

An overview of Simple Type Theory

GRADUATE SEMINAR
A.Y. 2016-2017

Cosimo Perini

Contents

Introduction	2
1 Type-free λ-calculus	4
1.1 $\lambda\beta$ -calculus	4
1.2 $\lambda\beta\eta$ -calculus	7
2 Simple Type Theory	7
2.1 Basics of λ_{\rightarrow}^A	8
2.2 Principal Type Algorithm for λ_{\rightarrow}^A	10
2.3 Church's typing	13
2.4 Weak Normalization Theorem	15
2.5 Strong Normalization Theorem	17
3 Curry-Howard Isomorphism	19
3.1 Implicational fragment of \mathbb{I}	19
3.2 Substructural Logics	21
3.3 Intuitionistic Proofs as Programs	22
4 Gödel's System T	25
4.1 Preliminary Definitions	25
4.2 Normalization	26
4.3 Kreisel's Modified Realizability	27
4.4 Expressive Power	31
References	32

Abstract

This work presents basic results in simple type theory, starting with a type-assignment system with arrow-type only. Type checking for this theory is discussed and an algorithm is given in proving the decidability of the problem. Termination of computations is proved both in weak and strong sense using standard techniques. Correspondence between intuitionistic propositional logic and richer typed systems is briefly investigated and some remarks on structural proof theory and typability are made in parallel. We then prove strong normalization for Gödel's *Dialectica* system. Expressive power of this theory is finally shown proving the correspondence between provability in Heyting Arithmetic and typability in this typed system, and stating the equivalence of definable functions with primitive recursive functions provably total in first-order arithmetic.

Keywords Simple Type Theory · Normalization · Proofs into Programs

Introduction

λ -calculus was proposed in the early 1930s by Alonzo Church as a foundation of logic and mathematics in which the concept of *abstraction* is taken as primitive. In 1936, Church's students Stephen Kleene and Barkley Rosser proved that the 1933 version of this system was inconsistent, but the pure formulation turned out to be surprisingly rich.

Progressively, pure λ -calculus revealed to be a successful model for the concept of *effectively computable function*, and, although the class of λ -definable numerical functions was proved to be identical with that of Turing computable ones, it differs from Turing model being what is now known as a (higher order) functional programming language, while Turing machines resemble programs in imperative programming languages.

Typed versions of λ -calculus were introduced first by Haskell Curry in 1934 (for the related system of combinatory logic in fact) and by Church in 1940, after he had given a series of lectures attended by Alan Turing during his doctorate studies in Princeton. Types are syntactic objects which can be assigned to λ -terms, providing a partial specification of the represented algorithm. Moreover, types are useful to check the safety of the very algorithms coded as λ -terms, and to improve the efficiency of this coding. In spite of this, Curry's and Church's works on typed systems were characterized by two different approaches to typing λ -terms:

- Curry assumed terms of type-free λ -calculus, and associated to each term a set of possible types;
- Church adopted an annotated version of λ -terms, so that there exist different terms of different type, although they are "morphologically identical".

Curry's approach, which here is called 'type assignment', corresponds to the programming paradigm of writing a program without typing at all, and then to check by means of a compiler whether this program can receive a type (and this will happen if the program is correct). This is what is usually called *implicit typing*.

Church's typing, on the other hand, corresponds to the programming paradigm of writing a program together with its type, so that the type-checking results easier. In literature this is known as *explicit typing*; systems with this kind of typing will here be called 'typed system'.

Several versions of λ -calculus with types have been developed within both paradigms, and the evolution of this kind of systems has been (and is being) so widespread that we decided to focus our exposition on some simple type theories only, excluding, in spite of their relevance, many extensions, as Girard's system F, or Martin-Löf dependent type theory.¹

¹[14] gives a friendly introduction to F including an exposition of a categorial semantics for this *polymorphic λ -calculus*; it corresponds to second-order logic and dates back to 1970. By means of a slightly modified computability predicate, Jean-Yves Girard proved SN for F and its extension F_ω , which corresponds to Takeuti's GLC sequent-calculus for higher-order logic: Girard's proofs of normalization for these systems solved also Takeuti's conjecture about cut-

Types were in fact introduced first in Russell and Whitehead’s *Principia Mathematica*, and since then no clear definition of what kind of object a type should be: From grammatical categories to spaces in the sense of homotopy theory, various but closely interrelated interpretations have been given.²

We decided not to give any specific semantic interpretation of λ -calculi we investigate, but, since its relevance for proof theory and influence in development of the whole subject, we will briefly discuss the correspondence between types and provable formulae of intuitionistic logic known as Curry-Howard isomorphism, to which Sect.3 is committed. For the reader’s sake, we give here a partial glossary to summarize this point of view:

TYPE THEORY	LOGIC
type	proposition
term	proof
type constructor	logic connective
constructor	introduction rule
destructor	elimination rule
redex	proof detour
reduction	normalization
normal form	normal proof
inhabitation	provability

After a concise review of untyped λ -calculus in Sect.1, we begin our exposition of simplest type system λ_{\rightarrow}^A , starting with a formal definition of type assignment, for which an algorithm of type checking is given in Sect.2.2. The existence of a terminating strategy for λ_{\rightarrow}^A -terms reduction is proved in Sect.2.4 by means of a Weak Normalization Theorem (WN) for the corresponding typed system defined in the previous section. We organize the proof following Turing’s proof, originally typewritten before 1942; it is probably the very first proof of this theorem.

A stronger result, namely that *all* reductions do terminate, is proved in the subsequent section using Tait’s *computability predicate*.³ This method is now

elimination in GLC, and (progressively extended) have become standard techniques in proof theory (see [12]).

Per Martin-Löf Type Theory dates back to 1971, and, after various revisions during subsequent years, consists of an intuitionistic theory of iterated inductive definitions which had widespread conceptual influence on logically-oriented programming-languages, and it is the basis of many contemporary computer proof-assistants as Coq (see [7] for an overview of Martin-Löf’s approach, and [27] for the philosophical relevance of inductive types in constructive mathematics). In dependent type theory, the correspondence between types and propositions is more systematically exploited than in systems we investigate in the present work, and this intuitionistic version of typed systems revealed interesting connections with homotopy theory and higher category theory, catalysing the development of homotopical interpretation of types and indirectly legitimizing, from a constructive point of view, the univalent approach to foundations of mathematics (see [6] for relevant results in this direction).

