subseteq \Lambda_B^\emptyset$, we say that \mathcal{X} is *adfluent* with \mathcal{X}' if every maximal reduction sequence starting in \mathcal{X} passes through a reduct of a term in \mathcal{X}' . Let $A \equiv A_1 \rightarrow \dots \rightarrow A_n \rightarrow \mathbf{N}$ with $n \geq 0$ and let $\mathcal{X}, \mathcal{X}' \subseteq \Lambda_B^\emptyset(A)$. We say that \mathcal{X} is *hereditarily adfluent* with \mathcal{X}' if $\mathcal{X}\vec{y}$ is adfluent with $\mathcal{X}'\vec{y}$, for all $y_1 \in \mathcal{H}\mathcal{F}_{A_1}, \dots, y_n \in \mathcal{H}\mathcal{F}_{A_n}$.

5.4.5. LEMMA. Let $\mathcal{X}, \mathcal{X}' \subseteq \Lambda_B^\emptyset(A)$ be such that \mathcal{X} is hereditarily adfluent with \mathcal{X}' . Then $\mathcal{X} \in \mathcal{H}\mathcal{F}_A$ whenever $\mathcal{X}' \in \mathcal{H}\mathcal{F}_A$.

PROOF. Let conditions be as in the lemma and assume $\mathcal{X}' \in \mathcal{H}\mathcal{F}_A$ with $A \equiv A_1 \rightarrow \dots \rightarrow A_n \rightarrow \mathbf{N}$. Let $y_1 \in \mathcal{H}\mathcal{F}_{A_1}, \dots, y_n \in \mathcal{H}\mathcal{F}_{A_n}$, then $\mathcal{X}\vec{y}$ is adfluent with $\mathcal{X}'\vec{y}$. It follows that $\mathcal{X}\vec{y} \subseteq \text{SN}$ since $\mathcal{X}'\vec{y} \subseteq \text{SN}$ and $nf(\mathcal{X}\vec{y}) \subseteq nf(\mathcal{X}'\vec{y})$, so $nf(\mathcal{X}\vec{y})$ is finite since $nf(\mathcal{X}'\vec{y})$ is. Applying Lemma 5.4.3 we obtain $\mathcal{X} \in \mathcal{H}\mathcal{F}_A$. ■

Note that the above lemma holds in particular if $n = 0$, that is, if $A \equiv \mathbf{N}$.

5.4.6. LEMMA. For every type A we have: (i) $\text{HF}_A \subseteq \text{SN}$ and (ii) $0^A \in \text{HF}_A$.

PROOF. We prove (ii) and (iii) $\text{HF}_A^0 \subseteq \text{SN}$ by simultaneous induction on A . Then (i) follows immediately. Obviously, $0 \in \text{HF}_{\mathbf{N}}$ and $\text{HF}_{\mathbf{N}}^0 \subseteq \text{SN}$. For the induction step $A \rightarrow B$, assume (ii) and (iii) hold for all smaller types. If $M \in \text{HF}_{A \rightarrow B}^0$, then by the induction hypothesis (ii) $0^A \in \text{HF}_A^0$, so $M0^A \in \text{HF}_B^0$, so $M0^A$ is SN by the induction hypothesis (iii), and hence M is SN. Recall that $0^{A \rightarrow B} \equiv \lambda x^A. 0^B$. We use Lemma 5.4.3 to prove the induction step for (ii). Let $\mathcal{X} \in \mathcal{H}\mathcal{F}_A$, then $\mathcal{X} \subseteq \text{SN}$ by the induction hypothesis. It follows that $0^{A \rightarrow B}\mathcal{X}$ is hereditarily adfluent with 0^B . By the induction hypothesis we have $0^B \in \text{HF}_B$, so $0^{A \rightarrow B}\mathcal{X} \in \mathcal{H}\mathcal{F}_B$ by Lemma 5.4.5. It follows that $0^{A \rightarrow B} \in \text{HF}_{A \rightarrow B}$. ■

The proofs of the following three lemmas are left to the reader.

5.4.7. LEMMA. *Every reduct of a hereditarily finite term is hereditarily finite.*

5.4.8. LEMMA. *Subsets of hereditarily finite sets of terms are hereditarily finite.*

In particular elements of a hereditarily finite set are hereditarily finite.

5.4.9. LEMMA. *Finite unions of hereditarily finite sets are hereditarily finite.*

In this connection of course only unions of the same type make sense.

5.4.10. EXERCISE. Prove the above three lemmas.

5.4.11. LEMMA. *The hereditarily finite terms are closed under application.*

PROOF. Immediate from Definition 5.4.2. ■

5.4.12. LEMMA. *The hereditarily finite terms are closed under lambda abstraction.*

PROOF. Let $M(x, x_1, \dots, x_k) \in \text{HF}$ be a term all whose free variables occur among x, x_1, \dots, x_k . We have to prove $\lambda x.M(x, x_1, \dots, x_k) \in \text{HF}$, that is,

$$\lambda x.M(x, \mathcal{X}_1, \dots, \mathcal{X}_k) \in \mathcal{HF}$$

for given $\vec{\mathcal{X}} = \mathcal{X}_1, \dots, \mathcal{X}_k \in \mathcal{HF}$ of appropriate types. Let $\mathcal{X} \in \mathcal{HF}$ be of the same type as the variable x , so $\mathcal{X} \subseteq \text{SN}$ by Lemma 5.4.6. We also have $M(x, \vec{\mathcal{X}}) \subseteq \text{SN}$ by the assumption on M and Lemma 5.4.6. It follows that $(\lambda x.M(x, \vec{\mathcal{X}}))\mathcal{X}$ is hereditarily adfluent with $M(\mathcal{X}, \vec{\mathcal{X}})$. Again by the assumption on M we have that $M(\mathcal{X}, \vec{\mathcal{X}}) \in \mathcal{HF}$, so that $(\lambda x.M(x, \vec{\mathcal{X}}))\mathcal{X} \in \mathcal{HF}$ by Lemma 5.4.5. We conclude that $\lambda x.M(x, \vec{\mathcal{X}}) \in \mathcal{HF}$, so $\lambda x.M(x, x_1, \dots, x_k) \in \text{HF}$. ■

5.4.13. THEOREM. *Every term of λ_T is hereditarily finite.*

PROOF. By Lemma 5.4.11 and Lemma 5.4.12, the hereditarily finite terms are closed under application and lambda abstraction, so it suffices to show that the constants and the variables are hereditarily finite. Variables and the constant 0 are obviously hereditarily finite. Regarding S^+ , let $\mathcal{X} \in \mathcal{HF}_{\mathbf{N}}$, then $S^+\mathcal{X} \subseteq \Lambda_B^\emptyset(\mathbf{N}) \cap \text{SN}$ and $nf(S^+\mathcal{X})$ is finite since $nf(\mathcal{X})$ is finite. Hence $S^+\mathcal{X} \in \mathcal{HF}_{\mathbf{N}}$, so S^+ is hereditarily finite. It remains to prove that the constants R_A are hereditarily finite. Let $\mathcal{M}, \mathcal{N}, \mathcal{X} \in \mathcal{HF}$ be of appropriate types and consider $R_A\mathcal{M}\mathcal{N}\mathcal{X}$. We have in particular $\mathcal{X} \in \mathcal{HF}_{\mathbf{N}}$, so $nf(\mathcal{X})$ is finite, and the proof of $R_A\mathcal{M}\mathcal{N}\mathcal{X} \in \mathcal{HF}$ goes by induction on the largest numeral in $nf(\mathcal{X})$. If $nf(\mathcal{X}) = \{0\}$, then $R_A\mathcal{M}\mathcal{N}\mathcal{X}$ is hereditarily adfluent with \mathcal{M} . Since $\mathcal{M} \in \mathcal{HF}$ we can apply Lemma 5.4.5 to obtain $R_A\mathcal{M}\mathcal{N}\mathcal{X} \in \mathcal{HF}$. For the induction step, assume $R_A\mathcal{M}\mathcal{N}\mathcal{X}' \in \mathcal{HF}$ for all $\mathcal{X}' \in \mathcal{HF}$ such that the largest numeral in $nf(\mathcal{X}')$ is n . Let, for some $\mathcal{X} \in \mathcal{HF}$, the largest numeral in $nf(\mathcal{X})$ be S^+n . Define

$$\mathcal{X}' = \{X \mid S^+X \text{ is a reduct of a term in } \mathcal{X}\}$$

Then $\mathcal{X}' \in \mathcal{HF}$ since $\mathcal{X} \in \mathcal{HF}$, and the largest numeral in $nf(\mathcal{X}')$ is n . It follows by the induction hypothesis that $R_A\mathcal{M}\mathcal{N}\mathcal{X}' \in \mathcal{HF}$, so $\mathcal{N}(R_A\mathcal{M}\mathcal{N}\mathcal{X}')\mathcal{X}' \in \mathcal{HF}$ and hence $\mathcal{N}(R_A\mathcal{M}\mathcal{N}\mathcal{X}')\mathcal{X}' \cup \mathcal{M} \in \mathcal{HF}$ by Lemma 5.4.11, 5.4.9. We have that $R_A\mathcal{M}\mathcal{N}\mathcal{X}$ is hereditarily adfluent with $\mathcal{N}(R_A\mathcal{M}\mathcal{N}\mathcal{X}')\mathcal{X}' \cup \mathcal{M}$ so $R_A\mathcal{M}\mathcal{N}\mathcal{X} \in \mathcal{HF}$ by Lemma 5.4.5. This completes the induction step. ■

Before we can prove that \mathcal{B} is hereditarily finite we need the following lemma.

5.4.14. LEMMA. *Let $\mathcal{Y}, \mathcal{G}, \mathcal{H}, \mathcal{X}, \mathcal{C} \in \mathcal{HF}$ be of appropriate type. Then*

$$\mathcal{B}\mathcal{Y}\mathcal{G}\mathcal{H}\mathcal{X}\mathcal{C} \in \mathcal{HF},$$

*whenever $\mathcal{B}\mathcal{Y}\mathcal{G}\mathcal{H}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} \mathcal{A}) \in \mathcal{HF}$ for all $\mathcal{A} \in \mathcal{HF}$ of appropriate type.*

PROOF. Let conditions be as above. Abbreviate $\mathcal{B}\mathcal{Y}\mathcal{G}\mathcal{H}$ by \mathcal{B} and $\mathcal{B}^c\mathcal{Y}\mathcal{G}\mathcal{H}$ by \mathcal{B}^c . Assume $\mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} \mathcal{A}) \in \mathcal{HF}$ for all $\mathcal{A} \in \mathcal{HF}$. Below we will frequently and implicitly use that $\div, *, []$ are primitive recursive and hence hereditarily finite, and that hereditary finiteness is closed under application. Since hereditarily finite terms are strongly normalizable, we have that $\mathcal{B}\mathcal{X}\mathcal{C}$ is hereditarily adfluent with $\mathcal{B}^c\mathcal{X}\mathcal{C}(\mathcal{X} \div \mathcal{Y}[\mathcal{C}]_{\mathcal{X}})$, and hence with $\mathcal{G}\mathcal{C}\mathcal{X} \cup \mathcal{H}(\lambda a. \mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} a))\mathcal{C}\mathcal{X}$. It suffices to show that the latter set is in \mathcal{HF} . We have $\mathcal{G}\mathcal{C}\mathcal{X} \in \mathcal{HF}$, so by Lemma 5.4.9 the union is hereditarily finite if $\mathcal{H}(\lambda a. \mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} a))\mathcal{C}\mathcal{X}$ is. It suffices that $\lambda a. \mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} a) \in \mathcal{HF}$, and this will follow by the assumption above. We first observe that $\{0^A\} \in \mathcal{HF}$ so $\mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} \{0^A\}) \in \mathcal{HF}$ and hence $\mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} a) \subseteq \text{SN}$ by Lemma 5.4.6. Let $\mathcal{A} \in \mathcal{HF}$. Since $\mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} a), \mathcal{A} \subseteq \text{SN}$ we have that $(\lambda a. \mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} a))\mathcal{A}$ is adfluent with $\mathcal{B}(\mathcal{S}^+\mathcal{X})(\mathcal{C} *_{\mathcal{X}} \mathcal{A}) \in \mathcal{HF}$ and hence hereditarily finite itself by Lemma 5.4.5. ■

We now have arrived at the crucial step, where not only the language of analysis will be used, but also the axiom of dependent choice in combination with classical logic. We will reason by contradiction. Suppose \mathcal{B} is not hereditarily finite. Then there are hereditarily finite $\mathcal{Y}, \mathcal{G}, \mathcal{H}, \mathcal{X}$ and \mathcal{C} such that $\mathcal{B}\mathcal{Y}\mathcal{G}\mathcal{H}\mathcal{X}\mathcal{C}$ is not hereditarily finite. We introduce the following abbreviations: \mathcal{B} for $\mathcal{B}\mathcal{Y}\mathcal{G}\mathcal{H}$ and $\mathcal{X}+n$ for $\mathcal{S}^+(\dots(\mathcal{S}^+\mathcal{X})\dots)$ (n times \mathcal{S}^+). By Lemma 5.4.14, there exists $\mathcal{U} \in \mathcal{HF}$ such that $\mathcal{B}(\mathcal{X}+1)(\mathcal{C} *_{\mathcal{X}} \mathcal{U})$ is not hereditarily finite. Hence again by Lemma 5.4.14, there exists $\mathcal{V} \in \mathcal{HF}$ such that $\mathcal{B}(\mathcal{X}+2)((\mathcal{C} *_{\mathcal{X}} \mathcal{U}) *_{\mathcal{X}+1} \mathcal{V})$ is not hereditarily finite. Using dependent choice, let

$$\mathcal{D} = \mathcal{C} \cup (\mathcal{C} *_{\mathcal{X}} \mathcal{U}) \cup ((\mathcal{C} *_{\mathcal{X}} \mathcal{U}) *_{\mathcal{X}+1} \mathcal{V}) \cup \dots$$

be the infinite union of the sets obtained by iterating the argument above. Note that all sets in the infinite union are hereditarily finite of type $A^{\mathbb{N}}$. Since the union is infinite, it does not follow from Lemma 5.4.9 that \mathcal{D} itself is hereditarily finite. However, since \mathcal{D} has been built up from terms of type $A^{\mathbb{N}}$ having longer and longer initial segments in common we will nevertheless be able to prove that $\mathcal{D} \in \mathcal{HF}$. Then we will arrive at a contradiction, since $\mathcal{Y}\mathcal{D} \in \mathcal{HF}$ implies that \mathcal{Y} is bounded on \mathcal{D} , so that the bar condition is satisfied after finitely many steps, which conflicts with the construction process.

5.4.15. LEMMA. *The set \mathcal{D} constructed above is hereditarily finite.*

PROOF. Let $\mathcal{N}, \vec{\mathcal{Z}} \in \mathcal{HF}$ be of appropriate type, that is, \mathcal{N} of type \mathbb{N} and $\vec{\mathcal{Z}}$ such that $\mathcal{D}\mathcal{N}\vec{\mathcal{Z}}$ is of type \mathbb{N} . We have to show $\mathcal{D}\mathcal{N}\vec{\mathcal{Z}} \in \mathcal{HF}$. Since all elements of \mathcal{D} are hereditarily finite we have $\mathcal{D}\mathcal{N}\vec{\mathcal{Z}} \subseteq \text{SN}$. By an easy generalization of Theorem 5.3.12 we have WCR

for λ_B , so by Newman's Lemma 5.3.14 we have $\mathcal{DN}\vec{Z} \subseteq \mathcal{CR}$. Since $\mathcal{N} \in \mathcal{HF}$ it follows that $nf(\mathcal{N})$ is finite, say $nf(\mathcal{N}) \subseteq \{0, \dots, n\}$ for n large enough. It remains to show that $nf(\mathcal{DN}\vec{Z})$ is finite. Since all terms in $\mathcal{DN}\vec{Z}$ are \mathcal{CR} , their normal forms are unique. As a consequence we may apply a leftmost innermost reduction strategy to any term $\mathcal{DN}\vec{Z} \in \mathcal{DN}\vec{Z}$. At this point it might be helpful to remind the reader of the intended meaning of $*$: $C *_x A$ represents the finite sequence $C0, \dots, C(x-1), A$. More formally,

$$(C *_x A)y = \begin{cases} C(y) & \text{if } y < x, \\ A & \text{otherwise.} \end{cases}$$

With this in mind it is easily seen that $nf(\mathcal{DN}\vec{Z})$ is a subset of $nf(\mathcal{D}_n\mathcal{N}\vec{Z})$, with

$$\mathcal{D}_n = \mathcal{C} \cup (\mathcal{C} *_x \mathcal{U}) \cup ((\mathcal{C} *_x \mathcal{U}) *_x \mathcal{V}) \cup \dots \cup (\dots (\mathcal{C} *_x \mathcal{U}) * \dots *_x \mathcal{W})$$

a finite initial part of the infinite union \mathcal{D} . The set $nf(\mathcal{D}_n\mathcal{N}\vec{Z})$ is finite since the union is finite and all sets involved are in \mathcal{HF} . Hence \mathcal{D} is hereditarily finite by Lemma 5.4.3. ■

Since \mathcal{D} is hereditarily finite, it follows that $nf(\mathcal{YD})$ is finite. Let k be larger than any numeral in $nf(\mathcal{YD})$. Consider

$$\mathcal{B}_k = \mathcal{B}(\mathcal{X}+k)(\dots (\mathcal{C} *_x \mathcal{U}) * \dots *_x \mathcal{W}')$$

as obtained in the construction above, iterating Lemma 5.4.14, hence not hereditarily finite. Since k is a strict upper bound of $nf(\mathcal{YD})$ it follows that the set $nf((\mathcal{X}+k) \div \mathcal{YD})$ consists of numerals greater than 0, so that \mathcal{B}_k is hereditarily adfluent with $\mathcal{G}(\mathcal{X}+k)\mathcal{D}$. The latter set is hereditarily finite since it is an application of hereditarily finite sets (use Lemma 5.4.15). Hence \mathcal{B}_k is hereditarily finite by Lemma 5.4.5, which yields a plain contradiction.

By this contradiction, \mathcal{B} must be hereditarily finite, and so is \mathcal{B}^c , which follows by inspection of the reduction rules. As a consequence we obtain the main theorem of this section.

5.4.16. THEOREM. *Every bar recursive term is hereditarily finite.*

5.4.17. COROLLARY. *Every bar recursive term is strongly normalizable.*

5.4.18. REMARK. The first normalization result for bar recursion is due to Tait [1971], who proves WN for λ_B . Vogel [1976] strengthens Tait's result to SN, essentially by introducing \mathcal{B}^c and by enforcing every \mathcal{B} -redex to reduce via \mathcal{B}^c . Both Tait and Vogel use infinite terms. The proof above is based on Bezem [1985] and avoids infinite terms by using the notion of hereditary finiteness, which is a syntactic version of Howard's compactness of functionals of finite type, see Troelstra [1973], Section 2.8.6.

If one considers λ_B also with η -reduction, then the above results can also be obtained in a similar way as for λ_T with η -reduction.

Semantics of λ_B

In this section we give some interpretations of bar recursion.

5.4.19. DEFINITION. A model of λ_B is a model of λ_T with interpretations of the constants $B_{A,B}$ and $B_{A,B}^c$ for all A, B , such that the rules for these constants can be interpreted as valid equations. In particular we have then that the schema of bar recursion is valid, with $\llbracket \varphi \rrbracket = \llbracket BYGH \rrbracket$.