²See [23] for a first “type-theoretic” approach to grammatical categories. [25] gives a concise history of set-theoretic interpretation of typed systems and of their “standard models”. [24] introduced first the notions of *hyperdoctrine* and *elementary topos* as models for λ -abstraction; for a unified exposition of logic and type theory by means of fibred category theory see [19]. [34] is a comprehensive account of the correspondence between various λ -calculi and systems of logic and arithmetic. For an introduction to the (now flourishing) field of Homotopy Type Theory see [39].

³According to [37], the idea of a computability predicate had been anticipated by Kurt Gödel in unpublished notes and lectures given in 1941.

considered standard, thus we adopt it to prove the analogous result for Gödel system \mathbb{T} , which we investigate in Sect.4.

This system can be considered an interpretation of first-order arithmetic in a logic-free theory of primitive recursive functionals of finite type: In [15] the consistency of arithmetic is deduced by finitary reasoning from an assumption that all closed terms in \mathbb{T} of type Int compute to unique numerals. Here the same result is obtained by means of *modified realizability* due to Georg Kreisel and, independently, A.G. Dragalin⁴ applied to the “modern version” of \mathbb{T} , explicitly defined here as a λ -calculus with integers and a recursion operator.⁵ Finally we show how terms of this system can be thought of as representing the computational contents of arithmetical reasoning. *Via* modified realizability, the class of \mathbb{T} -definable functions is proved to be identical with that of provably total functions of first-order arithmetic, and provability in Heyting Arithmetic is proved to correspond to type assignment in \mathbb{T} : These are the main results of Sects.4.3 and 4.4.

No aim of completeness leads the present work: It is nearly intended to be a (concise) primer on a still flourishing interdisciplinary subject.

1 Type-free λ -calculus

We remind here some basic definitions and results concerning β -reduction of untyped λ -terms; moreover some remarks concerning $\beta\eta$ -reduction are expressed.

These tools will be useful throughout the work, but the present section will not be a complete exposition of the matter.⁶

1.1 $\lambda\beta$ -calculus

Definition 1.1. We assume an infinite sequence of term-variables is given. Then the set of terms of type-free λ -calculus is defined inductively by the following grammar:

$$M ::= x \mid PQ \mid \lambda x.P$$

- PQ is called application;
- x is called atom (and it is a term-variable);
- $\lambda x.P$ is called abstraction.

Any term which is not an atom is called composite.

Notational Convention 1. Throughout the present work

- Term-variables are denoted by x, y, z, \dots ;
- Arbitrary terms are denoted by M, N, P, \dots ;

⁴See [22] and [10] respectively.

⁵This style of presenting \mathbb{T} comes from seminal papers by A. Grzegorzczuk [16] and Dragalin [9], and it differs from Gödel’s *Dialectica* paper for the absence, in the latter, of any recursor nor issue of normalization. Note also that Gödel’s works which William Tait refers to do not mention this issue at all, and the very Tait’s technique was introduced first to prove WN only, despite almost no change was necessary in [36] to obtain the stronger result.



⁶See [18] for a clear introduction to the topic.

- $M \equiv N$ means syntactic identity;
- We assume that application associates to the left and abstraction associates to the right.

Definition 1.2. The length $|M|$ of a term M is the number of occurrences of variables in M . Formally:

- $|x| = 1$;
- $|NP| = |N| + |P|$;
- $|\lambda x.N| = 1 + |N|$.

Notational Convention 2. We identify terms with their tree-structure and we will work on them modulo this convention:

TERM	TREE
x	x
MN	
$\lambda x.M$	

Definition 1.3. In accordance to our notation,

- A term N is a subterm of M iff the tree corresponding to N is a subtree of the tree corresponding to M .
- A variable x in a term M is bound by the first λx above x , and if there is no λx above some x , that x is said to be free in M .
- The set of free variables in a term is defined inductively:
 - $FV(x) := x$;
 - $FV(MN) := FV(M) \cup FV(N)$;
 - $FV(\lambda x.M) := FV(M) \setminus \{x\}$.

A term M is said to be closed when $FV(M) = \emptyset$.

- The substitution of M for all free occurrences of x in N is defined inductively:
 - $x[M/x] := M$;
 - $y[M/x] := y$ if $y \neq x$;
 - $NP[M/x] := N[M/x]P[M/x]$;
 - $(\lambda x.N)[M/x] := \lambda x.N$;

- $(\lambda y.N)[M/x] := \lambda y.N$ if $x \notin FV(N)$;
- $(\lambda y.N)[M/x] := \lambda y.N[M/x]$ if $x \in FV(N)$ and $y \notin FV(M)$;
- $(\lambda y.N)[M/x] := \lambda z.N[M/x][z/y]$ if $x \in FV(N)$, $y \in FV(M)$ and $z \notin FV(M) \cup FV(N)$.

Simultaneous substitution is defined as a natural extension.

- If $y \notin FV(M)$, then the act of replacing an occurrence of $\lambda x.M$ in a term by $\lambda y.M[y/x]$ is called change of bound variables. If N changes to P by a finite series of such changes, we say N α -converts to P : $N \equiv_\alpha P$.

We will work on terms modulo α -conversion; some basic facts about \equiv_α are given in [18, 1B].

Definition 1.4. A β -redex is any term $(\lambda x.M)N$; its contractum is $M[N/x]$. β -contraction \rightarrow_β is thus defined:

- $(\lambda x.M)N \rightarrow_\beta M[N/x]$;
- if $M \rightarrow_\beta M'$ then $(MN) \rightarrow_\beta (M'N)$;
- if $M \rightarrow_\beta M'$ then $(NM) \rightarrow_\beta (NM')$;
- if $M \rightarrow_\beta M'$ then $(\lambda x.M) \rightarrow_\beta (\lambda x.M')$.

Reflexive-transitive closure of \rightarrow_β modulo \equiv_α is called β -reduction and it is denoted by \twoheadrightarrow_β .

Finally we say M β -converts to N ($M =_\beta N$) when it is possible to change M to N by a finite sequence of β -reductions and reversed β -reductions (β -expansion).

We can now state the main results concerning untyped λ -calculus; we shall not prove them here, but various proofs are given in several places in literature.⁷ First remind the following

Definition 1.5. A β -normal form (β -nf) is any term which contains no β -redex. The class of all β -nf's is here denoted by NF_β . We say a term M has β -nf N iff $M \twoheadrightarrow_\beta N$ and $N \in NF_\beta$.

Theorem 1.1.1 (Church-Rosser Theorem).

- (i) If $M \twoheadrightarrow_\beta N$ and $M \twoheadrightarrow_\beta N'$, then there exists a P such that $N \twoheadrightarrow_\beta P$ and $N' \twoheadrightarrow_\beta P$.
- (ii) If $M =_\beta N$ then there exists a P such that $M \twoheadrightarrow_\beta P$ and $N \twoheadrightarrow_\beta P$.

Corollary 1.1.2 (Uniqueness of β -nf). A term M has at most one β -nf modulo α -conversion.

Remark 1. Using only tools we have so far defined it is quite easy to prove, by induction on $|M|$, that every β -nf M can be expressed uniquely as

$$M \equiv \lambda x_1 \dots x_m. y M_1 \dots M_n \quad (m \geq 0, n \geq 0)$$

where $M_1 \dots M_n$ are β -nf's and y may be one of the x_i .

We will refer to this fact as to *head-normal form lemma*.

⁷See e.g. [3, 2.3.10-20] for a "neat chain" of propositions leading to these results.

1.2 $\lambda\beta\eta$ -calculus

Church-Rosser theorem can be extended to $\beta\eta$ -reduction, which is naturally defined by allowing η -contractions

$$\lambda x.Mx \rightarrow_{\eta} M \quad x \notin FV(M)$$

in any sequence of β -contractions.

$=_{\beta\eta}$ is thus defined in the obvious way.

We will use the expression $M \twoheadrightarrow_{\beta\eta} N$ when there is a finite sequence of $\beta\eta$ -contractions (modulo \equiv_{α}) changing M to N .

Remark 2. All η -reductions are finite. In particular, any η -reduction $M \twoheadrightarrow_{\eta} N$ has length⁸ $\leq |M|/2$, for any η -contraction reduces $|M|$ to $|M| - 2$, so the set $\{M\}_{\eta}$ of all N such that $M \twoheadrightarrow_{\eta} N$ is always finite.

Definition 1.6. A $\beta\eta$ -redex is any term of the form $(\lambda x.M)N$ (β -redex) or $\lambda x.Mx$ (η -redex).

A term M is a $\beta\eta$ -normal form if it has no $\beta\eta$ -redex among its subterms.

Having a $\beta\eta$ -nf is defined as the natural extension of having a β -nf.

As before, Church-Rosser theorem assures, for a given M , the uniqueness of its $\beta\eta$ -nf; we will denote it by $M_{*\beta\eta}$. Finally we remind the following

Fact 1.2.1. [2, 15.1.5] *A term has a $\beta\eta$ -nf iff it has a β -nf.*

Remark 3. Not every term has a β -nf: consider, for instance,

$$(\lambda x.xx)(\lambda x.xx).$$

Moreover the set of terms with β -nf is not recursive ([2, 6.6.5]), thus we cannot have an algorithm to decide whether a term M has a β -nf. In spite of this, we can define a reduction strategy such that, given a term M , it proceeds in a sequence of leftmost β -contractions $M \rightarrow_{\beta} M_1 \dots \rightarrow_{\beta} M_{j-1} \rightarrow_{\beta} M_j \dots$, i.e. in reducing at every step i the leftmost β -redex of M_{i-1} . This process ends iff M has a β -nf, and if it terminates it ends at $M_{*\beta}$. This result is usually called *leftmost reduction theorem*.

Therefore to seek for $M_{*\beta\eta}$, one has to reduce M by its leftmost β -reduction:

- If the process does not terminate, then by Fact 1.2.1, $M_{*\beta\eta}$ does not exist;
- If the algorithm does terminate, then it does so at $M_{*\beta}$ by leftmost reduction theorem, hence proceeding with a leftmost η -reduction strategy one reaches an η -nf in $|M_{*\beta}|/2$ steps by Remark 2 and η -postponement theorem ([2, 15.1.6]).

2 Simple Type Theory

In this section we start the description of one the simplest typed system based on λ -calculus. It is usually denoted by TA for ‘type assignment’. Here the system is simply denoted by λ_{\rightarrow}^A and it is defined explicitly as a system of λ -abstraction. In spite of this, it has a parallel version in combinatory logic with the same main properties ([18, Ch.14]), that we won’t discuss here.

⁸We have not given a formal definition of ‘length of a reduction’, but the reader can consider it as the number of $\beta(\eta)$ -contractions (finite or not) constituting the given reduction.

2.1 Basics of λ_{\rightarrow}^A

Definition 2.1. We assume an infinite sequence of type-variables, distinct from term-variables, is given.

Types are built by the operator \rightarrow from type-variables: More formally

- each type-variable a, b, c, \dots is a type;
- if τ and σ are types, so is $\tau \rightarrow \sigma$;
- nothing else is a type.

Length of types is defined inductively:

- $|a| := 1$;
- $|\tau \rightarrow \sigma| := |\tau| + |\sigma|$.

The set of distinct type-variables in a type τ is denoted by $Vars(\tau)$.

Definition 2.2. Any expression of the form $M : \tau$ is said type-assignment of the subject M to the type τ .

A finite set of type-assignments $\Gamma = \{x_1 : \tau_1, \dots, x_m : \tau_m\}$ whose subjects are term-variables and which is consistent, i.e. no variable is subject of more than one assignment, is said a type-context.

Notational Convention 3.

- $\Gamma - x$ indicates the result of removing from context Γ the assignment whose subject is x .
- $\Gamma \upharpoonright_M$ is the result of removing from context Γ all the assignments $x_i : \tau_i$ for $x_i \notin FV(M)$.
- When $Subjects(\Gamma) = FV(M)$ for some M , we refer to Γ as an M -context.
- We will say that Γ_1 is consistent with Γ_2 when $\Gamma_1 \cup \Gamma_2$ is consistent.

A sequent-style Natural Deduction system associated to λ_{\rightarrow}^A is given in the table below. We will refer to any expression of the form $\Gamma \vdash M : \tau$, with Γ possibly empty, as a λ_{\rightarrow}^A -formula.

Axioms: $x : \tau \vdash x : \tau$

Rules for \rightarrow :
$$\frac{\Gamma_1 \vdash M : \tau \rightarrow \sigma \quad \Gamma_2 \vdash N : \tau}{\Gamma_1 \cup \Gamma_2 \vdash (MN) : \sigma} \rightarrow\text{-elim} \quad \text{with } \Gamma_1 \cup \Gamma_2 \text{ consistent.}$$

$$\frac{\Gamma \vdash M : \tau}{\Gamma - x \vdash (\lambda x.M) : \sigma \rightarrow \tau} \rightarrow\text{-intro} \quad \text{when } \Gamma \text{ is consistent with } x : \sigma.$$

When one applies \rightarrow -intro, x is said to have been (possibly vacuously) discharged by the rule.