We have seen at the beginning of this section that the full set theoretic model of Gödel's \mathcal{T} is not a model of bar recursion, due to the existence of functionals (such as Y unbounded on binary functions) for which the bar recursion is not well-founded. Designing a model of λ_B amounts to ruling out such functionals, while maintaining the necessary closure properties. There are various solutions to this problem. The simplest solution is to take the closed terms modulo convertibility, which form a model by CR and SN. However, interpreting terms (almost) by themselves does not explain very much. In Exercise 5.4.24 the reader is asked to prove that the closed term model is extensional. An interesting model is obtained by using continuity in the form of the Kleene [1959a] and Kreisel [1959] continuous functionals. Continuity is on one hand a structural property of bar recursive terms, since they can use only finitely many information about their arguments, and on the other hand ensures that bar recursion is well-founded, since a continuous Y becomes eventually constant on increasing initial segments $[C]_x$. In Exercise 5.4.23 the reader is asked to elaborate this model in detail. Refinements can be obtained by considering notions of computability on the continuous functionals, such as Kleene's [1959b] S1-S9 recursive functionals. Computability alone, without uniform continuity on all binary functions, does not yield a model of bar recursion, see Exercise 5.4.22. The model of bar recursion we will elaborate in the next paragraphs is based on the same idea as the proof of strong normalization in the previous section. Here we consider the notion of hereditary finiteness semantically instead of syntactically. The intuition is that the set of increasing initial segments is hereditarily finite, so that any hereditarily finite functional Y is bounded on that set, and hence the bar recursion is well-founded.

5.4.20. DEFINITION (hereditarily finite functionals). Recall the full type structure over the natural numbers: $\mathbb{N}_{\mathbb{N}} = \mathbb{N}$ and $\mathbb{N}_{A \rightarrow B} = \mathbb{N}_A \rightarrow \mathbb{N}_B$. A set $\mathcal{X} \subseteq \mathbb{N}_{\mathbb{N}}$ is hereditarily finite if \mathcal{X} is finite. A set $\mathcal{X} \subseteq \mathbb{N}_{A \rightarrow B}$ is hereditarily finite if $\mathcal{X}\mathcal{Y} \subseteq \mathbb{N}_B$ is hereditarily finite for every hereditarily finite $\mathcal{Y} \subseteq \mathbb{N}_A$. Here and below, $\mathcal{X}\mathcal{Y}$ denotes the set of all results that can be obtained by applying functionals from \mathcal{X} to functionals from \mathcal{Y} . A functional F is hereditarily finite if the singleton set $\{F\}$ is hereditarily finite. Let $\mathcal{H}\mathcal{F}$ be the substructure of the full type structure consisting of all hereditarily finite functionals.

The proof that $\mathcal{H}\mathcal{F}$ is a model of λ_B has much in common with the proof that λ_B is SN from the previous paragraph. The essential step is that the interpretation of the bar recursor is hereditarily finite. This requires the following semantic version of Lemma 5.4.14:

5.4.21. LEMMA. Let $\mathcal{Y}, \mathcal{G}, \mathcal{H}, \mathcal{X}, \mathcal{C}$ be hereditarily finite sets of appropriate type. Then $\llbracket \mathbf{B} \rrbracket \mathcal{Y} \mathcal{G} \mathcal{H} \mathcal{X} \mathcal{C}$ is well defined and hereditarily finite whenever $\llbracket \mathbf{B} \rrbracket \mathcal{Y} \mathcal{G} \mathcal{H} (\mathcal{X} + 1) (\mathcal{C} *_{\mathcal{X}} \mathcal{A})$ is so for all hereditarily finite \mathcal{A} of appropriate type.

The proof proceeds by iterating this lemma in the same way as how the SN proof proceeds after Lemma 5.4.14. The set of longer and longer initial sequences with elements taken from hereditarily finite sets (cf. the set \mathcal{D} in Lemma 5.4.15) is hereditarily finite itself. As a consequence, the bar recursion must be well-founded when the set \mathcal{Y} is also hereditarily finite. It follows that the interpretation of the bar recursor is well-defined and hereditarily finite.

5.4.22. EXERCISE. This exercise shows that *HEO* is *not* a model for bar recursion. Recall that $*$ stands for partial recursive function application. Consider functionals Y, G, H defined by $G(x, C) = 0$, $H(Z, x, C) = 1 + Z(0) + Z(1)$ and $Y(F)$ is the smallest number n such that $i * i$ converges in less than n steps for some $i < n$ and, moreover, $i * i = 0$ if and only if $F(i) = 0$ does *not* hold. The crux of the definition of Y is that no *total* recursive function F can distinguish between $i * i = 0$ and $i * i > 0$ for *all* i with $i * i \downarrow$. But for any finite number of such i 's we do have a total recursive function making the correct distinctions. This implies that Y , although continuous and well-defined on all total recursive functions, is not uniformly continuous and not bounded on total recursive binary functions. Show that all functionals involved can be represented in *HEO* and that the latter model of λ_T is not a model of λ_B .

5.4.23. EXERCISE. This exercise introduces the continuous functionals, Kleene [1959a]. Define for $f, g \in \mathbb{N} \rightarrow \mathbb{N}$ the (partial) application of f to g by $f(g) = f(\overline{g}n) - 1$, where n is the smallest number such that $f(\overline{g}n) > 0$, provided there is such n . If there is no such n , then $f * g$ is undefined. The idea is that f uses only finitely many information about g for determining the value of $f * g$ (if any). Define inductively for every type A a set \mathcal{C}_A together with an association relation between elements of $\mathbb{N} \rightarrow \mathbb{N}$ and elements of \mathcal{C}_A . For the base type we put $\mathcal{C}_{\mathbb{N}} = \mathbb{N}$ and let the constant functions be the associates of the corresponding natural numbers. For higher types we define that $f \in \mathbb{N} \rightarrow \mathbb{N}$ is an associate of $F \in \mathcal{C}_A \rightarrow \mathcal{C}_B$ if for any associate g of $G \in \mathcal{C}_A$ the function h defined by $h(n) = f(n:g)$ is an associate of $F(G) \in \mathcal{C}_B$. Here $n:g$ is shorthand for the function taking value n at 0 and value $g(k-1)$ for all $k > 0$. (Note that we have implicitly required that h is total.) Now $\mathcal{C}_{A \rightarrow B}$ is defined as the subset of those $F \in \mathcal{C}_A \rightarrow \mathcal{C}_B$ that have an associate. Show that \mathcal{C} is a model for bar recursion.

Following Troelstra [1973], Section 2.4.5 and 2.7.2, we define the following notion of hereditary extensional equality. We put $\approx_{\mathbb{N}}$ to be $=$, convertibility of closed terms in $\Lambda_B^{\emptyset}(\mathbb{N})$. For the type $A \equiv B \rightarrow B'$ we define $M \approx_A M'$ if and only if $M, M' \in \Lambda_B^{\emptyset}(A)$ and $MN \approx_{B'} M'N'$ for all N, N' such that $N \approx_B N'$. By (simultaneous) induction on A one shows easily that \approx_A is symmetric, transitive and partially reflexive, that is, $M \approx_A M$ holds whenever $M \approx_A N$ for some N .

The corresponding axiom of hereditary extensionality is simply stating that \approx_A is (totally) reflexive: $M \approx_A M$, schematic in $M \in \Lambda_B^\emptyset(A)$ and A . It follows from the next exercise that the closed term model is extensional.

5.4.24. EXERCISE. Show $M \approx M$ for any closed term $M \in \Lambda_B^\emptyset$. Hint: define a predicate $\text{Ext}(M(x_1, \dots, x_n))$ for any open term M with free variables among x_1, \dots, x_n by

$$M(X_1, \dots, X_n) \approx M(X'_1, \dots, X'_n)$$

for all $X_1, \dots, X_n, X'_1, \dots, X'_n \in \Lambda_B^\emptyset$ with $X_1 \approx X'_1, \dots, X_n \approx X'_n$. Then prove by induction on terms that Ext holds for any open term, so in particular for closed terms. For \mathbf{B} , prove first the lemma below.

5.4.25. LEMMA. For all $Y \approx Y', G \approx G', H \approx H', X \approx X', C \approx C'$, if

$$\text{BYGH}(S^+X)(C *_X A) \approx \text{BY}'G'H'(S^+X')(C' *_X A')$$

for all $A \approx A'$, then $\text{BYGHXC} \approx \text{BY}'G'H'X'C'$.

5.5. Platek's system \mathcal{Y} : fixed point recursion

Platek [1966] introduces a simply typed lambda calculus extended with fixed point combinators. Here we study Platek's system as an extension of Gödel's \mathcal{T} . An almost identical system is called PCF in Plotkin [1977].

A *fixed point combinator* is a functional Y of type $(A \rightarrow A) \rightarrow A$ such that YF is a fixed point of F , that is, $YF = F(YF)$, for every F of type $A \rightarrow A$. Fixed point combinators can be used to compute solutions to recursion equations. The only difference with the type-free lambda calculus is that here all terms are typed, including the fixed point combinators themselves.

As an example we consider the recursion equations of the schema of higher order primitive recursion in Gödel's system \mathcal{T} , Section 5.3. We can rephrase these equations as

$$RMNn = \text{If } n \ (N(RMN(n-1))(n-1))M,$$

where $\text{If } n M_1 M_0 = M_0$ if $n = 0$ and M_1 if $n > 0$. Hence we can write

$$\begin{aligned} RMN &= \lambda n. \text{If } n \ (N(RMN(n-1))(n-1))M \\ &= (\lambda f n. \text{If } n \ (N(f(n-1))(n-1))M)(RMN) \end{aligned}$$

This equation is of the form $YF = F(YF)$ with

$$F = \lambda f n. \text{If } n \ (N(f(n-1))(n-1))M$$

and $YF = RMN$. It is easy to see that YF satisfies the recursion equation for RMN *uniformly* in M, N . This shows that, given functionals If and a predecessor function (to compute $n-1$ in case $n > 0$), higher-order primitive recursion is definable by fixed point

recursion. However, for computing purposes it is convenient to have primitive recursors at hand. By a similar argument, one can show bar recursion to be definable by fixed point recursion.

In addition to the above argument we show that every partial recursive function can be defined by fixed point recursion, by giving a fixed point recursion for minimization. Let F be a given function. Define by fixed point recursion $G_F = \lambda n. \text{If } F(n) \text{ then } G_F(n) = F(n) \text{ else } G_F(n+1)$. Then we have $G_F(0) = 0$ if $F(0) = 0$, and $G_F(0) = G_F(1)$ otherwise. We have $G_F(1) = 1$ if $F(1) = 0$, and $G_F(1) = G_F(2)$ otherwise. By continuing this argument we see that

$$G_F(0) = \min\{n \mid F(n) = 0\},$$

that is, $G_F(0)$ computes the smallest n such that $F(n) = 0$, provided that such n exists. If there exists no n such that $F(n) = 0$, then $G_F(0)$ as well as $G_F(1), G_F(2), \dots$ are undefined. Given a function F of *two* arguments, minimization with respect to the second argument can now be obtained by the partial function $\lambda x. G_{F(x)}(0)$.

In the paragraph above we saw already that fixed point recursions may be indefinite: if F does not zero, then $G_F(0) = G_F(1) = G_F(2) = \dots$ does not lead to a definite value, although one could consistently assume G_F to be a constant function in this case. However, the situation is in general even worse: there is no natural number n that can consistently be assumed to be the fixed point of the successor function, that is, $n = Y(\lambda x. x + 1)$, since we cannot have $n = (\lambda x. x + 1)n = n + 1$. This is the price to be paid for a formalism that allows one to compute all partial recursive functions.

Syntax of λ_Y

In this section we formalize Platek's \mathcal{Y} as an extension of Gödel's \mathcal{T} called λ_Y .

5.5.1. DEFINITION. The *types* of λ_Y are the types of λ_T . The *terms* of λ_Y are obtained by adding constants

$$Y_A : (A \rightarrow A) \rightarrow A$$

for all types A to the constants of λ_T . The set of (closed) terms of λ_Y (of type A) is denoted with $\Lambda_Y^\emptyset(A)$. The *formulas* of λ_Y are equations between terms of λ_Y (of the same type). The *theory* of λ_Y extends the theory of λ_T with the schema $YF = F(YF)$ for all appropriate types. The *reduction relation* \rightarrow_Y of λ_Y extends \rightarrow_T by adding the following rule for the constants Y (omitting type annotations A):

$$Y \rightarrow_Y \lambda f. f(Yf)$$

The reduction rule for Y requires some explanation, as the rule $YF \rightarrow F(YF)$ seems simpler. However, with the latter rule we would have diverging reductions $\lambda f. Yf \rightarrow_\eta Y$ and $\lambda f. Yf \rightarrow_Y \lambda f. f(Yf)$ that cannot be made to converge, so that we would lose CR of \rightarrow_Y in combination with η -reduction.

The Church-Rosser property for λ_Y with β -reduction and with $\beta\eta$ -reduction can be proved by standard techniques from higher-order rewriting theory, for example, by using weak orthogonality, see Raamsdonk [1996].

5.5.2. EXERCISE. It is possible to define λ_Y as extension of λ_{ω}^o using Church numerals $c_n \equiv \lambda x^{\mathbb{N}} f^{\mathbb{N} \rightarrow \mathbb{N}}.f^n x$. Show that every partial recursive function is also definable in this version of λ_Y .

Although λ_Y has universal computational strength in the sense that all partial recursive functions can be computed, not every computational phenomenon can be represented. In the next section we shall see that λ_Y does not have enumerators. Moreover, λ_Y is inherently sequential. For example, λ_Y doesn't have a term P such that $PMN = 0$ if and only if $M = 0$ or $N = 0$. The problem is that M and N cannot be evaluated in parallel, and if the argument that is evaluated first happens to be undefined, then the outcome is undefined even if the other argument equals 0. For a detailed account of the so-called sequentiality of λ_Y , see Plotkin [1977].

Semantics of λ_Y

In this section we explore the semantics of λ_Y and give one model.

5.5.3. DEFINITION. A model of λ_Y is a model of λ_T with interpretations of the constants Y_A for all A , such that the rules for these constants can be interpreted as valid equations.

Models of λ_Y differ from those of λ_T, λ_B in that they have to deal with partiality. As we saw in the introduction of this section, no natural number n can consistently be assumed to be the fixed point of the successor function. Nevertheless, we want to interpret terms like YS^+ . The canonical way to do so is to add an element \perp to the natural numbers, representing undefined objects like the fixed point of the successor function. Let \mathbb{N}^\perp denote the set of natural numbers extended with \perp . Now higher types are interpreted as function spaces over \mathbb{N}^\perp . The basic intuition is that \perp contains less information than any natural number, and that functions and functionals give more informative output when the input becomes more informative. One way of formalizing these intuitions is by using partial orderings. We equip \mathbb{N}^\perp with the partial ordering \sqsubseteq such that $\perp \sqsubset n$ for all $n \in \mathbb{N}$. In order to be able to interpret Y , every function must have a fixed point. This requires some extra structure on the partial orderings, which can be formalized by the notion of complete partial ordering (cpo). The next lines bear some similarity to the introductory treatment of ordinals in Section 5.3. We call a set *directed* if it contains an upper bound for every two elements of it. Completeness of a partial ordering means that every directed set has a supremum. A function on cpo's is called *continuous* if it preserves suprema of directed sets. Every continuous function f of cpo's is monotone and has a least fixed point $\text{lfp}(f)$, being the supremum of the directed set enumerated by iterating f starting at \perp . The function lfp is itself continuous and serves as the interpretation of Y . We are now ready for the following definition.

5.5.4. DEFINITION. Define by induction $\mathbb{N}_\mathbb{N}^\perp = \mathbb{N}^\perp$ and $\mathbb{N}_{A \rightarrow B}^\perp$ is the set of all continuous mappings $\mathbb{N}_A^\perp \rightarrow \mathbb{N}_B^\perp$.

Given the fact that cpo's with continuous mappings form a cartesian closed category and that the successor, predecessor and conditional can be defined in a continuous way, the only essential step in the proof of the following lemma is to put $\llbracket Y \rrbracket = \text{lfp}$ for all appropriate types.

5.5.5. LEMMA. *The type structure of cpo's \mathbb{N}_A^\perp is a model for λ_Y .*

In fact, as the essential requirement is the existence of fixed points, we could have taken monotone instead of continuous mappings on cpo's. This option is elaborated in detail in van Draanen [1995].

5.6. Exercises

5.6.1. Prove proposition 5.2.9: for all types A one has $A \triangleleft_{SP} N_{\text{rank}(A)}$.

5.6.2. Let λ_P be λ_{\rightarrow}^o extended with a simple (not surjective) pairing. Show that theorem 5.2.42 does not hold for this theory. [Hint show that in this theory the equation $\lambda x o. \langle \pi_1 x, \pi_2 x \rangle = \lambda x o. x$ does not hold by constructing a counter model, but is nevertheless consistent.]

5.6.3. Does every model of λ_{SP} have the same first order theory?

5.6.4. (i) Show that if a pairing function $\langle \cdot, \cdot \rangle : o \rightarrow (o \rightarrow o)$ and projections $L, R : o \rightarrow o$ satisfying $L\langle x, y \rangle = x$ and $R\langle x, y \rangle = y$ are added to λ_{\rightarrow}^o , then for a non-trivial model \mathcal{M} one has (see 4.2)

$$\forall A \in \mathbb{T} \forall M, N \in \Lambda^\emptyset(A) [\mathcal{M} \models M = N \Rightarrow M =_{\beta\eta} N].$$

(ii) (Schwichtenberg and Berger [1991]) Show that for \mathcal{M} a model of λ_T one has (see 4.3)

$$\forall A \in \mathbb{T} \forall M, N \in \Lambda^\emptyset(A) [\mathcal{M} \models M = N \Rightarrow M =_{\beta\eta} N].$$

5.6.5. Show that $\mathcal{F}[x_1, \dots, x_n]$ for $n \geq 0$ does not have one generator. [Hint. Otherwise this monoid would be commutative, which is not the case.]

5.6.6. Show that $R \subseteq \Lambda^\emptyset(A) \times \Lambda^\emptyset(B)$ is equational iff

$$\exists M, N \in \Lambda^\emptyset(A \rightarrow B \rightarrow 1 \rightarrow 1) \forall F [R(F) \iff MF = NF].$$

5.6.7. Show that there is a Diophantine equation $1t \subseteq \mathcal{F}^2$ such that for all $n, m \in \mathbb{N}$

$$1t(R^n, R^m) \iff n < m.$$

5.6.8. Define $\text{Seq}_n^{\mathcal{N}_k}(h)$ iff $h = [R^{m_0}, \dots, R^{m_{n-1}}]$, for some $m_0, \dots, m_{n-1} < k$. Show that $\text{Seq}_n^{\mathcal{N}_k}$ is Diophantine uniformly in n .

5.6.9. Let \mathcal{B} be some finite subset of \mathcal{F} . Define $\text{Seq}_n^{\mathcal{B}}(h)$ iff $h = [g_0, \dots, g_{n-1}]$, with each $g_i \in \mathcal{B}$. Show that $\text{Seq}_n^{\mathcal{B}}$ is Diophantine uniformly in n .

5.6.10. For $\mathcal{B} \subseteq \mathcal{F}$ define \mathcal{B}^+ to be the submonoid generated by \mathcal{B} . Show that if \mathcal{B} is finite, then \mathcal{B}^+ is Diophantine.

5.6.11. Show that $\mathcal{F} \subseteq \mathcal{F}[x]$ is Diophantine.

5.6.12. Construct two concrete terms $t(a, b), s(a, b) \in \mathcal{F}[a, b]$ such that for all $f \in \mathcal{F}$ one has

$$f \in \{R^n \mid n \in \mathbb{N}\} \cup \{L\} \iff \exists g \in \mathcal{F} [t(f, g) = s(f, g)].$$

[Remark. It is not sufficient to notice that Diophantine sets are closed under union. But the solution is not hard and the terms are short.]

5.6.13. Let $2 = \{0, 1\}$ be the discrete topological space with two elements. Let Cantor space be $\mathbf{C} = 2^{\mathbb{N}}$ endowed with the product topology. Define $Z, O : \mathbf{C} \rightarrow \mathbf{C}$ ‘shift operators’ on Cantor space as follows.

$$\begin{aligned} Z(f)(0) &= 0; \\ Z(f)(n+1) &= f(n); \\ O(f)(0) &= 1; \\ O(f)(n+1) &= f(n). \end{aligned}$$

Write $0f = Z(f)$ and $1f = O(f)$. If $\mathcal{X} \subseteq \mathbf{C} \rightarrow \mathbf{C}$ is a set of maps, let \mathcal{X}^+ be the closure of \mathcal{X} under the rule

$$A_0, A_1 \in \mathcal{X} \Rightarrow A \in \mathcal{X},$$

where A is defined by

$$\begin{aligned} A(0f) &= A_0(f); \\ A(1f) &= A_1(f). \end{aligned}$$

(i) Show that if \mathcal{X} consists of continuous maps, then so does \mathcal{X}^+ .

(ii) Show that $A \in \{Z, O\}^+$ iff

$$A(f) = g \Rightarrow \exists r, s \in \mathbb{N} \forall t > s. g(t) = f(t - s + r).$$

(iii) Define on $\{Z, O\}^+$ the following.

$$\begin{aligned} I &= \lambda x \in \{Z, O\}^+. z; \\ L &= Z; \\ R &= O; \\ x * y &= y \circ x; \\ \langle x, y \rangle &= x(f), & \text{if } f(0) = 0; \\ &= y(f), & \text{if } f(0) = 1. \end{aligned}$$

Then $\langle \{Z, O\}^+, *, I, L, R, \langle -, - \rangle \rangle$ is a Cartesian monoid isomorphic to \mathcal{F} , via $\varphi : \mathcal{F} \rightarrow \{Z, O\}^+$.

(iv) The Thompson-Freyd-Heller group can be defined by

$$\{f \in \mathcal{I} \mid \varphi(f) \text{ preserves the lexicographical ordering on } \mathbf{C}\}.$$

Show that the B_n defined in definition 5.2.29 generate this group.

5.6.14. Prove proposition 5.2.9: for all types A one has $A \triangleleft_{SP} N_{rank(A)}$.

5.6.15. Let λ_P be λ_{\rightarrow}^o extended with a simple (not surjective) pairing. Show that theorem 5.2.42 does not hold for this theory. [Hint show that in this theory the equation $\lambda x o. \langle \pi_1 x, \pi_2 x \rangle = \lambda x o. x$ does not hold by constructing a counter model, but is nevertheless consistent.]

5.6.16. Does every model of λ_{SP} have the same first order theory?

5.6.17. Show that $\mathcal{F}[x_1, \dots, x_n]$ for $n \geq 0$ does not have one generator. [Hint. Otherwise this monoid would be commutative, which is not the case.]

5.6.18. Show that $R \subseteq \Lambda^\emptyset(A) \times \Lambda^\emptyset(B)$ is equational iff

$$\exists M, N \in \Lambda^\emptyset(A \rightarrow B \rightarrow 1 \rightarrow 1) \forall F [R(F) \iff MF = NF].$$

5.6.19. Show that there is a Diophantine equation $1t \subseteq \mathcal{F}^2$ such that for all $n, m \in \mathbb{N}$

$$1t(R^n, R^m) \iff n < m.$$

5.6.20. Define $\text{Seq}_n^{N_k}(h)$ iff $h = [R^{m_0}, \dots, R^{m_{n-1}}]$, for some $m_0, \dots, m_{n-1} < k$. Show that $\text{Seq}_n^{N_k}$ is Diophantine uniformly in n .

5.6.21. Let \mathcal{B} be some finite subset of \mathcal{F} . Define $\text{Seq}_n^{\mathcal{B}}(h)$ iff $h = [g_0, \dots, g_{n-1}]$, with each $g_i \in \mathcal{B}$. Show that $\text{Seq}_n^{\mathcal{B}}$ is Diophantine uniformly in n .

5.6.22. For $\mathcal{B} \subseteq \mathcal{F}$ define \mathcal{B}^+ to be the submonoid generated by \mathcal{B} . Show that if \mathcal{B} is finite, then \mathcal{B}^+ is Diophantine.

5.6.23. Show that $\mathcal{F} \subseteq \mathcal{F}[x]$ is Diophantine.

5.6.24. Construct two concrete terms $t(a, b), s(a, b) \in \mathcal{F}[a, b]$ such that for all $f \in \mathcal{F}$ one has

$$f \in \{R^n \mid n \in \mathbb{N}\} \cup \{L\} \iff \exists g \in \mathcal{F} [t(f, g) = s(f, g)].$$

[Remark. It is not sufficient to notice that Diophantine sets are closed under union. But the solution is not hard and the terms are short.]

5.6.25. Let $2 = \{0, 1\}$ be the discrete topological space with two elements. Let Cantor space be $\mathbf{C} = 2^{\mathbb{N}}$ endowed with the product topology. Define $Z, O : \mathbf{C} \rightarrow \mathbf{C}$ ‘shift operators’ on Cantor space as follows.

$$\begin{aligned} Z(f)(0) &= 0; \\ Z(f)(n+1) &= f(n); \\ O(f)(0) &= 1; \\ O(f)(n+1) &= f(n). \end{aligned}$$

Write $0f = Z(f)$ and $1f = O(f)$. If $\mathcal{X} \subseteq \mathbf{C} \rightarrow \mathbf{C}$ is a set of maps, let \mathcal{X}^+ be the closure of \mathcal{X} under the rule

$$A_0, A_1 \in \mathcal{X} \Rightarrow A \in \mathcal{X},$$

where A is defined by

$$\begin{aligned} A(0f) &= A_0(f); \\ A(1f) &= A_1(f). \end{aligned}$$

- (i) Show that if \mathcal{X} consists of continuous maps, then so does \mathcal{X}^+ .
- (ii) Show that $A \in \{Z, O\}^+$ iff

$$A(f) = g \Rightarrow \exists r, s \in \mathbb{N} \forall t > s. g(t) = f(t - s + r).$$

- (iii) Define on $\{Z, O\}^+$ the following.

$$\begin{aligned} I &= \lambda x \in \{Z, O\}^+. z; \\ L &= Z; \\ R &= O; \\ x * y &= y \circ x; \\ \langle x, y \rangle &= x(f), & \text{if } f(0) = 0; \\ &= y(f), & \text{if } f(0) = 1. \end{aligned}$$

Then $\langle \{Z, O\}^+, *, I, L, R, \langle -, - \rangle \rangle$ is a Cartesian monoid isomorphic to \mathcal{F} , via $\varphi : \mathcal{F} \rightarrow \{Z, O\}^+$.

- (iv) The Thompson-Freyd-Heller group can be defined by

$$\{f \in \mathcal{I} \mid \varphi(f) \text{ preserves the lexicographical ordering on } \mathbf{C}\}.$$

Show that the B_n defined in definition 5.2.29 generate this group.

Chapter 6

Applications

6.1. Functional programming

Lambda calculi are prototype programming languages. As is the case with imperative programming languages, where several examples are untyped (machine code, assembler, Basic) and several are typed (Algol-68, Pascal), systems of lambda calculi exist in untyped and typed versions. There are also other differences in the various lambda calculi. The lambda calculus introduced in Church [1936] is the untyped λ I-calculus in which an abstraction $\lambda x.M$ is only allowed if x occurs among the free variables of M . Nowadays, “lambda calculus” refers to the λ K-calculus developed under the influence of Curry, in which $\lambda x.M$ is allowed even if x does not occur in M . There are also typed versions of the lambda calculus. Of these, the most elementary are two versions of the simply typed lambda calculus λ_{\rightarrow} . One version is due to Curry [1934] and has implicit types. Simply typed lambda calculus with explicit types is introduced in Church [1940] (this system is inspired by the theory of types of Russell and Whitehead [1910–13] as simplified by Ramsey [1925]). In order to make a distinction between the two versions of simply typed lambda calculus, the version with explicit types is sometimes called the *Church* version and the one with implicit types the *Curry* version. The difference is that in the Church version one explicitly types a variable when it is bound after a lambda, whereas in the Curry version one does not. So for example in Church’s version one has $\mathbf{l}_A = (\lambda x A.x) : A \rightarrow A$ and similarly $\mathbf{l}_{A \rightarrow B} : (A \rightarrow B) \rightarrow (A \rightarrow B)$, while in Curry’s system one has $\mathbf{l} = (\lambda x.x) : A \rightarrow A$ but also $\mathbf{l} : (A \rightarrow B) \rightarrow (A \rightarrow B)$ for the same term \mathbf{l} . See B[92] for more information about these and other typed lambda calculi. Particularly interesting are the second and higher order calculi λ_2 and λ_ω introduced by Girard [1972] (under the names ‘system F ’ and ‘system $F\omega$ ’) for applications to proof theory and the calculi with dependent types introduced by de Bruijn [1970] for proof verification.

Computing on data types

In this subsection we explain how it is possible to represent data types in a very direct manner in the various lambda calculi.

Lambda definability was introduced for functions on the set of natural numbers \mathbb{N} .

In the resulting mathematical theory of computation (recursion theory) other domains of input or output have been treated as second class citizens by coding them as natural numbers. In more practical computer science, algorithms are also directly defined on other data types like trees or lists.

Instead of coding such data types as numbers one can treat them as first class citizens by coding them directly as lambda terms *while preserving their structure*. Indeed, lambda calculus is strong enough to do this, as was emphasized in Böhm [1963] and Böhm and Gross [1966]. As a result, a much more efficient representation of algorithms on these data types can be given, than when these types were represented via numbers. This methodology was perfected in two different ways in Böhm and Berarducci [1985] and Böhm et al. [1994] or Berarducci and Bohm [1993]. The first paper does the representation in a way that can be typed; the other papers in an essentially stronger way, but one that cannot be typed. We present the methods of these papers by treating labeled trees as an example.

Let the (inductive) data-type of labeled trees be defined by the following abstract syntax.

$$\begin{aligned}\text{tree} &= \bullet \mid \text{leaf nat} \mid \text{tree} + \text{tree} \\ \text{nat} &= 0 \mid \text{succ nat}\end{aligned}$$

We see that a label can be either a bud (\bullet) or a leaf with a number written on it. A typical such tree is $(\text{leaf } 3) + ((\text{leaf } 5) + \bullet)$. This tree together with its mirror image look as follows.

6.2. Logic and proof-checking

The Curry-de Bruijn-Howard correspondence

One of the main applications of type theory is its connection with logic. For several logical systems L there is a type theory λ - and a map translating formulas A of L into types $[A]$ of λ - such that

$$\vdash_L A \iff \Gamma_A \vdash_{\lambda-} M : [A], \text{ for some } M,$$

where Γ_A is some context ‘explaining’ A . The term M can be constructed canonically from a natural deduction proof D of A . So in fact one has

$$\vdash_L A, \text{ with proof } D \iff \Gamma_A \vdash_{\lambda-} [D] : [A], \quad (6.1)$$

where the map $[]$ is extended to cover also derivations. For deductions from a set of assumptions one has

$$\Delta \vdash_L A, \text{ with proof } D \iff \Gamma_A, [\Delta] \vdash_{\lambda-} [D] : [A].$$

Curry did not observe the correspondence in this precise form. He noted that inhabited types in λ_{\rightarrow} , like $A \rightarrow A$ or $A \rightarrow B \rightarrow A$, all had the form of a tautology of (the implication fragment of) propositional logic.

Howard [1980] (the work was done in 1968 and written down in the unpublished but widely circulated Howard [1969]), inspired by the observation of Curry and by Tait [1963], gave the more precise interpretation (6.1). He coined the term *propositions-as-types* and *proofs-as-terms*.

On the other hand, de Bruijn independently of Curry and Howard developed type systems satisfying (6.1). The work was started also in 1968 and the first publication was de Bruijn [1970]; see also de Bruijn [1980]. The motivation of de Bruijn was his visionary view that machine proof checking one day will be feasible and important. The collection of systems he designed was called the AUTOMATH family, derived from AUTOMATIC MATHematics verification. The type systems were such that the right hand side of (6.1) was efficiently verifiable by machine, so that one had machine verification of provability. Also de Bruijn and his students were engaged in developing, using and implementing these systems.

Initially the AUTOMATH project received little attention from mathematicians. They did not understand the technique and worse they did not see the need for machine verification of provability. Also the verification process was rather painful. After five ‘monk’ years of work, van Benthem Jutting [1977] came up with a machine verification of Landau [1900] fully rewritten in the terse ‘machine code’ of one of the AUTOMATH languages. Since then there have been developed second generation versions in the AUTOMATH family, like NUPRL [1979], COQ ([1989]) and LEGO ([1991]), in which considerable help from the computer environment is obtained for the formalization of proofs. With these systems a task of verifying Landau [1900] took something like five months. An important contribution to these second generation systems came from Scott and Martin-Löf, by adding inductive data-types to the systems in order to make formalizations more natural.¹ In Kahn [1995] methods are developed in order to translate proof objects automatically into natural language. It is hoped that in the near future new proofcheckers will emerge in which formalizing is not much more difficult than, say, writing an article in TeX.

Computer Mathematics

Modern systems for computer algebra (CA) are able to represent mathematical notions on a machine and compute with them. These objects can be integers, real or complex numbers, polynomials, integrals and the like. The computations are usually symbolic, but can also be numerical to a virtually arbitrary degree of precision. It is fair to say—as is sometimes done—that “a system for CA can represent $\sqrt{2}$ exactly”. In spite of the fact that this number has an infinite decimal expansion, this is not a miracle. The number $\sqrt{2}$ is represented in a computer just as a symbol (as we do on paper or in our

¹For example, proving Gödel’s incompleteness theorem is difficult for the following reason. The main step in the proof essentially consists of constructing a compiler from a universal programming language into arithmetic. For this one needs to describe strings over an alphabet in the structure of numbers with plus and times. This is difficult and Gödel used the Chinese remainder theorem to do this. Having available the datatype of strings, together with the corresponding operators, makes the translation much more natural.

mind), and the machine knows how to manipulate it. The common feature of these kind of notions represented in systems for CA is that in some sense or another they are all computable. Systems for CA have reached a high level of sophistication and efficiency and are commercially available. Scientists and both pure and applied mathematicians have made good use of them for their research.

There is now emerging a new technology, namely that of systems for Computer Mathematics (CM). In these systems virtually all mathematical notions can be represented exactly, including those that do not have a computational nature. How is this possible? Suppose, for example, that we want to represent a non-computable object like the co-Diophantine set

$$X = \{n \in \mathbb{N} \mid \neg \exists \vec{x} D(\vec{x}, n) = 0\}.$$

Then we can do as before and represent it by a special symbol. But now the computer in general cannot operate on it because the object may be of a non-computational nature.

Before answering the question in the previous paragraph, let us first analyze where non-computability comes from. It is always the case that this comes from the quantifiers \forall (for all) and \exists (exists). Indeed, these quantifiers usually range over an infinite set and therefore one loses decidability.

Nevertheless, for ages mathematicians have been able to obtain interesting information about these non-computable objects. This is because there is a notion of *proof*. Using proofs one can state with confidence that e.g.

$$3 \in X, \text{ i.e., } \neg \exists \vec{x} D(\vec{x}, 3) = 0.$$

Aristotle had already remarked that it is often hard to find proofs, but the verification of a putative one can be done in a relatively easy way. Another contribution of Aristotle was his quest for the formalization of logic. After about 2300 years, when Frege had found the right formulation of predicate logic and Gödel had proved that it is complete, this quest was fulfilled. Mathematical proofs can now be completely formalized and verified by computers. This is the underlying basis for the systems for CM.

Present day prototypes of systems for CM are able to help a user to develop from primitive notions and axioms many theories, consisting of defined concepts, theorems and proofs.² All the systems of CM have been inspired by the AUTOMATH project of de Bruijn (see de Bruijn [1970] and [1990] and Nederpelt et al. [1994]) for the automated verification of mathematical proofs.

Representing proofs as lambda terms

Now that mathematical proofs can be fully formalized, the question arises how this can be done best (for efficiency reasons concerning the machine and pragmatic reasons concerning the human user). Hilbert represented a proof of statement A from a set of axioms Γ as a finite sequence A_0, A_1, \dots, A_n such that $A = A_n$ and each A_i , for $0 \leq i \leq n$, is either in Γ or follows from previous statements using the rules of logic.

²This way of doing mathematics, the axiomatic method, was also described by Aristotle. It was Euclid [n.d.] who first used this method very successfully in his *Elements*.

A more efficient way to represent proofs employs typed lambda terms and is called the *propositions-as-types* interpretation discovered by Curry, Howard and de Bruijn. This interpretation maps propositions into types and proofs into the corresponding inhabitants. The method is as follows. A statement A is transformed into the type (i.e., collection)

$$[A] = \text{the set of proofs of } A.$$

So A is provable if and only if $[A]$ is ‘inhabited’ by a proof p . Now a proof of $A \Rightarrow B$ consists (according to the Brouwer-Heyting interpretation of implication) of a function having as argument a proof of A and as value a proof of B . In symbols

$$[A \Rightarrow B] = [A] \rightarrow [B].$$

Similarly

$$[\forall x \in X. Px] = \Pi x X.[Px],$$

where $\Pi x A.[Px]$ is the Cartesian product of the $[Px]$, because a proof of $\forall x \in A. Px$ consists of a function that assigns to each element $x \in A$ a proof of Px . In this way proof-objects become isomorphic with the intuitionistic natural deduction proofs of Gentzen [1969]. Using this interpretation, a proof of $\forall y \in A. Py \Rightarrow Py$ is $\lambda y A \lambda x Py.x$. Here $\lambda x A.B(x)$ denotes the function that assigns to input $x \in A$ the output $B(x)$. A proof of

$$(A \Rightarrow A \Rightarrow B) \Rightarrow A \Rightarrow B$$

is

$$\lambda p (A \Rightarrow A \Rightarrow B) \lambda q A.pqq.$$

A description of the typed lambda calculi in which these types and inhabitants can be formulated is given in B[92], which also gives an example of a large proof object. Verifying whether p is a proof of A boils down to verifying whether, in the given context, the type of p is equal (convertible) to $[A]$. The method can be extended by also representing connectives like \wedge and \neg in the right type system. Translating propositions as types has as default intuitionistic logic. Classical logic can be dealt with by adding the excluded middle as an axiom.