Definition 2.3. Given a type context Γ if there is a λ_{\rightarrow}^A -deduction of $\Gamma \vdash M : \tau$ we say $\Gamma' \vdash_{\lambda_{\rightarrow}^A} M : \tau$ where $\Gamma \subseteq \Gamma'$; when $\Gamma' = \emptyset$, τ is said the type of M , or equivalently we say M has type τ .

Remark 4. It is obvious that if $\Gamma \vdash_{\lambda_{\rightarrow}^A} M : \tau$, then $\Gamma^+ \vdash_{\lambda_{\rightarrow}^A} M : \tau$ for every $\Gamma^+ \supseteq \Gamma$. Moreover by an easy induction on $|M|$ one can prove that if $\Gamma \vdash M : \tau$ is λ_{\rightarrow}^A -deducible then $\text{Subjects}(\Gamma) = \text{FV}(M)$, hence if $\vdash_{\lambda_{\rightarrow}^A} M : \tau$, i.e. $\emptyset \vdash_{\lambda_{\rightarrow}^A} M : \tau$, then M is closed.

We have the following

Lemma 2.1.1 (Subject-construction lemma for λ_{\rightarrow}^A). *For any Γ, M, τ , a λ_{\rightarrow}^A -deduction Δ of $\Gamma \vdash M : \tau$ gives the term structure of M by erasing types and context in Δ . Consequently,*

(i) if $M \equiv x$ then $\Delta \equiv x : \tau \vdash x : \tau$.

(ii) if $M \equiv NP$ then $\Delta \equiv \frac{\begin{array}{c} \vdots \\ \Gamma_1 \vdash N : \sigma \rightarrow \tau \end{array} \quad \begin{array}{c} \vdots \\ \Gamma_2 \vdash P : \sigma \end{array}}{\Gamma_1 \cup \Gamma_2 \vdash NP : \tau} \rightarrow\text{-elim}$.

(iii) if $M \equiv \lambda x.N$ then $\Delta \equiv \frac{\begin{array}{c} \vdots \\ \Gamma, x : \rho \vdash N : \sigma \end{array}}{\Gamma - x \vdash \lambda x.N : \rho \rightarrow \sigma} \rightarrow\text{-intro}$.

Proof. The main statement follows directly from definitions of terms-as-trees and of λ_{\rightarrow}^A -deduction.

To (i)-(iii): we can reason by an easy induction on $|M|$, considering for (iii) both cases of discharge and vacuous discharge. \square

Remark 5. For a general term this result cannot be strengthened into a “terms-deductions correspondence”, since deductions are defined by types that occur in them. Thus deductions with the same tree-structure but occurrences of different types are different. In spite of this, reasoning by induction on $|M|$ and using the head normal form lemma, one can prove that, given a λ_{\rightarrow}^A -deduction Δ of $\Gamma \vdash M : \tau$ with M in β -nf,

(i) Δ is unique, and

(ii) every type in Δ has an occurrence in Γ or in τ .

In virtue of the well-known correspondence we will discuss in Sect.3, (ii) is the type-theoretic version of subformula property for NK-derivations, stating that any formula in an irreducible derivation occurs in the conclusion or in the living assumption of that deduction.

Thinking of type theories as tools to avoid errors of mismatching in programming, it is important that any $\beta\eta$ -calculation preserves safety of a given term of a certain type. This can be stated formally as follows:

Fact 2.1.2 (Subject-reduction Theorem). *If $\Gamma \vdash_{\lambda_{\rightarrow}^A} M : \tau$ and $M \rightarrow_{\beta} N$, then $\Gamma \vdash_{\lambda_{\rightarrow}^A} N : \tau$.*

Proof. See [17, 2C]. □

We do not have a similar result for β -expansion; hence, defining $Types(M)$ as the set of all τ such that $\vdash_{\lambda_{\rightarrow}^A} M : \tau$, we have that β -contraction enlarge this set. This means that the approach we used for λ_{\rightarrow}^A does not guarantee conversion-invariance of $Types(M)$. However this is only a small price we pay for having that $Types(M) \subseteq Types(N)$ whenever $M \rightarrow_{\beta} N$, which can be seen as assuring that β -reductions make M “safer” at any β -step. Moreover, this defect is overcome by adding a new rule to λ_{\rightarrow}^A to work with $=_{\beta(\eta)}$: See [17, Ch.4] for the details.

Definition 2.4. A term M is said $(\lambda_{\rightarrow}^A)$ -typable iff there exist Γ and τ such that $\Gamma \vdash_{\lambda_{\rightarrow}^A} M : \tau$.

Fact 2.1.3. *The class of λ_{\rightarrow}^A -typable terms is closed under taking subterms, abstraction and $\beta\eta$ -reduction.*

Proof. It follows from Lemma 2.1.1, Fact 2.1.2 and definition of \rightarrow -intro, respectively. □

2.2 Principal Type Algorithm for λ_{\rightarrow}^A

A λ_{\rightarrow}^A -typable term has in general an infinite set of types: The term $\lambda x.x$ e.g. is typable with any type of the form $\tau \rightarrow \tau$ but all the types of this set are instances of the type $a \rightarrow a$.

This type is called the *principal type* of $\lambda x.x$, and its corresponding λ_{\rightarrow}^A -deduction is said to be its *principal deduction*.

The Principal Type Algorithm or PT algorithm described in [17, Ch.3] gives a decision procedure for typability of any term of λ_{\rightarrow}^A . From a practical point of view, such an algorithm consists of a type-checking procedure which allows the programmer to decide whether the created program is safe, assuming, as we did before, typability as safety criterion.

We decided to develop this type-checking algorithm by means of Robinson’s notion of unification, first developed in [31].⁹ First some preliminary definitions:

Definition 2.5.