If a complicated computer system claims that a certain mathematical statement is correct, then one may wonder whether this is indeed the case. For example, there may be software errors in the system. A satisfactory methodological answer has been given by de Bruijn. Proof-objects should be public and written in such a formalism that a reasonably simple proof-checker can verify them. One should be able to verify the program for this proof-checker ‘by hand’. We call this the *de Bruijn criterion*. The proof-development systems Lego (see Luo and Pollack [1992]) and Coq (see Coquand and Huet [1988]) satisfy this criterion.

A way to keep proof-objects from growing too large is to employ the so-called Poincaré principle. Poincaré [1902], p. 12, stated that an argument showing that $2 + 2 = 4$ “is not a proof in the strict sense, it is a verification” (actually he claimed that an arbitrary mathematician will make this remark). In the AUTOMATH project of de Bruijn the

following interpretation of the Poincaré principle was given. If p is a proof of $A(t)$ and $t =_R t'$, then the same p is also a proof of $A(t')$. Here R is a notion of reduction consisting of ordinary β reduction and δ -reduction in order to deal with the unfolding of definitions. Since $\beta\delta$ -reduction is not too complicated to be programmed, the type systems enjoying this interpretation of the Poincaré principle still satisfy the de Bruijn criterion³.

In spite of the compact representation in typed lambda calculi and the use of the Poincaré principle, proof-objects become large, something like 10 to 30 times the length of a complete informal proof. Large proof-objects are tiresome to generate by hand. With the necessary persistence Jutting [1977] has written lambda after lambda to obtain the proof-objects showing that all proofs (but one) in Landau [1960] are correct. Using a modern system for CM one can do better. The user introduces the context consisting of the primitive notions and axioms. Then necessary definitions are given to formulate a theorem to be proved (the goal). The proof is developed in an interactive session with the machine. Thereby the user only needs to give certain ‘tactics’ to the machine. (The interpretation of these tactics by the machine does nothing mathematically sophisticated, only the necessary bookkeeping. The sophistication comes from giving the right tactics.) The final goal of this research is that the necessary effort to interactively generate formal proofs is not more complicated than producing a text in, say, \LaTeX . This goal has not been reached yet. See Barendregt [1996] for references, including those about other approaches to computer mathematics. (These include the systems NuPrl, HOL, Otter, Mizar and the Boyer-Moore theorem prover. These systems do not satisfy the de Bruijn criterion, but some of them probably can be modified easily so that they do.)

Computations in proofs

The following is taken from Barendregt and Barendsen [1997]. There are several computations that are needed in proofs. This happens, for example, if we want to prove formal versions of the following intuitive statements.

- (1) $\lfloor \sqrt{45} \rfloor = 6$, where $\lfloor r \rfloor$ is the integer part of a real;
- (2) $\text{Prime}(61)$;
- (3) $(x+1)(x+1) = x^2 + 2x + 1$.

A way to handle (1) is to use the Poincaré principle extended to the reduction relation \rightarrow_ι for primitive recursion on the natural numbers. Operations like $f(n) = \lfloor \sqrt{n} \rfloor$ are primitive recursive and hence are lambda definable (using $\rightarrow_{\beta\iota}$) by a term, say F , in the lambda calculus extended by an operation for primitive recursion R satisfying

$$\begin{aligned} R A B \text{zero} &\rightarrow_\iota A \\ R A B (\text{succ } x) &\rightarrow_\iota B x (R A B x). \end{aligned}$$

³The reductions may sometimes cause the proof-checking to be of an unacceptable time complexity. We have that p is a proof of A iff $\text{type}(p) =_{\beta\delta} A$. Because the proof is coming from a human, the necessary conversion path is feasible, but to find it automatically may be hard. The problem probably can be avoided by enhancing proof-objects with hints for a reduction strategy.

Then, writing $\ulcorner 0 \urcorner = \mathbf{zero}$, $\ulcorner 1 \urcorner = \mathbf{succ\ zero}$, ..., as

$$\ulcorner 6 \urcorner = \ulcorner 6 \urcorner$$

is formally derivable, it follows from the Poincaré principle that the same is true for

$$F\ulcorner 45 \urcorner = \ulcorner 6 \urcorner$$

(with the same proof-object), since $F\ulcorner 45 \urcorner \twoheadrightarrow_{\beta\iota} \ulcorner 6 \urcorner$. Usually, a proof obligation arises that F is adequately constructed. For example, in this case it could be

$$\forall n (Fn)^2 \leq n < ((Fn) + 1)^2.$$

Such a proof obligation needs to be formally proved, but only once; after that reductions like

$$F\ulcorner n \urcorner \twoheadrightarrow_{\beta\iota} \ulcorner f(n) \urcorner$$

can be used freely many times.

In a similar way, a statement like (2) can be formulated and proved by constructing a lambda defining term $K_{\mathbf{Prime}}$ for the characteristic function of the predicate **Prime**. This term should satisfy the following statement

$$\begin{aligned} \forall n \quad &[(\mathbf{Prime}\ n \leftrightarrow K_{\mathbf{Prime}}\ n = \ulcorner 1 \urcorner) \\ &(K_{\mathbf{Prime}}\ n = \ulcorner 0 \urcorner \vee K_{\mathbf{Prime}}\ n = \ulcorner 1 \urcorner)]. \end{aligned}$$

which is the proof obligation.

Statement (3) corresponds to a symbolic computation. This computation takes place on the syntactic level of formal terms. There is a function g acting on syntactic expressions satisfying

$$g((x+1)(x+1)) = x^2 + 2x + 1,$$

that we want to lambda define. While $x+1 : \mathbf{Nat}$ (in context $x\ \mathbf{Nat}$), the expression on a syntactic level represented internally satisfies ' $x+1$ ' : $\mathbf{term}(\mathbf{Nat})$, for the suitably defined inductive type $\mathbf{term}(\mathbf{Nat})$. After introducing a reduction relation $\twoheadrightarrow_{\iota}$ for primitive recursion over this data type, one can use techniques similar to those of §3 to lambda define g , say by G , so that

$$G\ulcorner (x+1)(x+1) \urcorner \twoheadrightarrow_{\beta\iota} \ulcorner x^2 + 2x + 1 \urcorner.$$

Now in order to finish the proof of (3), one needs to construct a self-interpreter E , such that for all expressions $p : \mathbf{Nat}$ one has

$$E\ulcorner p \urcorner \twoheadrightarrow_{\beta\iota} p$$

and prove the proof obligation for G which is

$$\forall t\ \mathbf{term}(\mathbf{Nat})\ E(Gt) = E\ t.$$

It follows that

$$E(G'(x+1)(x+1)') = E'(x+1)(x+1)';$$

now since

$$\begin{aligned} E(G'(x+1)(x+1)') &\rightarrow_{\beta_\iota} E'(x^2 + 2x + 1) \\ &\rightarrow_{\beta_\iota} x^2 + 2x + 1 \\ E'(x+1)(x+1)' &\rightarrow_{\beta_\iota} (x+1)(x+1), \end{aligned}$$

we have by the Poincaré principle

$$(x+1)(x+1) = x^2 + 2x + 1.$$

The use of inductive types like `Nat` and `term(Nat)` and the corresponding reduction relations for primitive reduction was suggested by Scott [1970] and the extension of the Poincaré principle for the corresponding reduction relations of primitive recursion by Martin-Löf [1984]. Since such reductions are not too hard to program, the resulting proof checking still satisfies the de Bruijn criterion.

In Oostdijk [1996] a program is presented that, for every primitive recursive predicate P , constructs the lambda term K_P defining its characteristic function and the proof of the adequacy of K_P . The resulting computations for $P = \text{Prime}$ are not efficient, because a straightforward (non-optimized) translation of primitive recursion is given and the numerals (represented numbers) used are in a unary (rather than n -ary) representation; but the method is promising. In Elbers [1996], a more efficient ad hoc lambda definition of the characteristic function of `Prime` is given, using Fermat's small theorem about primality. Also the required proof obligation has been given.

Choice of formal systems

There are several possibilities for the choice of a formal system to be used for the representation of theories in systems of computer mathematics. Since, in constructing proof-objects, cooperation between researchers is desirable, this choice has to be made with some care in order to reach an international standard. As a first step towards this, one may restrict attention to systems of typed lambda calculi, since they provide a compact representation and meet de Bruijn's criterion of having a simple proof-checker. In their simplest form, these systems can be described in a uniform way as pure type systems (PTS's) of different strength, see B[92]. The PTS's should be extended by a definition mechanism to become DPTS's (PTS's with definitions), see Severi and Poll [1994]. The DPTS's are good for describing several variants of logic: many sorted predicate logic in its first, second or higher order versions. As stated before, the default logic is intuitionistic, but can be made classical by assuming the excluded middle.

The next step consists of adding inductive types (IT's) and the corresponding reduction relations in order to capture primitive recursion. We suggest that the right formal systems to be used for computer mathematics are the type systems (TS), consisting of DPTS's extended by IT's, as described e.g. in Paulin-Mohring [1994]. TS's come with two

parameters. The first is the specification \mathcal{A} of the underlying PTS specifying its logical strength, see B[92]. The second is \mathcal{B} the collection of inductive types and their respective notions of reduction \rightarrow_i , specifying its mathematical and computational strength. In my opinion, a system for proof-checking should be able to verify proof-objects written in all the systems $\text{TS}(\mathcal{A}, \mathcal{B})$ (for a ‘reasonable’ choice spectrum of the parameters). If someone wants to use it for only a subclass of the choice of parameters—dictated by that person’s foundational views—then the proof-checker will do its work anyway. I believe that this generality will not be too expensive in terms of the complexity of the checking.⁴

Illative lambda calculus

Curry and his students continued to look for a way to represent functions and logic into one adequate formal system. Some of the proposed systems turned out to be inconsistent, other ones turned out to be incomplete. Research in TS’s for the representation of logic has resulted in an unexpected side effect. By making a modification inspired by the TS’s, it became possible, after all, to give an extension of the untyped lambda calculus, called *Illative Lambda Calculi* (ILC; ‘illative’ from the Latin word *inferre* which means to infer), such that first order logic can be faithfully and completely embedded into it. The method can be extended for an arbitrary PTS⁵, so that higher order logic can be represented too.

The resulting ILC’s are in fact simpler than the TS’s. But doing computer mathematics via ILC is probably not very practical, as it is not clear how to do proof-checking for these systems.

One nice thing about the ILC is that the old dream of Church and Curry came true, namely, there is one system based on untyped lambda calculus (or combinators) on which logic, hence mathematics, can be based. More importantly there is a ‘combinatory transformation’ between the ordinary interpretation of logic and its propositions-as-types interpretation. Basically, the situation is as follows. The interpretation of predicate logic in ILC is such that

$$\begin{aligned} \vdash_{\text{logic}} A \text{ with proof } p &\iff \forall r \vdash_{\text{ILC}} [A]_r[p] \\ &\iff \vdash_{\text{ILC}} [A]_I[p] \\ &\iff \vdash_{\text{ILC}} [A]_K[p] = K[A]'_I[p] = [A]'_I, \end{aligned}$$

where r ranges over untyped lambda terms. Now if $r = I$, then this translation is the propositions-as-types interpretation; if, on the other hand, one has $r = K$, then the interpretation becomes an isomorphic version of first order logic denoted by $[A]'_I$. See

⁴It may be argued that the following list of features is so important that they deserve to be present in TS’s as primitives and be implemented: quotient types (see Hofmann [1977]), subtypes (see Aspinall and Compagnoni [1996]) and type inclusion (see Luo and Pollack [1992]). This is an interesting question and experiments should be done to determine whether this is the case or whether these can be translated into the more basic TS’s in a sufficiently efficient way (possibly using some macros in the system for CM).

⁵For first order logic, the embedding is natural, but e.g. for second order logic this is less so. It is an open question whether there exists a natural representation of second and higher order logic in ILC.

Barendregt et al. [1993] and Dekkers et al. [1997] for these results. A short introduction to ILC (in its combinatory version) can be found in B[84], Appendix B.

6.3. Proof theory

Lambda terms for natural deduction, sequent calculus and cut elimination

It is well-known that there is a good correspondence between natural deduction derivations and typed lambda terms. Moreover normalizing these terms is equivalent to eliminating cuts in the corresponding sequent calculus derivations. Several papers have been written on this topic. The correspondence between sequent calculus derivations and natural deduction derivations is, however, not a one-to-one map. This causes some syntactic technicalities. The correspondence is best explained by two extensionally equivalent type assignment systems for untyped lambda terms, one corresponding to natural deduction (λN) and the other to sequent calculus (λL). These two systems constitute different grammars for generating the same (type assignment relation for untyped) lambda terms. The second grammar is ambiguous, but the first one is not. This fact explains the many-one correspondence mentioned above. Moreover, the second type assignment system has a ‘cut-free’ fragment (λL^{cf}). This fragment generates exactly the typeable lambda terms in normal form. The cut elimination theorem becomes a simple consequence of the fact that typed lambda terms possess a normal form.

Introduction

It is well-known that there is a good correspondence between natural deduction derivations and typed lambda terms. The relation between lambda terms and derivations in sequent calculus, between normal lambda terms and cut-free derivations in sequent calculus and finally between normalization of terms and cut elimination of derivations has been observed by several authors (Prawitz [1965], Zucker [1974] and Pottinger [1977]). This relation is less perfect because several cut-free sequent derivations correspond to one lambda term. In Herbelin [1995] a lambda calculus with explicit substitution operators is used in order to establish a perfect match between terms of that calculus and sequent derivations. We will not avoid the mismatch, but get a satisfactory view of it, by seeing the sequent calculus as a more intensional way to do the same as natural deduction: assigning lambda terms to provable formulas.

Next to the well-known system λ_{\rightarrow} of Curry type assignment to type free terms, which here will be denoted by λN , there are two other systems of type assignment: λL and its cut-free fragment λL^{cf} . The three systems λN , λL and λL^{cf} correspond exactly to the natural deduction calculus NJ , the sequent calculus LJ and the cut-free fragment of LJ , here denoted by N , L and L^{cf} respectively. Moreover, λN and λL generate the same type assignment relation. The system λL^{cf} generates the same type assignment relation as λN restricted to normal terms and cut elimination corresponds exactly to normalization. The mismatch between the logical systems that was observed above, is due to the fact that λN is a syntax directed system, whereas both λL and λL^{cf} are not.

(A syntax directed version of λL is possible if rules with arbitrarily many assumptions are allowed, see Capretta and Valentini [1998].)

The type assignment system of this paper is a subsystem of one in Barbanera et al. [1995] and also implicitly present in Mints [1996]. So our contribution is mainly expository.

For simplicity the results are presented only for the essential kernel of intuitionistic logic, i.e. for the minimal implicational fragment. The method probably can be extended to the full logical system, using the terms as in Mints [1996].

The logical systems N , L and $L^{\mathbf{cf}}$

6.3.1. DEFINITION. The set **form** of formulas (of minimal implicational propositional logic) is defined by the following abstract syntax.

$\begin{aligned} \text{form} &= \text{atom} \mid \text{form} \rightarrow \text{form} \\ \text{atom} &= p \mid \text{atom}' \end{aligned}$

We write p, q, r, \dots for arbitrary atoms and A, B, C, \dots for arbitrary formulas. Sets of formulas are denoted by Γ, Δ, \dots . The set Γ, A stands for $\Gamma \cup \{A\}$.

6.3.2. DEFINITION. (i) A statement A is *derivable* in the system N from the set Γ , notation $\Gamma \vdash_N A$, if $\Gamma \vdash A$ can be generated by the following axiom and rules.

N	
$\frac{A \in \Gamma}{\Gamma \vdash A}$	axiom
$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$	\rightarrow elim
$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$	\rightarrow intr

(ii) A statement A is *derivable* from assumptions Γ in the system L , notation $\Gamma \vdash_L A$, if $\Gamma \vdash A$ can be generated by the following axiom and rules.

L	
$\frac{A \in \Gamma}{\Gamma \vdash A}$	axiom
$\frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \rightarrow B \vdash C}$	\rightarrow left
$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$	\rightarrow right
$\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B}$	cut

(iii) The system L^{cf} is obtained from the system L by omitting the rule (cut).

L^{cf}	
$\frac{A \in \Gamma}{\Gamma \vdash A}$	axiom
$\frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \rightarrow B \vdash C}$	\rightarrow left
$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$	\rightarrow right

6.3.3. LEMMA. *Suppose $\Gamma \subseteq \Gamma'$. Then*

$$\Gamma \vdash A \Rightarrow \Gamma' \vdash A$$

in all systems.

PROOF. By a trivial induction on derivations. ■

6.3.4. PROPOSITION. *For all Γ and A we have*

$$\Gamma \vdash_N A \iff \Gamma \vdash_L A.$$

PROOF. (\Rightarrow) By induction on derivations in N . For the rule (\rightarrow elim) we need the rule (cut).

$$\frac{\Gamma \vdash_L A \rightarrow B \quad \frac{\Gamma \vdash_L A \quad \overline{\Gamma, B \vdash_L B}^{(\text{axiom})}}{\Gamma, A \rightarrow B \vdash_L B}^{(\rightarrow \text{ left})}}{\Gamma \vdash_L B}^{(\text{cut})}$$

(\Leftarrow) By induction on derivations in L . The rule (\rightarrow left) is treated as follows.

$$\frac{\frac{\Gamma \vdash_N A}{\Gamma, A \rightarrow B \vdash_N A}^{(6.3.3)} \quad \frac{\overline{\Gamma, A \rightarrow B \vdash_N A \rightarrow B}^{(\text{axiom})}}{\Gamma, A \rightarrow B \vdash_N B}^{(\rightarrow \text{ elim})} \quad \frac{\Gamma, B \vdash_N C}{\Gamma \vdash_N B \rightarrow C}^{(\rightarrow \text{ intr})}}{\Gamma, A \rightarrow B \vdash_N C}^{(\rightarrow \text{ elim})}$$

The rule (cut) is treated as follows.

$$\frac{\Gamma \vdash_N A \quad \frac{\Gamma, A \vdash_N B}{\Gamma \vdash_N A \rightarrow B}^{(\rightarrow \text{ intr})}}{\Gamma \vdash_N B}^{(\rightarrow \text{ elim}). \blacksquare}$$

6.3.5. DEFINITION. Consider the following rule as alternative to the rule (cut).

$$\frac{\Gamma, A \rightarrow A \vdash B}{\Gamma \vdash B}^{(\text{cut}')} \quad \blacksquare$$

The system L' is defined by replacing the rule (cut) by (cut').

6.3.6. PROPOSITION. For all Γ and A

$$\Gamma \vdash_L A \iff \Gamma \vdash_{L'} A.$$

PROOF. (\Rightarrow) The rule (cut) is treated as follows.

$$\frac{\frac{\Gamma \vdash_{L'} A \quad \Gamma, A \vdash_{L'} B}{\Gamma, A \rightarrow A \vdash_{L'} B}^{(\rightarrow \text{ left})}}{\Gamma \vdash_{L'} B}^{(\text{cut}')} \quad \blacksquare$$

(\Leftarrow) The rule (cut') is treated as follows.

$$\frac{\Gamma, A \rightarrow A \vdash_L B \quad \frac{\overline{\Gamma, A \vdash_L A}^{(\text{axiom})}}{\Gamma \vdash_L A \rightarrow A}^{(\rightarrow \text{ right})}}{\Gamma \vdash_L B}^{(\text{cut}). \blacksquare}$$

Note that we have not yet investigated the role of L^{cf} .

The type assignment systems λN , λL and λL^{cf}

6.3.7. DEFINITION. The set **term** of type-free lambda terms is defined as follows.

$\begin{array}{lcl} \text{term} & = & \text{var} \mid \text{term term} \mid \lambda \text{var. term} \\ \text{var} & = & x \mid \text{var}' \end{array}$
--

We write x, y, z, \dots for arbitrary variables in terms and P, Q, R, \dots for arbitrary terms. Equality of terms (up to renaming of bound variables) is denoted by \equiv . The identity is $I \equiv \lambda x.x$. A term P is called a β normal form (P is in β -nf) if P has no *redex* as part, i.e. no subterm of the form $(\lambda x.R)S$. Such a redex is said to reduce as follows

$$(\lambda x.R)S \rightarrow_{\beta} R[x:=S],$$

where $R[x:=S]$ denotes the result of substituting S for the free occurrences of x . The transitive reflexive closure of \rightarrow_{β} is denoted by \rightarrow_{β}^* . If $P \rightarrow_{\beta}^* Q$ and Q is in β -nf, then Q is called the β -nf of P (one can show it is unique). A collection \mathcal{A} of terms is said to be *strongly normalizing* if for no $P \in \mathcal{A}$ there is an infinite reduction path

$$P \rightarrow_{\beta} P_1 \rightarrow_{\beta} P_2 \dots$$

6.3.8. DEFINITION. (i) A *type assignment* is an expression of the form

$$P : A,$$

where P is a lambda term and A is a formula.

(ii) A *declaration* is a type assignment of the form

$$x : A.$$

(iii) A *context* Γ is a set of declarations such that for every variable x there is at most one declaration $x : A$ in Γ .

6.3.9. DEFINITION. (i) A type assignment $P : A$ is *derivable* from the context Γ in the system λN (also known as λ_{\rightarrow}), notation

$$\Gamma \vdash_{\lambda N} P : A,$$

if $\Gamma \vdash P : A$ can be generated by the following axiom and rules.

λN	
$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$	axiom
$\frac{\Gamma \vdash P : (A \rightarrow B) \quad \Gamma \vdash Q : A}{\Gamma \vdash (PQ) : B}$	\rightarrow elim
$\frac{\Gamma, x : A \vdash P : B}{\Gamma \vdash (\lambda x.P) : (A \rightarrow B)}$	\rightarrow intr

(ii) A type assignment $P : A$ is *derivable* from the context Γ in the system λL , notation

$$\Gamma \vdash_{\lambda L} P : A,$$

if $\Gamma \vdash P : A$ can be generated by the following axiom and rules.

λL	
$\frac{(x A) \in \Gamma}{\Gamma \vdash x A}$	axiom
$\frac{\Gamma \vdash Q : A \quad \Gamma, x B \vdash P : C}{\Gamma, y : A \rightarrow B \vdash P[x:=yQ] : C}$	\rightarrow left
$\frac{\Gamma, x A \vdash P : B}{\Gamma \vdash (\lambda x.P) : (A \rightarrow B)}$	\rightarrow right
$\frac{\Gamma \vdash Q : A \quad \Gamma, x A \vdash P : B}{\Gamma \vdash P[x:=Q] : B}$	cut

In the rule (\rightarrow left) it is required that $\Gamma, y A \rightarrow B$ is a context. This is the case if y is fresh or if $\Gamma = \Gamma, y A \rightarrow B$, i.e. $y A \rightarrow B$ already occurs in Γ .

(iii) The system λL^{cf} is obtained from the system λL by omitting the rule (cut).

λL^{cf}	
$\frac{(x A) \in \Gamma}{\Gamma \vdash x A}$	axiom
$\frac{\Gamma \vdash Q : A \quad \Gamma, x B \vdash P : C}{\Gamma, y : A \rightarrow B \vdash P[x:=yQ] : C}$	\rightarrow left
$\frac{\Gamma, x A \vdash P : B}{\Gamma \vdash (\lambda x.P) : (A \rightarrow B)}$	\rightarrow right

6.3.10. REMARK. The alternative rule (cut') could also have been used to define the variant $\lambda L'$. The right version for the rule (cut') with term assignment is as follows.

Rule cut' for $\lambda L'$	
$\frac{\Gamma, x A \rightarrow A \vdash P : B}{\Gamma \vdash P[x:=I] : B}$	cut'

NOTATION. Let $\Gamma = \{A_1, \dots, A_n\}$ and $\vec{x} = \{x_1, \dots, x_n\}$. Write

$$\Gamma_{\vec{x}} = \{x_1 A_1, \dots, x_n A_n\}$$

and

$$\Lambda^\circ(\vec{x}) = \{P \in \mathbf{term} \mid FV(P) \subseteq \vec{x}\},$$

where $FV(P)$ is the set of free variables of P .

The following result has been observed for N and λN by Curry, Howard and de Bruijn. (See Troelstra and Schwichtenberg [1996] 2.1.5. and Hindley [1997] 6B3, for some fine points about the correspondence between deductions in N and corresponding terms in λN .)

6.3.11. PROPOSITION (Propositions—as—types interpretation). *Let S be one of the logical systems N , L or L^{cf} and let λS be the corresponding type assignment system. Then*

$$\Gamma \vdash_S A \Leftrightarrow \exists \vec{x} \exists P \in \Lambda^\circ(\vec{x}) \Gamma_{\vec{x}} \vdash_{\lambda S} P : A.$$

PROOF. (\Rightarrow) By an easy induction on derivations, just observing that the right lambda term can be constructed. (\Leftarrow) By omitting the terms. ■

Since λN is exactly λ_{\rightarrow} , the simply typed lambda calculus, we know the following results whose proofs are not hard, but are omitted here. From corollary 6.3.15 it follows that the results also hold for λL .

6.3.12. PROPOSITION. (i) (*Normalization theorem for λN*)

$$\Gamma \vdash_{\lambda N} P : A \Rightarrow P \text{ has a } \beta\text{-nf } P^{nf}.$$

(ii) (*Subject reduction theorem for λN*)

$$\Gamma \vdash_{\lambda N} P : AP \rightarrow_\beta P' \Rightarrow \Gamma \vdash_{\lambda N} P' : A.$$

(iii) (*Generation lemma for λN*) *Type assignment for terms of a certain syntactic form is caused in the obvious way.*

- (1) $\Gamma \vdash_{\lambda N} x : A \Rightarrow (x A) \in \Gamma.$
- (2) $\Gamma \vdash_{\lambda N} PQ : B \Rightarrow \Gamma \vdash_{\lambda N} P : (A \rightarrow B) \Gamma \vdash_{\lambda N} Q : A,$
for some type A .
- (3) $\Gamma \vdash_{\lambda N} \lambda x.P : C \Rightarrow \Gamma, x A \vdash_{\lambda N} P : BC \equiv A \rightarrow B,$
for some types A, B .

PROOF. See e.g. Gandy [1980a] for (i) and Barendregt [1992] for (ii) and (iii). ■

Actually, even strong normalization holds for terms typeable in λN (see e.g. de Vrijer [1987] or Barendregt [1992]).

Relating λN , λL and λL^{cf}

Now the proof of the equivalence between systems N and L will be ‘lifted’ to that of λN and λL .

6.3.13. PROPOSITION. $\Gamma \vdash_{\lambda N} P : A \Rightarrow \Gamma \vdash_{\lambda L} P : A$.

PROOF. By inductions on derivations in λN . Modus ponens (\rightarrow elim) is treated as follows.

$$\frac{\Gamma \vdash_{\lambda L} P : A \rightarrow B \quad \frac{\Gamma \vdash_{\lambda L} Q : A \quad \Gamma, x B \vdash_{\lambda L} x B}{\Gamma, y A \rightarrow B \vdash_{\lambda L} y Q : B} (\rightarrow \text{ left})}{\Gamma \vdash_{\lambda L} P Q : B} (\text{cut}). \blacksquare$$

6.3.14. PROPOSITION. (i) $\Gamma \vdash_{\lambda L} P : A \Rightarrow \Gamma \vdash_{\lambda N} P' : A$, for some $P' \rightarrow_{\beta} P$.

(ii) $\Gamma \vdash_{\lambda L} P : A \Rightarrow \Gamma \vdash_{\lambda N} P : A$.

PROOF. (i) By induction on derivations in λL . The rule (\rightarrow left) is treated as follows (the justifications are left out, but they are as in the proof of 6.3.4).

$$\frac{\frac{\Gamma \vdash_{\lambda N} Q : A}{\Gamma, y A \rightarrow B \vdash_{\lambda N} Q : A} \quad \frac{}{\Gamma, y A \rightarrow B \vdash_{\lambda N} y A \rightarrow B} \quad \frac{\Gamma, x B \vdash_{\lambda N} P : C}{\Gamma \vdash_{\lambda N} (\lambda x. P) : B \rightarrow C}}{\Gamma, y A \rightarrow B \vdash_{\lambda N} y Q : B} \quad \frac{}{\Gamma, y A \rightarrow B \vdash_{\lambda N} (\lambda x. P)(y Q) : C} (\text{cut})$$

Now $(\lambda x. P)(y Q) \rightarrow_{\beta} P[x:=yQ]$ as required. The rule (cut) is treated as follows.

$$\frac{\frac{\Gamma, x A \vdash_{\lambda N} P : B}{\Gamma \vdash_{\lambda N} Q : A \quad \Gamma \vdash_{\lambda N} (\lambda x. P) : A \rightarrow B} (\rightarrow \text{ intr})}{\Gamma \vdash_{\lambda N} (\lambda x. P) Q : B} (\rightarrow \text{ elim})$$

Now $(\lambda x. P) Q \rightarrow_{\beta} P[x:=Q]$ as required.

(ii) By (i) and the subject reduction theorem for λN (6.3.12(ii)). \blacksquare

6.3.15. COROLLARY. $\Gamma \vdash_{\lambda L} P : A \iff \Gamma \vdash_{\lambda N} P : A$.

PROOF. By propositions 6.3.13 and 6.3.14(ii). \blacksquare

Now we will investigate the role of the cut-free system.

6.3.16. PROPOSITION.

$$\Gamma \vdash_{\lambda L^{\text{cf}}} P : A \Rightarrow P \text{ is in } \beta\text{-nf}.$$

PROOF. By an easy induction on derivations. ■

6.3.17. LEMMA. *Suppose*

$$\Gamma \vdash_{\lambda L^{\text{cf}}} P_1 : A_1, \dots, \Gamma \vdash_{\lambda L^{\text{cf}}} P_n : A_n.$$

Then

$$\Gamma, x A_1 \rightarrow \dots \rightarrow A_n \rightarrow B \vdash_{\lambda L^{\text{cf}}} x P_1 \dots P_n : B$$

for those variables x such that $\Gamma, x A_1 \rightarrow \dots \rightarrow A_n \rightarrow B$ is a context.

PROOF. We treat the case $n = 2$, which is perfectly general. We abbreviate $\vdash_{\lambda L^{\text{cf}}}$ as \vdash .

$$\frac{\Gamma \vdash P_1 : A_1 \quad \frac{\Gamma \vdash P_2 : A_2 \quad \overline{\Gamma, z B \vdash z : B}^{\text{(axiom)}}}{\Gamma, y A_2 \rightarrow B \vdash y P_2 \equiv z[z:=y P_2] : B} (\rightarrow \text{ left})}{\Gamma, x A_1 \rightarrow A_2 \rightarrow B \vdash x P_1 P_2 \equiv (y P_2)[y:=x P_1] : B} (\rightarrow \text{ left})$$

Note that x may occur in some of the P_i . ■

6.3.18. PROPOSITION. *Suppose that P is a β -nf. Then*

$$\Gamma \vdash_{\lambda N} P : A \Rightarrow \Gamma \vdash_{\lambda L^{\text{cf}}} P : A.$$

PROOF. By induction on the following generation of normal forms.

$$\text{nf} = \text{var} \mid \text{var nf}^+ \mid \lambda \text{var.nf}$$

The cases $P \equiv x$ and $P \equiv \lambda x. P_1$ are easy. The case $P \equiv x P_1 \dots P_n$ follows from the previous lemma, using the generation lemma for λN (6.3.12(iii)). ■

Now we get as bonus the Hauptsatz of Gentzen [1936] for minimal implicational sequent calculus.

6.3.19. THEOREM (Cut elimination).

$$\Gamma \vdash_L A \Rightarrow \Gamma \vdash_{L^{\text{cf}}} A.$$

$$\begin{aligned} \text{PROOF. } \Gamma \vdash_L A &\Rightarrow \Gamma_{\vec{x}} \vdash_{\lambda L} P : A, && \text{for some } P \in \Lambda^\circ(\vec{x}), \text{ by 6.3.11,} \\ &\Rightarrow \Gamma_{\vec{x}} \vdash_{\lambda N} P : A, && \text{by 6.3.14(ii),} \\ &\Rightarrow \Gamma_{\vec{x}} \vdash_{\lambda N} P^{\text{nf}} : A, && \text{by 6.3.12(i),(ii),} \\ &\Rightarrow \Gamma_{\vec{x}} \vdash_{\lambda L^{\text{cf}}} P^{\text{nf}} : A, && \text{by 6.3.18,} \\ &\Rightarrow \Gamma \vdash_{L^{\text{cf}}} A, && \text{by 6.3.11. } \blacksquare \end{aligned}$$

As it is clear that the proof implies that cut-elimination can be used to normalize terms typable in $\lambda N = \lambda \rightarrow$, Statman [1979] implies that the expense of cut-elimination is beyond elementary time (Grzegorzczuk class 4). Moreover, as the cut-free deduction is of the same order of complexity as the corresponding normal lambda term, the size of the cut-free version of a derivation is non elementary in the size of the original derivation.

Discussion

The main technical tool is the type assignment system λL corresponding exactly to sequent calculus (for minimal propositional logic). The type assignment system λL is a subsystem of a system studied in Barbanera et al. [1995]. The terms involved in λL are also in Mints [1996]. The difference between the present approach and the one by Mints is that in that paper derivations in L are first order citizens, whereas in λL the provable formulas and the lambda terms are.

In λN typeable terms are built up as usual (following the grammar of lambda terms). In λL^{cf} only normal terms are typeable. They are built up from variables by transitions like

$$P \mapsto \lambda x.P$$

and

$$P \mapsto P[x:=yQ]$$

This is an ambiguous way of building terms, in the sense that one term can be built up in several ways. For example, one can assign to the term $\lambda x.yz$ the type $C \rightarrow B$ (in the context $z A, y A \rightarrow B$) via two different cut-free derivations:

$$\frac{\frac{x C, z A \vdash z : A \quad x C, z A, u B \vdash u : B}{x C, z A, y A \rightarrow B \vdash yz : B} (\rightarrow \text{ left})}{z A, y A \rightarrow B \vdash \lambda x.yz : C \rightarrow B} (\rightarrow \text{ right})$$

and

$$\frac{\frac{z A \vdash z A \quad x C, z A, u B \vdash u : B}{z A, u B \vdash \lambda x.u : C \rightarrow B} (\rightarrow \text{ right})}{z A, y A \rightarrow B \vdash \lambda x.yz : C \rightarrow B} (\rightarrow \text{ left})$$

These correspond, respectively, to the following two formations of terms

$$\begin{aligned} u &\mapsto yz && \mapsto \lambda x.yz, \\ u &\mapsto \lambda x.u && \mapsto \lambda x.yz. \end{aligned}$$

Therefore there are more sequent calculus derivations giving rise to the same lambda term. This is the cause of the mismatch between sequent calculus and natural deduction as described in Zucker [1974], Pottinger [1977] and Mints [1996]. See also Dyckhoff and Pinto [1997], Schwichtenberg [1997] and Troelstra [1998].

In Herbelin [1995] the mismatch between L-derivations and lambda terms is repaired by translating these into terms with explicit substitution:

$$\begin{aligned} &\lambda x.(u < u:=yz >), \\ &(\lambda x.u) < u:=yz > . \end{aligned}$$

In our paper lambda terms are considered as first class citizens also for sequent calculus. This gives an insight into the mentioned mismatch by understanding it as an intensional aspect how the sequent calculus generates these terms.

It is interesting to note, how in the full system λL the rule (cut) generates terms not in β -normal form. The extra transition now is

$$P \mapsto P[x:=F].$$

This will introduce a redex, if x occurs actively (in a context xQ) and F is an abstraction ($F \equiv \lambda x.R$), the other applications of the rule (cut) being superfluous. Also, the alternative rule (cut') can be understood better. Using this rule the extra transition becomes

$$P \mapsto P[x:=I].$$

This will have the same effect (modulo one β -reduction) as the previous transition, if x occurs in a context xFQ . So with the original rule (cut) the argument Q (in the context xQ) is waiting for a function F to act on it. With the alternative rule (cut') the function F comes close (in context xFQ), but the 'couple' FQ has to wait for the 'green light' provided by I .

Also, it can be observed that if one wants to manipulate derivations in order to obtain a cut-free proof, then the term involved gets reduced. By the strong normalization theorem for $\lambda N (= \lambda_{\rightarrow})$ it follows that eventually a cut-free proof will be reached.

We have not studied in detail whether cut elimination can be done along the lines of this paper for the full system of intuitionistic predicate logic, but there seems to be no problem. More interesting is the question, whether there are similar results for classical and linear logic.

6.4. Grammars, terms and types

Typed lambda calculus is widely used in the study of natural language semantics, in combination with a variety of rule-based syntactic engines. In this section, we focus on categorial type logics. The type discipline, in these systems, is responsible both for the construction of grammatical form (syntax) and for meaning assembly. We address two central questions. First, what are the *invariants* of grammatical composition, and how do they capture the uniformities of the form/meaning correspondence across languages? Secondly, how can we reconcile grammatical invariants with structural diversity, i.e. variation in the realization of the form/meaning correspondence in the 6000 or so languages of the world?

The grammatical architecture to be unfolded below has two components. Invariants are characterized in terms of a minimal *base system*: the pure logic of residuation for composition and structural incompleteness. Viewing the types of the base system as formulas, we model the syntax-semantics interface along the lines of the Curry-Howard interpretation of derivations. Variation arises from the combination of the base logic with a *structural module*. This component characterizes the structural deformations under which the basic form-meaning associations are preserved. Its rules allow reordering and/or restructuring of grammatical material. These rules are not globally available, but keyed to unary type-forming operations, and thus anchored in the lexical type declarations.

It will be clear from this description that the type-logical approach has its roots in the type calculi developed by Jim Lambek in the late Fifties of the last century. The technique of controlled structural options is a more recent development, inspired by the modalities of linear logic.

Grammatical invariants: the base logic

Compared to the systems used elsewhere in this chapter, the type system of categorial type logics can be seen as a specialization designed to take linear order and phrase structure information into account.

$$\mathcal{F} ::= \mathcal{A} \mid \mathcal{F}/\mathcal{F} \mid \mathcal{F} \bullet \mathcal{F} \mid \mathcal{F} \backslash \mathcal{F}$$

The set of type atoms \mathcal{A} represents the basic ontology of phrases that one can think of as grammatically ‘complete’. Examples, for English, could be np for noun phrases, s for sentences, n for common nouns. There is no claim of universality here: languages can differ as to which ontological choices they make. Formulas A/B , $B \backslash A$ are directional versions of the implicational type $B \rightarrow A$. They express incompleteness in the sense that expressions with slash types produce a phrase of type A in composition with a phrase of type B to the right or to the left. Product types $A \bullet B$ explicitly express this composition.