- A type-substitution $\mathbb{S}(\tau)$ of a type τ is a function from type-variables $a_1^{\tau}, \dots, a_n^{\tau}$ in τ to types

$$a_i^{\tau} \mapsto \sigma_i \quad \text{such that}$$

- $\mathbb{S}(a_i^{\tau}) \equiv \sigma_i$ for every $i \in \{1, \dots, n\}$;
- $\mathbb{S}(b) \equiv b$ if $b \neq a_i^{\tau}$ for any $i \in \{1, \dots, n\}$;

⁹For a different treatment based on an equation-solving algorithm see the historical remarks in [17, Ch.3].

$$\circ \mathbb{S}(\rho \rightarrow \sigma) \equiv \mathbb{S}(\rho) \rightarrow \mathbb{S}(\sigma).$$

- Given ρ, σ , a unifier of $\langle \rho, \sigma \rangle$ is any \mathbb{S} such that $\mathbb{S}(\rho) \equiv \mathbb{S}(\sigma)$. Similarly a unifier of $\langle \langle \rho_1, \dots, \rho_n \rangle, \langle \sigma_1, \dots, \sigma_n \rangle \rangle$ is any \mathbb{S} such that $\mathbb{S}(\langle \rho_1, \dots, \rho_n \rangle) \equiv \mathbb{S}(\langle \sigma_1, \dots, \sigma_n \rangle)$.
- A most general unifier (**mgu**) of $\langle \rho, \sigma \rangle$ is a unifier \mathbb{U} such that for any other unifier \mathbb{S} of $\langle \rho, \sigma \rangle$ we have $\mathbb{S}(\rho) \equiv \mathbb{S}'(\mathbb{U}(\rho))$ for some substitution \mathbb{S}' .

We now state the unification algorithm we will use to define the PT procedure. A proof of its correctness is given in [31], but the main idea is to progressively build a **mgu** for the input by means of appropriate concatenation of substitutions.

Unification Algorithm. INPUT: Any pair $\langle \rho, \tau \rangle$ of types.

OUTPUT: Either a correct statement that $\langle \rho, \tau \rangle$ is not unifiable, or a **mgu** \mathbb{U} of $\langle \rho, \tau \rangle$.

Step 0: Choose $k = 0$ and $\mathbb{U}_0 =$ empty substitution.

Step $k + 1$: Given k and \mathbb{U}_k , construct $\rho_k := \mathbb{U}_k(\rho)$ and $\tau_k := \mathbb{U}_k(\tau)$, then apply the following procedure to $\langle \rho_k, \tau_k \rangle$:

Given a pair $\langle \mu, \nu \rangle$ of types, write μ and ν as strings of symbols, say $\mu \equiv s_1 \dots s_m$ and $\nu \equiv t_1 \dots t_n$ ($m, n \geq 1$) where each of $s_1, \dots, s_m, t_1, \dots, t_n$ is an occurrence of a parenthesis, arrow or variable.

Now

- if $\mu \equiv \nu$, then state that $\mu \equiv \nu$ and stop;
- else, choose the least $p \leq \min\{m, n\}$ such that $s_p \not\equiv t_p$: It is easy to verify that one of s_p, t_p must be a variable and the other must be a left parenthesis or a different variable. Further s_p can be shown to be the leftmost symbol of a unique subtype μ^* of μ . Similarly t_p is the leftmost symbol of a unique subtype ν^* of ν . Choose one of μ^*, ν^* that is a variable and call it \aleph , and if both are variables choose the first in a given sequence of all type-variables of \mathcal{L}_{λ_A} . Then call the remaining member of $\langle \mu^*, \nu^* \rangle$ α . The pair $\langle \aleph, \alpha \rangle$ is said disagreement pair for $\langle \mu, \nu \rangle$.

This procedure will output either a correct statement that $\rho_k \equiv \tau_k$, or a disagreement pair $\langle \aleph, \alpha \rangle$ such that $\aleph \not\equiv \alpha$.

- If $\rho_k \equiv \tau_k$, choose $\mathbb{U} = \mathbb{U}_k$;
 - else decide whether $\aleph \in \text{Vars}(\alpha)$:
 - ◇ If $\aleph \in \text{Vars}(\alpha)$, state that $\langle \rho, \tau \rangle$ is not unifiable and stop;
 - ◇ else replace k by $k + 1$, choose $\mathbb{U}_{k+1} = [\alpha/\aleph] \circ \mathbb{U}_k$ and go to Step $k + 2$.
-

Theorem 2.2.1 (PT Theorem for λ_{\rightarrow}^A). *Every typable term has a principal type in λ_{\rightarrow}^A . Moreover, there is an algorithm that will decide whether a given term M is typable in λ_{\rightarrow}^A , and if it is so will output a principal deduction and a principal type for M .*

Proof. Consider the following algorithm:

INPUT: Any term M , closed or not.

OUTPUT: Either a principal deduction Δ_M for M or a correct statement that M is not typable.

Case 1: If $M \equiv x$, choose $\Delta_M = x : a \vdash x : a$ where a is any type-variable.

Case 2: If $M \equiv \lambda x.M_1$ and $x \in FV(M_1) := \{x, x_1, \dots, x_m\}$, then apply the algorithm to M_1 . If M_1 is not typable, neither is M . If M_1 has principal deduction Δ_{M_1} whose conclusion is $x : \alpha, x_1 : \alpha_1, \dots, x_m : \alpha_m \vdash M_1 : \beta$, extend Δ_{M_1} by a \rightarrow -intro discharging x and identify Δ_M with the resulting deduction.

Case 3: If $M \equiv \lambda x.M_1$ and $x \notin FV(M_1)$, proceed as in the previous case, with a vacuous discharge of x when extending Δ_{M_1} by \rightarrow -intro.

Case 4: If $M \equiv M_1 M_2$, apply the algorithm to M_1 and M_2 . If M_1 or M_2 is untypable, so is M . If M_1 and M_2 are both typable, let Δ_{M_1} and Δ_{M_2} be their respective principal deductions. Modulo rename of type-variables we may assume Δ_{M_1} and Δ_{M_2} not to have any common type-variable. Next list free term-variables in M_1 and M_2 , say

$$FV(M_1) = \{u_1, \dots, u_p, w_1, \dots, w_r\} \quad (p, r \geq 0)$$

$$FV(M_2) = \{v_1, \dots, v_q, w_1, \dots, w_r\} \quad (q \geq 0)$$

where $u_1, \dots, u_p, v_1, \dots, v_q, w_1, \dots, w_r$ are distinct.

If $PT(M_1) \equiv \rho \rightarrow \sigma$, then the conclusions of Δ_{M_1} and Δ_{M_2} have form, respectively

$$u_1 : \theta_1, \dots, u_p : \theta_p, w_1 : \psi_1, \dots, w_r : \psi_r \vdash M_1 : \rho \rightarrow \sigma \quad (1)$$

$$v_1 : \phi_1, \dots, v_q : \phi_q, w_1 : \chi_1, \dots, w_r : \chi_r \vdash M_2 : \tau. \quad (2)$$

Apply unification algorithm to $\langle \psi_1, \dots, \psi_r, \rho \rightarrow \sigma \rangle$ and $\langle \chi_1, \dots, \chi_r, \tau \rightarrow c \rangle$ where c is a fresh variable.