Frame semantics provides the tools to make the informal description of the interpretation of the type language in the structural dimension precise. Frames $F = (W, R_\bullet)$, in this setting, consist of a set W of linguistic resources (expressions, ‘signs’), structured in terms of a ternary relation R_\bullet , the relation of grammatical composition or ‘Merge’ as it is known in the generative tradition. A valuation $V : \mathcal{S} \mapsto \mathcal{P}(W)$ interprets types as sets of expressions. For complex types, the valuation respects the clauses below, i.e. expressions x with type $A \bullet B$ can be disassembled into an A part y and a B part z . The interpretation for the directional implications is dual with respect to the y and z arguments of the Merge relation, thus expressing incompleteness with respect to composition.

$$x \in V(A \bullet B) \text{ iff } \exists yz. R_\bullet xyz \text{ and } y \in V(A) \text{ and } z \in V(B)$$

$$y \in V(C/B) \text{ iff } \forall xz. (R_\bullet xyz \text{ and } z \in V(B)) \text{ implies } x \in V(C)$$

$$z \in V(A \backslash C) \text{ iff } \forall xy. (R_\bullet xyz \text{ and } y \in V(A)) \text{ implies } x \in V(C)$$

Algebraically, this interpretation turns the product and the left and right implications into a residuated triple in the sense of the following biconditionals:

$$A \longrightarrow C/B \iff A \bullet B \longrightarrow C \iff B \longrightarrow A \backslash C \quad (\text{Res})$$

In fact, we have the *pure* logic of residuation here: (Res), together with Reflexivity ($A \longrightarrow A$) and Transitivity (from $A \longrightarrow B$ and $B \longrightarrow C$, conclude $A \longrightarrow C$), fully characterizes the derivability relation, as the following completeness result shows.

COMPLETENESS $A \longrightarrow B$ is provable in the grammatical base logic iff for every valuation V on every frame F we have $V(A) \subseteq V(B)$ (Došen 1992, Kurtonina 1995).

Notice that we do not impose any restrictions on the interpretation of the Merge relation. In this sense, the laws of the base logic capture grammatical *invariants*: properties of type combination that hold no matter what the structural particularities of individual languages may be. And indeed, at the level of the base logic important grammatical notions, rather than being postulated, can be seen to emerge from the type structure.

- Valency. Selectional requirements distinguishing intransitive $np \backslash s$, transitive $(np \backslash s)/np$, ditransitive $((np \backslash s)/np)/np$, etc verbs are expressed in terms of the directional implications.

In a context-free grammar, these would require the postulation of new non-terminals.

- Case. The distinction between phrases that can fulfill any noun phrase selectional requirement versus phrases that insist on playing the subject $s/(np \backslash s)$, direct object $((np \backslash s)/np)/(np \backslash s)$, prepositional object $(pp/np) \backslash pp$, etc role, is expressed through higher-order type assignment.

- Complements versus modifiers. Compare exocentric types A/B with $A \neq B$ versus endocentric types A/A . The latter express modification; optionality of A/A type phrases follows.

- Filler-gap dependencies. Nested implications $A/(C/B)$, $A/(B \backslash C)$, etc, signal the withdrawal of a gap hypothesis of type B in a domain of type C .

Parsing-as-deduction For automated proof search, one turns the algebraic presentation in terms of (Res) into a sequent presentation enjoying cut elimination. Sequent for the grammatical base logic are statements $\Gamma \Rightarrow A$ with Γ a structure, A a type formula. Structures are binary branching trees with formulas at the leaves: $\mathcal{S} ::= \mathcal{F} \mid (\mathcal{S}, \mathcal{S})$. In the rules, we write $\Gamma[\Delta]$ for a structure Γ containing a substructure Δ . Lambek (1958, 1961) proves that Cut is an admissible rule in this presentation. Top-down backward-chaining proof search in the cut-free system respects the subformula property and yields a decision procedure.

$$\begin{array}{c}
\frac{}{A \Rightarrow A} \text{Ax} \quad \frac{\Delta \Rightarrow A \quad \Gamma[A] \Rightarrow B}{\Gamma[\Delta] \Rightarrow B} \text{Cut} \\
\\
\frac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{(\Gamma, \Delta) \Rightarrow A \bullet B} (\bullet R) \quad \frac{\Gamma[(A, B)] \Rightarrow C}{\Gamma[A \bullet B] \Rightarrow C} (\bullet L) \\
\\
\frac{\Delta \Rightarrow B \quad \Gamma[A] \Rightarrow C}{\Gamma[(\Delta, B \backslash A)] \Rightarrow C} (\backslash L) \quad \frac{(B, \Gamma) \Rightarrow A}{\Gamma \Rightarrow B \backslash A} (\backslash R) \\
\\
\frac{\Delta \Rightarrow B \quad \Gamma[A] \Rightarrow C}{\Gamma[(A/B, \Delta)] \Rightarrow C} (/L) \quad \frac{(\Gamma, B) \Rightarrow A}{\Gamma \Rightarrow A/B} (/R)
\end{array}$$

To specify a grammar for a particular language it is enough now to give its lexicon. $Lex \subseteq \Sigma \times \mathcal{F}$ is a relation associating each word with a finite number of types. A string belongs to the language for lexicon Lex and goal type B , $w_1 \cdots w_n \in L(Lex, B)$ iff for $1 \leq i \leq n$, $(w_i, A_i) \in Lex$, and $\Gamma \Rightarrow B$ where Γ is a tree with yield $A_1 \cdots A_n$.

Buszkowski and Penn (1990) model the acquisition of lexical type assignments as a process of solving type equations. Their unification-based algorithms take function-argument structures as input (binary trees with a distinguished daughter); one obtains variations depending on whether the solution should assign a unique type to every vocabulary item, or whether one accepts multiple assignments. Kanazawa (1998) studies learnable classes of grammars from this perspective, in the sense of Gold's notion of identifiability 'in the limit'; the formal theory of learnability for type-logical grammars has recently developed into a quite active field of research.

Meaning assembly Lambek's original work looked at categorial grammar from a purely syntactic point of view, which probably explains why this work was not taken into account by Richard Montague when he developed his theory of modeltheoretic semantics for natural languages. In the 1980-ies, Van Benthem played a key role in bringing the two traditions together, by introducing the Curry-Howard perspective, with its dynamic, derivational view on meaning assembly rather than the static, structure-based view of rule-based approaches.

For semantic interpretation, we want to associate every type A with a semantic domain D_A , the domain where expressions of type A find their denotations. It is convenient to set up semantic domains via a mapping from the directional syntactic types used so far to the undirected type system of the typed lambda calculus. This indirect approach is attractive for a number of reasons. On the level of atomic types, one may want to make different basic distinctions depending on whether one uses syntactic or semantic criteria. For complex types, a map from syntactic to semantic types makes it possible to forget information that is relevant only for the way expressions are to be configured in the form dimension. For simplicity, we focus on implicational types here — accommodation of product types is straightforward.

For a simple extensional interpretation, the set of atomic semantic types could consist of types e and t , with D_e the domain of discourse (a non-empty set of entities, objects), and $D_t = \{0, 1\}$, the set of truth values. $D_{A \rightarrow B}$, the semantic domain for a functional type $A \rightarrow B$, is the set of functions from D_A to D_B . The mapping from syntactic to semantic types $(\cdot)'$ could now stipulate for basic syntactic types that $np' = e$, $s' = t$, and $n' = e \rightarrow t$. Sentences, in this way, denote truth values; (proper) noun phrases individuals; common nouns functions from individuals to truth values. For complex syntactic types, we set $(A/B)' = (B \setminus A)' = B' \rightarrow A'$. On the level of semantic types, the directionality of the slash connective is no longer taken into account. Of course, the distinction between numerator and denominator — domain and range of the interpreting functions — is kept. Below some common parts of speech with their corresponding syntactic and semantic types.

determiner	$(s/(np \backslash s))/n$	$(e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow t$
intransitive verb	$np \backslash s$	$e \rightarrow t$
transitive verb	$(np \backslash s)/np$	$e \rightarrow e \rightarrow t$
reflexive pronoun	$((np \backslash s)/np) \backslash (np \backslash s)$	$(e \rightarrow e \rightarrow t) \rightarrow e \rightarrow t$
relative pronoun	$(n \backslash n)/(np \backslash s)$	$(e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow e \rightarrow t$

Formulas-as-types, proofs as programs Curry's basic insight was that one can see the functional types of type theory as logical implications, giving rise to a one-to-one correspondence between typed lambda terms and natural deduction proofs in positive intuitionistic logic. Translating Curry's 'formulas-as-types' idea to the categorial type logics we are discussing, we have to take the differences between intuitionistic logic and the grammatical resource logic into account. Below we give the slash rules of the base logic in natural deduction format, now taking term-decorated formulas as basic declarative units. Judgements take the form of sequents $\Gamma \vdash M : A$. The antecedent Γ is a structure with leaves $x_1 : A_1, \dots, x_n : A_n$. The x_i are unique variables of type A'_i . The succedent is a term M of type A' with exactly the free variables x_1, \dots, x_n , representing a program which given inputs $k_1 \in D_{A'_1} \dots, k_n \in D_{A'_n}$ produces a value of type A' under the assignment that maps the variables x_i to the objects k_i . The x_i in other words are the parameters of the meaning assembly procedure; for these parameters we will substitute the actual lexical meaning recipes when we rewrite the leaves of the antecedent tree to terminal symbols (words). A derivation starts from axioms $x : A \vdash x : A$. The Elimination and Introduction rules have a version for the right and the left implication. On the meaning assembly level, this syntactic difference is ironed out, as we already saw that $(A/B)' = (B \backslash A)'$. As a consequence, we don't have the *isomorphic* (one-to-one) correspondence between terms and proofs of Curry's original program. But we do read off meaning assembly from the categorial derivation.

$$\frac{(\Gamma, x : B) \vdash M : A}{\Gamma \vdash \lambda x. M : A/B} I/ \quad \frac{(x : B, \Gamma) \vdash M : A}{\Gamma \vdash \lambda x. M : B \backslash A} I \backslash$$

$$\frac{\Gamma \vdash M : A/B \quad \Delta \vdash N : B}{(\Gamma, \Delta) \vdash MN : A} E/ \quad \frac{\Gamma \vdash N : B \quad \Delta \vdash M : A}{(\Gamma, \Delta) \vdash MN : A} E \backslash$$

A second difference between the programs/computations that can be obtained in intuitionistic implicational logic, and the recipes for meaning assembly associated with categorial derivations has to do with the resource management of assumptions in a derivation. In Curry's original program, the number of occurrences of assumptions (the 'multiplicity' of the logical resources) is not critical. One can make this style of resource management explicit in the form of structural rules of Contraction and Weakening, allowing for the duplication and waste of resources.

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} C \quad \frac{\Gamma \vdash B}{\Gamma, A \vdash B} W$$

In contrast, the categorial type logics are resource sensitive systems where each assumption has to be used exactly once. We have the following correspondence between resource constraints and restrictions on the lambda terms coding derivations:

1. no empty antecedents: each subterm contains a free variable;
2. no Weakening: each λ operator binds a variable free in its scope;
3. no Contraction: each λ operator binds at most one occurrence of a variable in its scope.

Taking into account also word order and phrase structure (in the absense of Associativity and Commutativity), the slash introduction rules responsible for the λ operator can only reach the immediate daughters of a structural domain.

These constraints imposed by resource-sensitivity put severe limitations on the expressivity of the derivational semantics. There is an interesting division of labour here in natural language grammars between derivational and lexical semantics. The proof term associated with a derivation is a *uniform* instruction for meaning assembly that fully abstracts from the contribution of the particular lexical items on which it is built. At the level of the lexical meaning recipes, we do not impose linearity constraints. Below some examples of non-linearity; syntactic type assignment for these words was given above. The lexical term for the reflexive pronoun is a pure combinator: it identifies the first and second coordinate of a binary relation. The terms for relative pronouns or determiners have a double bind λ to compute the intersection of their two $(e \rightarrow t)$ arguments (noun and verb phrase), and to test the intersection for non-emptiness in the case of ‘some’.

a, some (determiner)	$(e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow t$	$\lambda P \lambda Q. (\exists \lambda x. ((P\ x) \wedge (Q\ x)))$
himself (reflexive pronoun)	$(e \rightarrow e \rightarrow t) \rightarrow e \rightarrow t$	$\lambda R \lambda x. ((R\ x)\ x)$
that (relative pronoun)	$(e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow e \rightarrow t$	$\lambda P \lambda Q \lambda x. ((P\ x) \wedge (Q\ x))$

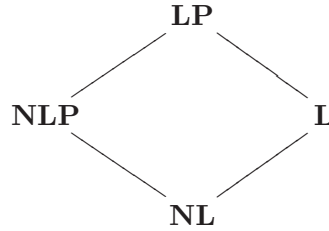
The interplay between lexical and derivational aspects of meaning assembly is illustrated with the natural deduction below. Using variables x_1, \dots, x_n for the leaves in left to right order, the proof term for this derivation is $((x_1\ x_2)\ (x_4\ x_3))$. Substituting the above lexical recipes for ‘a’ and ‘himself’ and non-logical constants **boy** ^{$e \rightarrow t$} and **hurt** ^{$e \rightarrow e \rightarrow t$} , we obtain, after β conversion, $(\exists \lambda y. ((\mathbf{boy}\ y) \wedge ((\mathbf{hurt}\ y)\ y)))$. Notice that the proof term reflects the derivational history (modulo directionality); after lexical substitution this transparency is lost. The full encapsulation of lexical semantics is one of the strong attractions of the categorial approach.

$$\frac{\frac{\frac{a}{(s/(np \backslash s))/n} \quad \frac{boy}{n}}{(a, boy) \vdash s/(np \backslash s)} (/E) \quad \frac{\frac{hurt}{(np \backslash s)/np} \quad \frac{himself}{((np \backslash s)/np) \backslash (np \backslash s)}}{(hurt, himself) \vdash np \backslash s} (\backslash E)}{((a, boy), (hurt, himself)) \vdash s} (/E)$$

(NOTE insert an example of derivational ambiguity: one structure, multiple proofs = multiple readings)

Structural variation

A second source of expressive limitations of the grammatical base logic is of a more structural nature. Consider situations where a word or phrase makes a uniform semantic contribution, but appears in contexts which the base logic cannot relate derivationally. In generative grammar, such situations are studied under the heading of ‘displacement’, a suggestive metaphor from our type-logical perspective. Displacement can be *overt* (as in the case of question words, relative pronouns and the like: elements that enter into a dependency with a ‘gap’ following at a potentially unbounded distance, cf. ‘Who do you think that Mary likes (gap)?’), or *covert* (as in the case of quantifying expressions with the ability for non-local scope construal, cf. ‘Alice thinks someone is cheating’, which can be construed as ‘there is a particular x such that Alice thinks x is cheating’). We have seen already that such expressions have higher-order types of the form $(A \rightarrow B) \rightarrow C$. The Curry-Howard interpretation then effectively dictates the uniformity of their contribution to the meaning assembly process as expressed by a term of the form $(M^{(A \rightarrow B) \rightarrow C} \lambda x^A. N^B)^C$, where the ‘gap’ is the λ bound hypothesis. What remains to be done, is to provide the fine-structure for this abstraction process, specifying which subterms of N^B are in fact ‘visible’ for the λ binder.



To work out this notion of visibility or structural accessibility, we introduce structural rules, in addition to the logical rules of the base logic studied so far. From the pure residuation logic, one obtains a hierarchy of categorial calculi by adding the structural rules of Associativity, Commutativity or both. For reasons of historical precedence, the system of Lambek (1958), with an associative composition operation, is known as **L**; the more fundamental system of Lambek (1961) as **NL**, i.e. the non-associative version of **L**. Addition of commutativity turns these into **LP** and **NLP**, respectively. For linguistic application, it is clear that *global* options of associativity and/or commutativity are too crude: they would entail that arbitrary changes in constituent structure and/or word order cannot affect wellformedness of an expression. What is needed, is a *controlled* form of structural reasoning, anchored in lexical type assignment.

Control operators The strategy is familiar from linear logic: the type language is extended with a pair of unary operators (‘modalities’). They are constants in their own right, with logical rules of use and of proof. In addition, they can provide controlled access to structural rules.

$$\mathcal{F} ::= \mathcal{A} \mid \Diamond \mathcal{F} \mid \Box \mathcal{F} \mid \mathcal{F} \backslash \mathcal{F} \mid \mathcal{F} \bullet \mathcal{F} \mid \mathcal{F} / \mathcal{F}$$

Consider the logical properties first. The truth conditions below characterize the control operators \Diamond and \Box as inverse duals with respect to a binary accessibility relation R_\diamond . This interpretation turns them into a residuated pair, just like composition and the left and right slash operations, i.e. we have $\Diamond A \longrightarrow B$ iff $A \longrightarrow \Box B$ (Res).

$$x \in V(\Diamond A) \text{ iff } \exists y. R_\diamond xy \text{ and } y \in V(A) \quad x \in V(\Box A) \text{ iff } \forall y. R_\diamond yx \text{ implies } y \in V(A)$$

We saw that for composition and its residuals, completeness with respect to the frame semantics doesn't impose restrictions on the interpretation of the merge relation R_\bullet . Similarly, for R_\diamond in the pure residuation logic of \Diamond, \Box . This means that consequences of (Res) characterize grammatical invariants, in the sense indicated above. From (Res) one easily derives the fact that the control operators are monotonic ($A \longrightarrow B$ implies $\Diamond A \longrightarrow \Diamond B$ and $\Box A \longrightarrow \Box B$), and that their compositions satisfy $\Diamond \Box A \longrightarrow A \longrightarrow \Box \Diamond A$. These properties can be put to good use in refining lexical type assignment so that selectional dependencies are taken into account. Compare the effect of an assignment A/B versus $A/\Diamond \Box B$. The former will produce an expression of type A in composition both with expressions of type B and $\Diamond \Box B$, the latter only with the more specific of these two, $\Diamond \Box B$. An expression typed as $\Box \Diamond B$ will *resist* composition with either A/B or $A/\Diamond \Box B$.

For sequent presentation, the antecedent tree structures now have unary in addition to binary branching: $\mathcal{S} ::= \mathcal{F} \mid (\mathcal{S}) \mid (\mathcal{S}, \mathcal{S})$. The residuation pattern then gives rise to the following rules of use and proof. Cut elimination carries over straightforwardly to the extended system, and with it decidability and the subformula property.

$$\begin{array}{c} \frac{\Gamma[(A)] \Rightarrow B}{\Gamma[\Diamond A] \Rightarrow B} \Diamond L \quad \frac{\Gamma \Rightarrow A}{(\Gamma) \Rightarrow \Diamond A} \Diamond R \\[10pt] \frac{\Gamma[A] \Rightarrow B}{\Gamma[(\Box A)] \Rightarrow B} \Box L \quad \frac{(\Gamma) \Rightarrow A}{\Gamma \Rightarrow \Box A} \Box R \end{array}$$

Controlled structural rules Let us turn then to use of \Diamond, \Box as control devices, providing restricted access to structural options that would be destructive in a global sense. Consider the role of the relative pronoun 'that' in the phrases below. The (a) example, where the gap hypothesis is in subject position, is derivable in the structurally-free base logic with the type-assignment given. The (b) example might suggest that the gap in object position is accessible via rebracketing of $(np, ((np \backslash s)/np, np))$ under associativity. The (c) example shows that apart from rebracketing also reordering would be required to access a non-peripheral gap.

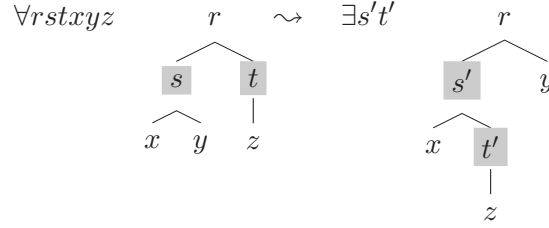
- (a) the paper that appeared today $(n \backslash n)/(np \backslash s)$
- (b) the paper that John wrote $(n \backslash n)/(s/np) + \text{Ass}$
- (c) the paper that John wrote today $(n \backslash n)/(s/np) + \text{Ass, Com??}$

The controlled structural rules below allow the required restructuring and reordering only for \Diamond marked resources. In combination with a type assignment $(n \backslash n)/(s/\Diamond \Box np)$ to the relative pronoun, they make the right branches of structural configurations accessible

for gap introduction. As long as the gap subformula $\Diamond\Box np$ carries the licensing \Diamond , the structural rules are applicable; as soon as it has found the appropriate structural position where it is selected by the transitive verb, it can be used as a regular np , given $\Diamond\Box np \longrightarrow np$.

$$(P1) \quad (A \bullet B) \bullet \Diamond C \longrightarrow A \bullet (B \bullet \Diamond C) \quad (P2) \quad (A \bullet B) \bullet \Diamond C \longrightarrow (A \bullet \Diamond C) \bullet B$$

Frame constraints, term assignment Whereas the structural interpretation of the pure residuation logic does not impose restrictions on the R_\Diamond and R_\bullet relations, completeness for structurally extended versions requires a frame constraint for each structural postulate. In the case of (P2) above, the constraint guarantees that whenever we can connect root r to leaves x, y, z via internal nodes s, t , one can rewire root and leaves via internal nodes s', t' .



As for term assignment and meaning assembly, we have two options. The first is to treat \Diamond, \Box purely as syntactic control devices. One then sets $(\Diamond A)' = (\Box A)' = A'$, and the inference rules affecting the modalities leave no trace in the term associated with a derivation. The second is to actually provide denotation domains $D_{\Diamond A}, D_{\Box A}$ for the new types, and to extend the term language accordingly. This is done in Wansing (2002), who develops a set-theoretic interpretation of minimal temporal intuitionistic logic. The temporal modalities of future possibility and past necessity are indistinguishable from the control operators \Diamond, \Box , prooftheoretically and as far as their relational interpretation is concerned, which in principle would make Wansing's approach a candidate for linguistic application.

Embedding translations A general theory of substructural communication in terms of \Diamond, \Box is worked out in Kurtonina and Moortgat (1996). Let \mathcal{L} and \mathcal{L}' be neighbours in the landscape of Figure NN. We have translations \cdot^\natural from $\mathcal{F}(/, \bullet, \backslash)$ of \mathcal{L} to $\mathcal{F}(\Diamond, \Box, /, \bullet, \backslash)$ of \mathcal{L}' such that

$$\mathcal{L} \vdash A \longrightarrow B \quad \text{iff} \quad \mathcal{L}' \vdash A^\natural \longrightarrow B^\natural$$

The \cdot^\natural translation decorates formulas of the source logic \mathcal{L} with the control operators \Diamond, \Box . The modal decoration has two functions. In the case where the target logic \mathcal{L}' is more discriminating than \mathcal{L} , it provides access to controlled versions of structural rules that are globally available in the source logic. This form of communication is familiar from the embedding theorems of linear logic, showing that no expressivity is lost by removing free duplication and deletion (Contraction/Weakening). The other direction

of communication obtains when the target logic \mathcal{L}' is less discriminating than \mathcal{L} . The modal decoration in this case blocks the applicability of structural rules that by default are freely available in the more liberal \mathcal{L} .

As an example, consider the grammatical base logic **NL** and its associative neighbour **L**. For $\mathcal{L} = \mathbf{NL}$ and $\mathcal{L}' = \mathbf{L}$, the \cdot^\natural translation below affectively removes the conditions for applicability of the associativity postulate $A \bullet (B \bullet C) \longleftrightarrow (A \bullet B) \bullet C$ (Ass), restricting the set of theorems to those of **NL**. For $\mathcal{L} = \mathbf{L}$ and $\mathcal{L}' = \mathbf{NL}$, the \cdot^\natural translation provides access to a controlled form of associativity (Ass_◊) $\diamond(A \bullet \diamond(B \bullet C)) \longleftrightarrow \diamond(\diamond(A \bullet B) \bullet C)$, the image of (Ass) under \cdot^\natural .

$$\begin{aligned} p^\natural &= p \quad (p \in \mathcal{A}) \\ (A \bullet B)^\natural &= \diamond(A^\natural \bullet B^\natural) \\ (A/B)^\natural &= \Box A^\natural / B^\natural \\ (B \setminus A)^\natural &= B^\natural \setminus \Box A^\natural \end{aligned}$$

Generative capacity, computational complexity The embedding results discussed above allow one to determine the Cartesian coordinates of a language in the logical space for diversity. Which regions of that space are actually populated by natural language grammars? In terms of the Chomsky hierarchy, recent work in a variety of frameworks has converged on the so-called mildly context-sensitive grammars: formalisms more expressive than context free, but strictly weaker than context-sensitive, and allowing polynomial parsing algorithms. The minimal system in the categorial hierarchy **NL** is strictly context-free and has a polynomial recognition problem, but, as we have seen, needs structural extensions. Such extensions are not innocent, as shown in Pentus (1993, 2003): whereas **L** remains strictly context-free, the addition of global associativity makes the derivability problem NP complete. Also for **LP**, coinciding with the multiplicative fragment of linear logic, we have NP completeness. Moreover, Van Benthem (1995) shows that **LP** recognizes the full permutation closure of context-free languages, a lack of structural discrimination making this system unsuited for actual grammar development. The situation with \diamond controlled structural rules is studied in Moot (2002), who establishes a PSPACE complexity ceiling for linear (for \bullet), non-expanding (for \diamond) structural rules via simulation of lexicalized context-sensitive grammars. The identification of tighter restrictions on allowable structure rules, leading to mildly context-sensitive expressivity, is an open problem.

For a grammatical framework assigning equal importance to syntax and semantics, strong generative capacity is more interesting than weak capacity. Tiede (1999, 2002) studies the natural deduction proof trees that form the skeleton for meaning assembly from a tree-automata perspective, arriving at a strong generative capacity hierarchy. The base logic **NL**, though strictly context-free at the string level, can assign *non-local* derivation trees, making it more expressive than context-free grammars in this respect. Normal form **NL** proof trees remain regular; the proof trees of the associative neighbour **L** can be non-regular, but do not extend beyond the expressivity of indexed grammars, generally considered to be an upper bound for the complexity of natural

language grammars.

Variants, further reading

The material discussed in this section is covered in greater depth in Moortgat and Buszkowski's chapters in the Handbook of Logic and Language. Van Benthem (1995) is an indispensable monograph for the relations between categorial derivations, type theory and lambda calculus. and for discussion of the place of type-logical grammars within the general landscape of resource-sensitive logics. Morrill (1994) provides a detailed type-logical analysis of syntax and semantics for a rich fragment of English grammar, and situates the type-logical approach within Richard Montague's Universal Grammar framework. A versatile computational tool for categorial exploration is the the grammar development environment GRAIL of Moot (2002). The kernel is a general type-logical theorem prover based on proof nets and structural graph rewriting. The system is publicly available, or can be accessed online at <http://grail.let.uu.nl>. Bernardi (2002) and Vermaat (2006) are recent PhD theses studying syntactic and semantic aspects of cross-linguistic variation for a wide variety of languages.

This section has concentrated on the Lambek-style approach to type-logical deduction. The framework of Combinatory Categorial Grammar, studied by Steedman and his co-workers, takes its inspiration more from the Curry-Feys tradition of combinatory logic. The particular combinators used in CCG are not so much selected for completeness with respect to some structural model for the type-forming operations (such as the frame semantics introduced above) but for their computational efficiency, which places CCG among the mildly context-sensitive formalisms. Steedman (2000) is a good introduction to this line of work, whereas Baldrige (2002) shows how one can fruitfully import the technique of lexically anchored modal control into the CCG framework.

Another variation elaborating on Curry's distinction between an abstract level of *tectogrammatical* organization and its concrete *phenogrammatical* realizations is the framework of Abstract Categorial Grammar (ACG, De Groote, Muskens). An abstract categorial grammar is a structure $(\Sigma_1, \Sigma_2, \mathcal{L}, s)$, where the Σ_i are higher-order linear signatures, the abstract vocabulary Σ_1 versus the object vocabulary Σ_2 , \mathcal{L} a mapping from the abstract to the object vocabulary, and s the distinguished type of the grammar. In this setting, one can model the syntax-semantics interface in terms of the abstract versus object vocabulary distinction. But one can also study the composition of natural language *syntax* from the perspective of non-directional linear implicational types, using the canonical λ term encodings of strings and trees and operations on them discussed elsewhere in this book. Expressive power for this framework can be measured in terms of the maximal order of the constants in the abstract vocabulary and of the object types interpreting the atomic abstract types. A survey of results for the ensuing complexity hierarchy can be found in De Groote (2006). Ironically, Lambek categorial grammars themselves have eluded characterization in terms of ACGs so far. Whether one approaches natural language grammars from the top (non-directional linear implications at the **LP** level) or from the bottom (the structurally-free base logic NL) of the categorial hierarchy is to a certain extent a matter of taste, reflecting the choice, for the structural regime,

between allowing everything except what is explicitly forbidden, or forbidding everything except what is explicitly allowed. The Kurtonina-Moortgat theory of structural control shows that the two viewpoints are feasible.

Chapter 7

Further Reading

- Baxter, L. D. [1976]. *The Complexity of Unification*, Dissertation, University of Waterloo.
- Bezem, M. A. [1986]. *Bar Recursion and Functionals of Finite Type*, Dissertation, University of Utrecht.
- Bezem, M. A. [1988]. Equivalence of bar recursors in the theory of functionals of finite type, *Arch. Math. Logic* **27**(2), pp. 149–160.
- Bruce, K. B., R. Di Cosmo and G. Longo [1992]. Provable isomorphisms of types, *Math. Structures Comput. Sci.* **2**(2), pp. 231–247. International Conference on Symbolic Computation (Zurich, 1990).
- Damm, W. [1982]. The IO- and OI-hierarchies, *Theoret. Comput. Sci.* **20**(2), pp. 95–207.
- Gandy, R. [1980]. An early proof of normalization by A. M. Turing, *in*: J. P. Seldin and J. R. Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York, pp. 453–457.
- Goldfarb, W. D. [1981]. The undecidability of second-order unification, *Theoretical Computer Science* **13**, pp. 225–230.
- Harrington, L. A., M. D. Morley, A. Šcedrov and S. G. Simpson (eds.) [1985]. *Harvey Friedman's research on the foundations of mathematics*, Studies in Logic and the Foundations of Mathematics 117, North-Holland Publishing Co., Amsterdam.
- Howard, W. A. [1973]. Appendix: Hereditarily majorizable function of finite type, *Metamathematical investigation of intuitionistic arithmetic and analysis*, Springer, Berlin, pp. 454–461. Lecture Notes in Math., Vol. 344.
- Howard, W.A. [1980]. The formulas-as-types notion of construction, *in*: J.P. Seldin and J.R. Hindley (eds.), *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York, pp. 479–490.

- Huet, G. P. [1975]. A unification algorithm for typed lambda-calculus, *Theoretical Computer Science* **1**, pp. 27–57.
- Joly, Th. [1997]. Un plongement de la logique classique du 2nd ordre dans AF2, *C.R. de l'Academie des Sciences de Paris* **325**(1), pp. 1–4.
- Joly, Th. [2000a]. *Codages, separabilité et représentation de fonctions en lambda-calcul simplement typé et dans d'autres systèmes de typés*, Dissertation, Paris VII.
- Joly, Th. [2000b]. Non finitely generated types and lambda-terms combinatoric representation cost, *C.R. de l'Academie des Sciences de Paris* **331**(8), pp. 581–586.
- Joly, Th. [2001a]. Constant time parallel computations in lambda-calculus, *TCS* **266**(1), pp. 975–985.
- Joly, Th. [2001b]. The finitely generated types of the λ -calculus, *Proceedings of TLCA '01*, LNCS 2044, Springer, pp. 240–252.
- Joly, Th. [2002]. About λ -definability and combinatoric generation of types, Thierry.Joly@pps.jussieu.fr.
- Joly, Th. [2003]. Encoding of the halting problem into the monster type & applications, *Proceedings of TLCA '03*, LNCS 2701, Springer, pp. 153–166.
- Joly, Th. [2005]. On λ -Definability I: the Fixed Model Problem and Generalizations of the Matching Problem., *Fundamenta Informaticae* **66**, pp. 1–17.
- Jung, A. and J. Tiuryn [1993]. A new characterization of lambda definability, *Typed lambda calculi and applications (Utrecht, 1993)*, Lecture Notes in Comput. Sci. 664, Springer, Berlin, pp. 245–257.
- Kreisel, G. [1959]. Interpretation of analysis by means of constructive functionals of finite types, in: A. Heyting (ed.), *Constructivity in Mathematics*, North-Holland, Amsterdam, pp. 101–128. *Studies in Logic and the Foundations of Mathematics*.
- Läuchli, H. [1970]. An abstract notion of realizability for which intuitionistic predicate calculus is complete, *Intuitionism and Proof Theory (Proc. Conf., Buffalo, N.Y., 1968)*, North-Holland, Amsterdam, pp. 227–234.
- Loader, R. [2001]. The undecidability of lambda definability, *Church memorial volume*, Reidel.
- Loader, R. [2003]. Higher order β matching is undecidable, *Log. J. IGPL* **11**(1), pp. 51–68.
- Padovani [1995]. On equivalence classes of interpolation equations, in: M. Dezani-Ciancaglini and G. Plotkin (eds.), *Proc.Int'l Conf. Typed Lambda Calculi and Applications (TLCA '95)*, LNCS 902, pp. 335–349.

- Padovani, V. [1994]. Atomic matching is decidable, Unpublished manuscript.
- Padovani, V. [1996a]. Decidability of all minimal models, *in*: S. Berardi and M. Coppo (eds.), *Proc. 3rd Int'l Workshop Types for Proofs and Programs (TYPES'95)*, LNCS 1158, pp. 201–215.
- Padovani, V. [1996b]. *Filtrage d'ordre supérieur*, Dissertation, Université de Paris VII. Thèse de Doctorat.
- Padovani, V. [2000]. Decidability of fourth order matching, *Mathematical Structures in Computer Science* **3**(10), pp. 361 – 372.
- Padovani, V. [2001]. Retracts in simple types, *in*: S. Abramsky (ed.), *Proc. Int'l Conf. Typed Lambda Calculi and Applications (TLCA '01)*, LNCS 2044, Springer, pp. 376–384.
- Plotkin, G. D. [1980]. Lambda-definability in the full type hierarchy, *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*, Academic Press, London, pp. 363–373.
- Pollack, R. [1994]. Closure under alpha-conversion, *in*: H. P. Barendregt and T. Nipkov (eds.), *Types for proofs and programs (Nijmegen, 1993)*, Lecture Notes in Comput. Sci. 806, Springer, Berlin, pp. 313–332.
- Statman, R. [1985]. Equality between functionals revisited, *in Harrington et al. [1985]*, North-Holland, Amsterdam, pp. 331–338.
- Statman, R. [1997]. On Cartesian monoids, *in*: D. van Dalen and M. Bezem (eds.), *Computer science logic (Utrecht, 1996)*, Lecture Notes in Comput. Sci. 1258, Springer, Berlin, pp. 446–459.
- Statman, R. [2000]. Church's lambda delta calculus, *Logic for programming and automated reasoning (Reunion Island, 2000)*, Lecture Notes in Comput. Sci. 1955, Springer, Berlin, pp. 293–307.
- Statman, R. [2004]. On the λY calculus, *Ann. Pure Appl. Logic* **130**(1-3), pp. 325–337.
- Tait, W. W. [1967]. Intensional interpretations of functionals of finite type. I, *J. Symbolic Logic* **32**, pp. 198–212.
- de Vrijer, R. [1987]. Exactly estimating functionals and strong normalization, *Indagationes Mathematicae* **49**, pp. 479–493.
- Zaionc, Marek [1987]. Word operation definable in the typed λ -calculus, *Theoret. Comput. Sci.* **52**(1-2), pp. 1–14.
- Zaionc, Marek [1989]. On the λ definable higher order Boolean operations, *Fund. Inform.* **12**(2), pp. 181–189.

Zaionc, Marek [1990]. A characterization of lambda definable tree operations, *Inform. and Comput.* **89**(1), pp. 35–46.

Zaionc, Marek [1991]. λ -definability on free algebras, *Ann. Pure Appl. Logic* **51**(3), pp. 279–300.

@ T.Joly The ksi-rule of the untyped lambda-calculus is not finitely axiomatizable, preprint (submitted to JFP)

@ T.Joly Normalization by reducibility and Bohm theorems for the typed lambda-calculus, preprint

@ T.Joly About lambda-definability and combinatoric generation of types preprint

Bibliography

- Abramsky, S., Dov M. Gabbay and T.S.E. Maibaum (eds.) [1992]. *Handbook of Logic in Computer Science, Volume 2: Background: Computational Structures*, Oxford Science Publications.
- Ackermann, W. [1928]. Zum Hilbertschen Aufbau der reellen Zahlen, *Mathematische Annalen* **99**, pp. 118–133.
- Aspinall, D. and A. Compagnoni [1996]. Subtyping dependent types, in: E. Clarke (ed.), *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, IEEE Computer Society Press, New Brunswick, New Jersey, pp. 86–97.
- Barbanera, F., M. Dezani-Ciancaglini and U. de' Liguoro [1995]. Intersection and union types: syntax and semantics, *Information and Computation* **119**, pp. 202–230.
- Barendregt, H. P. and E. Barendsen [1997]. Efficient computations in formal proofs. to appear.
- Barendregt, Henk [1974]. Pairing without conventional restraints, *Z. Math. Logik Grundlagen Math.* **20**, pp. 289–306.
- Barendregt, Henk P. [1984]. *The Lambda Calculus: its Syntax and Semantics*, revised edition, North-Holland, Amsterdam.
- Barendregt, H.P. [1992]. Lambda calculi with types, Abramsky et al. [1992], Oxford University Press, pp. 117–309.
- Barendregt, H.P. [1996]. The quest for correctness, *Images of SMC Research*, Stichting Mathematisch Centrum, P.O. Box 94079, 1090 GB Amsterdam, pp. 39–58.
- Barendregt, H.P., M. Bunder and W. Dekkers [1993]. Systems of illative combinatory logic complete for first order propositional and predicate calculus, *The Journal of Symbolic Logic* **58**(3), pp. 89–108.
- Berarducci, Alessandro and Corrado Bohm [1993]. A self-interpreter of lambda calculus having a normal form, *Lecture Notes in Computer Science* **702**, pp. 85–99.