- If this pair has no unifier, then $M_1 M_2$ is not typable;
- else the unification algorithm gives a mgu \mathbb{U} and modulo renaming we have

$$\text{Dom}(\mathbb{U}) = \text{Vars}(\psi_1, \dots, \psi_r, \rho, \chi_1, \dots, \chi_r, \tau \rightarrow c) \quad (3)$$

$$\text{Range}(\mathbb{U}) \cap \mathbf{V} = \emptyset, \quad (4)$$

where $\mathbf{v} := (\text{Vars}(\Delta_{M_1}) \cup \text{Vars}(\Delta_{M_2})) \setminus \text{Dom}(\mathbb{U})$. Applying \mathbb{U} to Δ_{M_1} and Δ_{M_2} changes their conclusions to

$$u_1 : \mathbb{U}(\theta_1), \dots, u_p : \mathbb{U}(\theta_p), w_1 : \mathbb{U}(\psi_1), \dots, w_r : \mathbb{U}(\psi_r) \vdash M_1 : \mathbb{U}(\rho \rightarrow \sigma)$$

and

$$v_1 : \mathbb{U}(\phi_1), \dots, v_q : \mathbb{U}(\phi_q), w_1 : \mathbb{U}(\chi_1), \dots, w_r : \mathbb{U}(\chi_r) \vdash M_2 : \mathbb{U}(\tau).$$

Thus, by definition of \mathbb{U} , $\mathbb{U}(\rho \rightarrow \sigma) = \mathbb{U}(\tau \rightarrow c) = \mathbb{U}(\tau) \rightarrow \mathbb{U}(c)$, and we can apply \rightarrow -intro to $\mathbb{U}(\Delta_{M_1})$ and $\mathbb{U}(\Delta_{M_2})$. Choose Δ_M to be the resulting deduction.

The case of $PT(M_1)$ atomic is parallel, and it is left to the reader.

For a proof of correctness of this algorithm see [17, 3E1].

□

Hence we have

Corollary 2.2.2. *Type-checking and typability for λ_{\rightarrow}^A are decidable.*

For the sake of completeness we remind here that the “dual” of PT theorem also holds for λ_{\rightarrow}^A .

Fact 2.2.3 (Second PT Theorem). *If a type τ is assignable to a closed term M , then it is the principal type of a closed term M^* (possibly $M \equiv M^*$).*

A detailed proof is given in [17, Ch.7], where an algorithm is also defined so that, given τ and M , one can construct M^* combining an occurrence of M with restricted sets of “building blocks” consisting in extra terms whose principal type is known.

2.3 Church’s typing

So far we have worked with Curry’s approach to typing, which can be seen as an ancestor of polymorphic type theory with unexpressed polymorphism.

In contrast, Church’s way of introducing types in λ -calculus restricts the definition of term by giving to each M a type as part of its structure. Formally we have

Definition 2.6. Given a type-context Γ , the set $\text{TT}(\Gamma)$ of typed terms relative to Γ is a set of expressions such that

- if $x : \sigma \in \Gamma$, $\text{TT}(\Gamma)$ contains the term-variable x^σ ;
- if $\Gamma_1 \cup \Gamma_2$ is consistent and $M^{\sigma \rightarrow \tau} \in \text{TT}(\Gamma_1)$ and $N^\sigma \in \text{TT}(\Gamma_2)$, then $(M^{\sigma \rightarrow \tau} N^\sigma)^\tau \in \text{TT}(\Gamma_1 \cup \Gamma_2)$;
- if Γ is consistent with $\{x : \sigma\}$ and $M^\tau \in \text{TT}(\Gamma)$, then $(\lambda x^\sigma. M^\tau)^{\sigma \rightarrow \tau} \in \text{TT}(\Gamma - x)$.

If M^τ is a typed-term (relative to some Γ), τ is said the type of M^τ .

Notational Convention 4. Types of terms are omitted whenever they are obvious, so that, for instance, we shall write $(\lambda x.M)^\sigma \rightarrow \tau$ for $(\lambda x^\sigma.M^\tau)^\sigma \rightarrow \tau$.

Remark 6. We are working with typed terms relative to a given context, while Church's system in [5] has terms typed in an absolute sense, so that if $M^{\sigma \rightarrow \tau}$ and N^σ are (typed) terms so is $(MN)^\tau$, while this happens here only when the union of their respective contexts is consistent. This is, so to speak, a median between Curry-style typing, known as type-assignment, and Church-style one, whose corresponding system is usually said typed. We adopt the notation λ_{\rightarrow}^A for systems with former approach and λ_{\rightarrow}^C for those with the latter.

Notational Convention 5. We indicate with M^\dagger the type-erasure of M^τ , i.e. the untyped term obtained by erasing all types from M^τ .¹⁰

In spite of formal differences, one can easily identify these approaches in the following sense:

Lemma 2.3.1. *The translation here given from $\mathcal{L}_{\lambda_{\rightarrow}^A}$ to $\mathcal{L}_{\lambda_{\rightarrow}^C}$ is a one-to-one correspondence between λ_{\rightarrow}^A -deductions Δ and typed terms M_{Δ}^τ :*

◦ If $\Delta = x : \tau \vdash x : \tau$, define $M_{\Delta}^\tau \equiv x^\tau$;

◦ If $\Delta = \frac{\Gamma_1 \vdash N : \sigma \rightarrow \tau \quad \Gamma_2 \vdash P : \sigma}{\Gamma_1 \cup \Gamma_2 \vdash NP : \tau} \rightarrow\text{-elim}$,

define $M_{\Delta}^\tau \equiv (N_{\Delta_1}^{\sigma \rightarrow \tau} P_{\Delta_2}^\sigma)^\tau$, where $N_{\Delta_1}^{\sigma \rightarrow \tau} \in \mathbf{TT}(\Gamma_1)$ and $P_{\Delta_2}^\sigma \in \mathbf{TT}(\Gamma_2)$;

◦ If $\Delta = \frac{\Gamma, x : \rho \vdash N : \sigma}{\Gamma - x \vdash \lambda x.N : \rho \rightarrow \sigma} \rightarrow\text{-intro}$,

define $M_{\Delta}^\tau \equiv (\lambda x^\rho.N_{\Delta'}^\sigma)^\tau$, where $N_{\Delta'}^\sigma \in \mathbf{TT}(\Gamma)$ and $\tau \equiv \rho \rightarrow \sigma$.

In particular, if Δ is a λ_{\rightarrow}^A -deduction of $\Gamma \vdash M : \tau$, then

$$M_{\Delta}^\tau \in \mathbf{TT}(\Gamma) \quad \text{and} \quad M_{\Delta}^\dagger \equiv M.$$

Proof. By construction we have $M_{\Delta}^\tau \in \mathbf{TT}(\Gamma)$ and $M^\dagger \equiv M$ whenever Δ is a λ_{\rightarrow}^A -deduction of $\Gamma \vdash M : \tau$. To show that this is indeed a one-to-one correspondence we define the following inverse translation $\bar{\Delta}(-)$ from typed terms to λ_{\rightarrow}^A -deduction:

- if $M^\tau \equiv x^\tau$, define $\bar{\Delta}(M^\tau) = x : \tau \vdash x : \tau$;
- if $M^\tau \equiv (M_1^{\sigma \rightarrow \tau} M_2^\sigma)$, $M_1^{\sigma \rightarrow \tau} \in \mathbf{TT}(\Gamma_1)$, $M_2^\sigma \in \mathbf{TT}(\Gamma_2)$, $\Gamma_1 \cup \Gamma_2$ is consistent, and $\bar{\Delta}(M_1^{\sigma \rightarrow \tau}) = \frac{\vdots}{\Gamma'_1 \vdash M_1 : \sigma \rightarrow \tau}$, $\bar{\Delta}(M_2^\sigma) = \frac{\vdots}{\Gamma'_2 \vdash M_2 : \sigma}$ for $\Gamma'_1 \subseteq \Gamma_1$ and $\Gamma'_2 \subseteq \Gamma_2$, then define $\bar{\Delta}(M^\tau)$ by application of $\rightarrow\text{-elim}$ to $\bar{\Delta}(M_1^{\sigma \rightarrow \tau})$, $\bar{\Delta}(M_2^\sigma)$;

¹⁰Obviously, one could define type-erasure as a forgetful function from typed-terms to λ_{\rightarrow}^A -terms by induction on the structure of M , but for the present purpose our convention is sufficient.

- if $M^\tau \equiv (\lambda x^\rho. M_1^\sigma)^{\rho \rightarrow \sigma}$, $\tau \equiv \rho \rightarrow \sigma$, $M_1^\sigma \in \mathbf{TT}(\Gamma)$, Γ is consistent with $\{x : \rho\}$, and $\bar{\Delta}(M_1^\sigma) = \frac{\dot{\quad}}{\Gamma' \vdash M_1 : \sigma}$ with $\Gamma' \subseteq \Gamma$, define $\bar{\Delta}(M^\tau)$ by application of \rightarrow -intro to $\bar{\Delta}(M_1^\sigma)$, possibly discharging $x : \rho$.

It is straightforward that $\bar{\Delta}(M_\Delta^\tau) = \Delta$, and the result is proved. \square

Note that from this result we have that if $M^\tau \in \mathbf{TT}(\Gamma)$, then $\bar{\Delta}(M^\tau)$ is a λ_{\rightarrow}^A -deduction of $\Gamma_{\upharpoonright M^\tau} \vdash M^\tau : \tau$, so that we can think of M^τ as encoding a λ_{\rightarrow}^A -deduction whose conclusion has M^τ as subject.

Definitions of length, replacement, binding, (typed) substitution, α -conversion, $\beta(\eta)$ -contraction (and reduction) are just the same as those for λ_{\rightarrow}^A , with the obvious caveats concerning type-mismatches to assure we are working with well-defined typed terms. Thus we can state the following

Fact 2.3.2.

- (i) If a β - or η -contraction in a typed term M^τ changes M^τ to an expression E , then E is also a typed term with type τ .
- (ii) If $M^\tau \rightarrow_{\beta\eta} N^\tau$, then $\Gamma_{\upharpoonright N^\tau} \subseteq \Gamma_{\upharpoonright M^\tau}$ and $M^\tau \rightarrow_{\beta\eta} N^\tau$.¹¹
- (iii) A typed term M^τ has a β -reduction with length n iff M^τ does; and the same holds for η -reduction.
- (iv) A typed term M^τ is a β -nf iff M^τ is a β -nf; the same holds for η -nf.

Proving these statements involves only verifications of formal definitions of the previous notions, thus we leave it to [17, 5B].

Moreover, reasoning as with untyped λ -terms we can prove the Church-Rosser theorem for $\beta\eta$ -reduction of typed terms. Again, by this we obtain the uniqueness of $\beta\eta$ -nf's for typed terms, but we still do not know if each term has a nf. That is the question we are now going to answer.

2.4 Weak Normalization Theorem

We assumed that reductions are, so to speak, imitations of the process of computing values. What we will now prove is that a computation can always be continued to a final result.

Theorem 2.4.1 (Weak Normalization Theorem, Turing 1942). *Every typed term M^π has both a β -nf and a $\beta\eta$ -nf.*

Proof. If M^π has a β -nf, then it has also a $\beta\eta$ -nf by the typed version of Fact 1.2.1. Thus it suffices to prove the existence of a β -nf for M^π . This is obtained by enumerating first all the β -reductions in one step, all the β -reductions in two steps, and so on, until the β -nf is found: This will eventually happen since typed systems do not allow self-application, so there are only finitely many reductions of length n starting at a fixed term.

Formally, define:

¹¹Proof-theoretically (i)-(ii) mean that a reduction of a deduction of $\Gamma \vdash M : \tau$ in the sense of [29] gives a genuine deduction of $\Gamma' \vdash N : \tau$ for some N , Γ' such that $M \rightarrow_\beta N$ and $\Gamma' \subseteq \Gamma$.

- the degree $\partial(\tau)$ of a type τ , such that

$$\partial(a) := 1 \text{ for } a \text{ type variable}$$

$$\partial(\rho \rightarrow \sigma) := \max(\partial(\rho), \partial(\sigma)) + 1;$$
- the degree $\partial(\mathcal{R})$ of a β -redex \mathcal{R}^σ , such that $\partial((\lambda x.N)P) := \partial(\rho \rightarrow \sigma)$ where $(\lambda x.N)^{\rho \rightarrow \sigma}$ and P^ρ ;
- the degree $d(N^\tau)$ of a term as the *sup* of the degrees of the redexes it contains, and if N^τ is a β -nf, then $d(N^\tau) := 0$.