- Bezem, M. and J.F. Groote (eds.) [1993]. *Typed Lambda Calculi and Applications, TLCA'93*, number 664 in *Lecture Notes in Computer Science*, Springer Verlag, Berlin.
- Bezem, M.A. [1985]. Strong normalization of bar recursive terms without using infinite terms, *Archiv für Mathematische Logik und Grundlagenforschung* **25**, pp. 175–181.
- Böhm, C. [1963]. The CUCH as a formal and description language, *in*: Richard Goodman (ed.), *Annual Review in Automatic Programming, Vol. 3*, Pergamon Press, Oxford, pp. 179–197.
- Böhm, C. and W. Gross [1966]. Introduction to the CUCH, *in*: E.R. Caianiello (ed.), *Automata Theory*, Academic Press, New York, pp. 35–65.
- Böhm, C., A. Piperno and S. Guerrini [1994]. lambda-definition of function(al)s by normal forms, *in*: D. Sanella (ed.), *ESOP'94*, Vol. 788, Springer, Berlin, pp. 135–154.
- Böhm, Corrado and Alessandro Berarducci [1985]. Automatic synthesis of typed Λ -programs on term algebras, *Theoretical Computer Science* **39**, pp. 135–154.
- de Bruijn, N. G. [1970]. The mathematical language AUTOMATH, its usage and some of its extensions, *in*: M. Laudet, D. Lacombe and M. Schuetzenberger (eds.), *Symposium on Automatic Demonstration*, Springer Verlag, Berlin, 1970, IRIA, Versailles, pp. 29–61. *Lecture Notes in Mathematics* **125**; also in Nederpelt et al. [1994].
- de Bruijn, N. G. [1972]. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem, *Nederl. Akad. Wetensch. Proc. Ser. A* **75**=*Indag. Math.* **34**, pp. 381–392.
- de Bruijn, N.G. [1990]. Reflections on Automath, Eindhoven University of Technology. Also in Nederpelt et al. [1994], pp 201–228.
- Capretta, V. and S. Valentini [1998]. A general method for proving the normalization theorem for first and second order typed λ -calculi, *Mathematical Structures in Computer Science*. To appear.
- Church, A. [1940]. A formulation of the simple theory of types, *The Journal of Symbolic Logic* **5**, pp. 56–68.
- Church, Alonzo [1936]. An unsolvable problem of elementary number theory, *American Journal of Mathematics* **58**, pp. 354–363.
- Comon, Hubert and Yan Jurski [1998]. Higher-order matching and tree automata, *Computer science logic (Aarhus, 1997)*, *Lecture Notes in Comput. Sci.* 1414, Springer, Berlin, pp. 157–176.

- Coquand, Thierry and Gérard Huet [1988]. The Calculus of Constructions, *Information and Computation* **76**(2/3), pp. 95–120.
- Curry, H.B. [1934]. Functionality in combinatory logic, *Proceedings of the National Academy of Science of the USA* **20**, pp. 584–590.
- Davis, M. [1973]. Hilbert’s tenth problem is unsolvable, *American Mathematical Monthly* **80**, pp. 233–269.
- Davis, M., J. Robinson and H. Putnam [1960]. The decision problem for exponential Diophantine equations, *Annals of Mathematics*.
- Dekkers, W., M. Bunder and H.P. Barendregt [1997]. Completeness of the propositions-as-types interpretation of intuitionistic logic into illative combinatory logic, *The Journal of Symbolic Logic*.
- Dekkers, Wil [1988]. Reducibility of types in typed lambda calculus. Comment on: “On the existence of closed terms in the typed λ -calculus, I” (Statman [1980a]), *Inform. and Comput.* **77**(2), pp. 131–137.
- Dowek, Gilles [1994]. Third order matching is decidable, *Ann. Pure Appl. Logic* **69**(2-3), pp. 135–155. Invited papers presented at the 1992 IEEE Symposium on Logic in Computer Science (Santa Cruz, CA).
- van Draanen, J.-P. [1995]. *Models for simply typed lambda-calculi with fixed point combinators and enumerators*, Dissertation, Catholic University of Nijmegen.
- Dyckhoff, R. and L. Pinto [1997]. Permutability of proofs in intuitionistic sequent calculi, *Theoretical Computer Science*. To appear.
- Elbers, H. [1996]. Personal communication.
- Euclid [n.d.]. The Elements, 325 B.C. English translation in Heath [1956].
- Flajolet, P. and R. Sedgewick [1993]. The average case analysis of algorithms: counting and generating functions, *Technical Report 1888*, INRIA.
- Friedman, H.M. [1975]. Equality between functionals, in: R. Parikh (ed.), *Logic Colloquium*, Lecture Notes in Mathematics 453, Springer, Berlin, New York.
- Gandy, R. [1980a]. An early proof of normalization by A. M. Turing, in: J. P. Seldin and J. R. Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York, pp. 453–457.
- Gandy, R.O. [1980b]. Church’s Thesis and principles for mechanisms, *The Kleene Symposium*, North-Holland Publishing Company, Amsterdam, pp. 123–148.
- Gentzen, G. [1936]. Untersuchungen über das logischen Schliessen, *Mathematische Zeitschrift* **39**, pp. 405–431. Translation in: *Collected papers of Gerhard Gentzen*, ed. M. E. Szabo, North-Holland, Amsterdam [1969], pp. 68–131.

- Gentzen, Gerhard [1969]. Investigations into logical deduction, *in*: M. E. Szabo (ed.), *The Collected Papers of Gerhard Gentzen*, North-Holland.
- Girard, J.-Y. [1972]. Interpretation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse D'Etat, Université Paris VII.
- Girard, Jean-Yves [1971]. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types, *Proceedings of the Second Scandinavian Logic Symposium (Univ. Oslo, Oslo, 1970)*, North-Holland, Amsterdam, pp. 63–92. *Studies in Logic and the Foundations of Mathematics*, Vol. 63.
- Gödel, K. [1931]. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatshefte für Mathematik und Physik* **38**, pp. 173–198. German; English translation in van Heijenoort [1967], pages 592–618.
- Gödel, K. [1958]. Ueber eine bisher noch nicht benützte Erweiterung des finiten Standpunktes, *Dialectica* **12**, pp. 280–287.
- Goldfarb, Warren D. [1981]. The undecidability of the second-order unification problem, *Theoret. Comput. Sci.* **13**(2), pp. 225–230.
- Harrington, L. A., M. D. Morley, A. Šcedrov and S. G. Simpson (eds.) [1985]. *Harvey Friedman's research on the foundations of mathematics*, *Studies in Logic and the Foundations of Mathematics* 117, North-Holland Publishing Co., Amsterdam.
- Heath, T.L. [1956]. *The Thirteen Books of Euclid's Elements*, Dover Publications, Inc., New York.
- van Heijenoort, J. (ed.) [1967]. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, Harvard University Press, Cambridge, Massachusetts.
- Herbelin, H. [1995]. A lambda calculus structure isomorphic to Gentzen-style sequent calculus structure, *Computer Science Logic (CSL'94)*, *Lecture Notes in Computer Science* **933**, Springer Verlag, Berlin, pp. 61–75.
- Hindley, J.R. [1997]. *Basic Simple Type Theory*, Cambridge University Press, Cambridge, UK.
- Hofmann, M. [1977]. A simple model for quotient types, *Typed lambda calculi and applications*, *Lecture Notes in Computer Science*, Springer, Berlin and New York, pp. 216–234.
- Howard, W. A. [1970]. Assignment of ordinals to terms for primitive recursive functionals of finite type, *in*: J. Myhill A. Kino and R.E. Vesley (eds.), *Intuitionism and Proof Theory, Proceedings of the summer conference at Buffalo N.Y. 1968*, *Studies in logic and the foundations of mathematics*, North-Holland, Amsterdam, pp. 443–458.

- Jacopini, Giuseppe [1975]. A condition for identifying two elements of whatever model of combinatory logic, *λ -calculus and computer science theory (Proc. Sympos., Rome, 1975)*, Springer, Berlin, pp. 213–219. Lecture Notes in Comput. Sci., Vol. 37.
- Joly, Th. [2001]. Constant time parallel computations in lambda-calculus, *TCS* **266**(1), pp. 975–985.
- Joly, Th. [2002]. About λ -definability and combinatoric generation of types, Thierry.Joly@pps.jussieu.fr.
- Joly, Th. [2005]. On λ -Definability I: the Fixed Model Problem and Generalizations of the Matching Problem., *Fundamenta Informaticae* **66**, pp. 1–17.
- Jones, J. P. [1982]. Universal Diophantine equation, *J. Symbolic Logic* **47**(3), pp. 549–571.
- Jutting, L.S. van Benthem [1977]. *Checking Landau's "Grundlagen" in the AUTOMATH System*, Dissertation, Eindhoven University of Technology.
- Kleene, S. C. [1959a]. Countable functionals, *Constructivity in mathematics: Proceedings of the colloquium held at Amsterdam, 1957 (edited by A. Heyting)*, Studies in Logic and the Foundations of Mathematics, North-Holland Publishing Co., Amsterdam, pp. 81–100.
- Kleene, S. C. [1959b]. Recursive functionals and quantifiers of finite types. I, *Trans. Amer. Math. Soc.* **91**, pp. 1–52.
- Klop, J.W. [1980]. *Combinatory Reduction Systems*, Dissertation, Utrecht University. Appeared as Mathematical Centre Tracts 127, Kruislaan 413, 1098 SJ Amsterdam.
- Kreisel, G. [1959]. Interpretation of analysis by means of constructive functionals of finite types, in: A. Heyting (ed.), *Constructivity in Mathematics*, North-Holland, Amsterdam, pp. 101–128. Studies in Logic and the Foundations of Mathematics.
- Lambek, J. [1980]. From λ -calculus to Cartesian closed categories, in: *Seldin and (Eds.) [1980]*, Academic Press, London, pp. 375–402.
- Landau, E. [1960]. *Grundlagen der Analysis*, 3-rd edition edition, Chelsea Publishing Company.
- Loader, R. [1997]. An algorithm for the minimal model, Unpublished manuscript, obtainable from homepages.ihug.co.nz/~suckfish/papers/papers.html.
- Loader, R. [2001a]. Finitary PCF is not decidable, *TCS* **266**(1-2), pp. 341–364.
- Loader, R. [2001b]. The undecidability of lambda definability, *Church memorial volume*, Reidel.

- Loader, R. [2003]. Higher order β matching is undecidable, *Log. J. IGPL* **11**(1), pp. 51–68.
- Luo, Zhaohui and Robert Pollack [1992]. The LEGO proof development system: A user's manual, *Technical Report ECS-LFCS-92-211*, University of Edinburgh.
- Mairson, Harry G. [1992]. A simple proof of a theorem of Statman, *Theoret. Comput. Sci.* **103**(2), pp. 387–394.
- Makanin, G.S. [1977]. The problem of solvability of equations in a free semigroup, *Mathematics of the USSR-Sbornik* **32**, pp. 129–198.
- Martin-Löf, P. [1984]. *Intuitionistic Type Theory*, Studies in Proof Theory, Bibliopolis, Napoli.
- Matijasevič, Yu. V. [1971]. Diophantine representation of recursively enumerable predicates, *Proceedings of the Second Scandinavian Logic Symposium (Univ. Oslo, Oslo, 1970)*, Studies in Logic and the Foundations of Mathematics, Vol. 63, North-Holland, Amsterdam, pp. 171–177.
- Matijasevič, Yuri V. [1993]. *Hilbert's tenth problem*, Foundations of Computing Series, MIT Press, Cambridge, MA. Translated from the 1993 Russian original by the author, With a foreword by Martin Davis.
- Mayr, Richard and Tobias Nipkow [1998]. Higher-order rewrite systems and their confluence, *Theoret. Comput. Sci.* **192**(1), pp. 3–29. Rewriting systems and applications (Kaiserslautern, 1995).
- Mints, G. [1996]. Normal forms for sequent derivations, in: P. Odifreddi (ed.), *Kreiseliana. About and Around Georg Kreisel*, A.K. Peters, Wellesley, Massachusetts, pp. 469–492.
- Nederpelt, R.P., J.H. Geuvers and R.C. de Vrijer (eds.) [1994]. *Selected Papers on Automath*, Studies in Logic and the Foundations of Mathematics **133**, North-Holland, Amsterdam.
- Oostdijk, M. [1996]. *Proof by calculation*, Master's thesis, 385, Universitaire School voor Informatica, Catholic University Nijmegen.
- Padovani, V. [1996]. *Filtrage d'ordre supérieur*, Dissertation, Université de Paris VII. Thèse de Doctorat.
- Padovani, V. [2000]. Decidability of fourth order matching, *Mathematical Structures in Computer Science* **3**(10), pp. 361 – 372.
- Paulin-Mohring, C. [1994]. Inductive definitions in the system coq; rules and properties, *Bezem and Groote [1993]*, number 664 in *Lecture Notes in Computer Science*, Springer Verlag, Berlin, pp. 328–345.

- Péter, R. [1967]. *Recursive functions*, third revised edition edition, Academic Press, New York.
- Platek, R.A. [1966]. *Foundations of recursions theory*, Dissertation, Stanford University.
- Plotkin, G. [1977]. LCF considered as a programming language, *Theoretical Computer Science* **5**, pp. 225–255.
- Plotkin, G. [1985?]. Personal communication, email.
- Poincaré, H. [1902]. *La Science et l'Hypothèse*, Flammarion, Paris.
- Post, E. [1947]. Recursive unsolvability of a problem of thue, *Journal of Symbolic Logic* **12**(1), pp. 1–11.
- Pottinger, G. [1977]. Normalization as a homomorphic image of cut-elimination, *Annals of Mathematical Logic* **12**, pp. 323–357.
- Pottinger, G. [1981]. The church-rosser theorem for the typed λ -calculus with surjective pairing, *Notre Dame J. Formal Logic* **22**(3), pp. 264–268.
- Prawitz, D. [1965]. *Natural Deduction*, Almqvist & Wiksell, Stockholm.
- Prawitz, D. [1971]. Ideas and results in proof theory, *in*: J. E. Fenstad (ed.), *Proceedings of the 2nd Scandinavian Logic Symposium*, North-Holland, Amsterdam, pp. 235–307.
- Raamsdonk, F. van [1996]. *Confluence and Normalisation for Higher-Order Rewriting*, Dissertation, Vrije Universiteit, Amsterdam, The Netherlands.
- Ramsey, F.P. [1925]. The foundations of mathematics, *Proceedings of the London Mathematical Society* pp. 338–384.
- Russell, Bertrand A.W. and Alfred North Whitehead [1910–13]. *Principia Mathematica*, Cambridge University Press.
- Schmidt-Schauß, Manfred [1999]. Decidability of behavioural equivalence in unary PCF, *Theoret. Comput. Sci.* **216**(1-2), pp. 363–373.
- Schwichtenberg, H. [1997]. Termination of permutative conversion in intuitionistic Gentzen calculi, *Theoretical Computer Science*. To appear.
- Schwichtenberg, H. and U. Berger [1991]. An inverse of the evaluation functional for typed λ -calculus, *Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science*, IEEE, pp. 203–211.
- Schwichtenberg, Helmut [1975]. Elimination of higher type levels in definitions of primitive recursive functionals by means of transfinite recursion, *Logic Colloquium '73 (Bristol, 1973)*, North-Holland, Amsterdam, pp. 279–303. *Studies in Logic and the Foundations of Mathematics*, Vol. 80.

- Scott, D.S. [1970]. Constructive validity, *in*: D. Lacombe M. Laudet and M. Schuetzenberger (eds.), *Symposium on Automated Demonstration*, Lecture Notes in Mathematics 125, Springer, Berlin, pp. 237–275.
- Scott, D.S. [1980]. Relating theories of the λ -calculus, *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*, Academic Press, pp. 403–450.
- Seldin, J.P. and J.R. Hindley (Eds.) [1980]. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press.
- Severi, P. and E. Poll [1994]. Pure type systems with definitions, *in*: A. Nerode and Yu.V. Matiyasevich (eds.), *Proceedings of LFCS'94 (LNCS 813)*, LFCS'94, St. Petersburg, Russia, Springer Verlag, New York, pp. 316–328.
- Spector, C. [1962]. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles formulated in current intuitionistic mathematics, *in*: J.C.E. Dekker (ed.), *Recursive function theory, Proc. Symp. in pure mathematics V*, AMS, Providence, pp. 1–27.
- Statman, R. [1982]. Completeness, invariance and λ -definability, *J. Symbolic Logic* **47**(1), pp. 17–26.
- Statman, R. [1985]. Equality between functionals revisited, *in Harrington et al. [1985]*, North-Holland, Amsterdam, pp. 331–338.
- Statman, R. [2000]. Church's lambda delta calculus, *Logic for programming and automated reasoning (Reunion Island, 2000)*, Lecture Notes in Comput. Sci. 1955, Springer, Berlin, pp. 293–307.
- Statman, Richard [1979]. The typed λ -calculus is not elementary recursive, *Theoret. Comput. Sci.* **9**(1), pp. 73–81.
- Statman, Richard [1980a]. On the existence of closed terms in the typed λ -calculus. I, *in Seldin and (Eds.) [1980]*, Academic Press, London, pp. 511–534.
- Statman, Richard [1980b]. On the existence of closed terms in the typed λ -calculus. III, Dept. of Mathematics, CMU, Pittsburgh, USA.
- Tait, W. W. [1967]. Intensional interpretations of functionals of finite type. I, *J. Symbolic Logic* **32**, pp. 198–212.
- Tait, William W. [1965]. Infinitely long terms of transfinite type I, *in*: J. Crossley and M. Dummett (eds.), *Formal Systems and Recursive Functions*, North-Holland, pp. 176–185.
- Tait, W.W. [1971]. Normal form theorem for barrecursive functions of finite type, *in*: J.E. Fenstad (ed.), *Proceedings of the 2nd Scandinavian Logic Symposium*, North-Holland, Amsterdam, pp. 353–367.

- Terlouw, Jan [1982]. On definition trees of ordinal recursive functionals: reduction of the recursion orders by means of type level raising, *J. Symbolic Logic* **47**(2), pp. 395–402.
- Thatcher, J.W. [1973]. *Tree automata: an informal survey*, Vol. Currents in the Theory of Computing, Prentice-Hall, chapter 4, pp. 143–172.
- Troelstra, A. S. [1998]. Marginalia on sequent calculi, *Studia Logica*. To appear.
- Troelstra, A. S. and H. Schwichtenberg [1996]. *Basic Proof Theory*, Cambridge University Press, Cambridge, UK.
- Troelstra, A.S. [1973]. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, number 344 in *Lecture Notes in Mathematics*, Springer-Verlag, Berlin.
- Vogel, H. [1976]. Ein starker Normalisationssatz für die barrekursiven Funktionale, *Archiv für Mathematische Logik und Grundlagenforschung* **18**, pp. 81–84.
- de Vrijer, R. [1987]. Exactly estimating functionals and strong normalization, *Indagationes Mathematicae* **49**, pp. 479–493.
- Waite, W. and G. Goos [1984]. *Compiler Construction*, Springer.
- Wells, J. B. [1999]. Typability and type checking in system F are equivalent and undecidable, *Annals of Pure and Applied Logic* **98**(1-3), pp. 111–156.
- van Wijngaarden, A., B.J. Mailloux, J.E.L. Peck, C.H.A. Koster, M. Sintzoff, C.H. Lindsey, L.G.L.T. Meertens and R.G. Fisker (eds.) [1976]. *Revised Report on the Algorithmic Language ALGOL 68*, Springer Verlag, Berlin.
- Zucker, J. [1974]. Cut-elimination and normalization, *Annals of Mathematical Logic* **7**, pp. 1–112.