Now it is pure routine to check that $d(N[P^\sigma/x^\sigma]^\tau) \leq \max(d(N^\tau), d(P^\sigma), \partial(\sigma))$, and obviously, for any \mathcal{R}^σ , $\partial(\mathcal{R}^\sigma) > \partial(\sigma)$. It is also straightforward that if $N^\sigma \rightarrow_\beta P^\sigma$ then $d(P^\sigma) \leq d(N^\sigma)$. Moreover we have that

- (\star) If \mathcal{R}^σ is a redex of maximal degree n in M^π and all the redexes strictly contained in \mathcal{R}^σ have degree less than n , then, if N^π is obtained from M^π by converting \mathcal{R}^σ to \mathcal{C}^σ , N^π has strictly fewer redexes of degree n .

Proof of (\star). After the contraction, the redexes outside \mathcal{R}^σ remain, while those strictly contained in \mathcal{R}^σ are in general conserved, but sometimes proliferated. However, these redexes have degrees less than n by hypothesis. Moreover \mathcal{R}^σ is possibly replaced by some redexes of strictly smaller degree. From these facts, the result follows. \square

Finally, define $\mu(M^\pi) := (n, m)$ with $n := d(M^\pi)$, $m := \#\mathcal{R}^\sigma$ such that $\partial(\mathcal{R}^\sigma) = n$. By (\star), it is possible to choose a redex \mathcal{R}^σ of M^π such that, after contraction to \mathcal{C}^σ , the resulting term M'^π satisfies $\mu(M'^\pi) < \mu(M^\pi)$ w.r.t. lexicographic order. Thus the main statement is proved by a double induction. \square

As a consequence we have

Corollary 2.4.2 (Weak Normalization Theorem for λ_{\rightarrow}^A). *Every typable λ_{\rightarrow}^A -term has both a β -nf and a $\beta\eta$ -nf.*

Proof. This follows from Lemma 2.3.1, Fact 2.3.2 and the Weak Normalization Theorem (WN) for typed terms. \square

Corollary 2.4.3. *There is a decision-procedure for β -equality of typable λ_{\rightarrow}^A -terms; i.e. an algorithm which, given any typable terms P and Q , will decide whether $P =_\beta Q$. Similarly for $\beta\eta$ -equality.*

Proof. Reduce P and Q to their respective β -nf's (which exists by WN, and can be found using the leftmost reduction strategy) and see whether they differ. \square

2.5 Strong Normalization Theorem

WN says that every λ_{\rightarrow}^A -typable term can be reduced to a normal form, so that we can use a defined strategy to be sure that our computation terminates. But we will now prove that *all* computations do terminate, independently to any strategy we use. That is the content of the Strong Normalization Theorem (SN), by which we obtain the following

Lemma 2.5.1. *There is an algorithm which accepts M^π as input and outputs a number $\nu(M^\pi)$ such that all reductions of M^π have length $\leq \nu(M^\pi)$.*

Proof. Organise all the possible $\beta\eta$ -reduction of M^π as a branching tree, specifying a redex of $M_0^\pi := M^\pi$, then a redex of M_1^π , and so on. Since any term has only a finite number of subterms, this tree is finitely branching, and since, by SN, every reduction is finite, the tree has no infinite branch. Thus, by Kőning's Lemma, the whole tree must be finite, so that we can measure its branches and define $\nu(M^\pi)$ as the maximum of their lengths.¹²

□

We are going to prove SN for typed terms using what is usually known as a *computability predicate* defined by a suitable induction. This method is due essentially to [36] and it has revealed extremely flexible. For this reason, it will be the *exemplar* for proofs of the analogous results concerning the type theories we will investigate in the following sections.

Theorem 2.5.2 (Strong Normalization Theorem). *All typed terms are strongly β -normalizable; i.e. given a typed term M^π , all β -reductions of M^π are finite.*

Proof. First define a set COMP_ρ by induction on $|\rho|$:

- for ρ atomic, $N^\rho \in \text{COMP}_\rho$ iff N^ρ is strongly normalizable;
- for $\rho \equiv \sigma \rightarrow \tau$, $N^\rho \in \text{COMP}_\rho$ iff, for all $P^\sigma \in \text{COMP}_\sigma$, $NP \in \text{COMP}_\tau$.

Next, let a term be said *neutral* iff it is not of the form $\lambda x.Q$.

We will now prove by induction on $|\rho|$ that COMP_ρ satisfies the following conditions:¹³

- (CR1) if $N \in \text{COMP}_\rho$, then N is strongly normalizable;
- (CR2) if $N \in \text{COMP}_\rho$ and $N \rightarrow_\beta N'$, then $N' \in \text{COMP}_\rho$;
- (CR3) if N is neutral and whenever one converts a redex of N a term $N' \in \text{COMP}_\rho$ is obtained, then $N \in \text{COMP}_\rho$.¹⁴

□ If ρ is an atom, then

(CR1) is trivial;

¹²This is clearly an inefficient algorithm, since the proof that $\nu(M^\pi)$ is well-define depends on Kőning's Lemma, which is non-constructive. But there exist more efficient ways of computing these bounds, as is proved in [32].

¹³Types will be here omitted to facilitate readability, since they can be easily derived from the context.

¹⁴Note that the following is a special case of (CR3):

(CR4) if N^ρ is neutral and normal, then $N \in \text{COMP}_\rho$.

(CR2) holds for if N is strongly normalizable so is any N' such that $N \rightarrow_{\beta} N'$;

(CR3) follows from the fact that the number $\nu(N)$ is equal to the greatest of the $\nu(N') + 1$ (which are well-defined, since it is obvious that a term is strongly normalizable iff there exists such a length-bound of its normalizations) as N' varies over the one-step reductions of N , so N is strongly normalizable.

□ If $\rho \equiv \sigma \rightarrow \tau$, then $N \in \text{COMP}_{\rho}$ iff all its applications to computable terms are computable. Hence

to (CR1): Let $N \in \text{COMP}_{\rho}$ and x^{σ} be a typed-variable. The induction hypothesis (CR3) for σ gives $x \in \text{COMP}_{\sigma}$, hence $Nx \in \text{COMP}_{\tau}$, and $\nu(N) \leq \nu(Nx)$, since from any reduction sequence $N N' \dots$ starting at N one can construct a reduction sequence $Nx N'x \dots$. The induction hypothesis (CR1) for τ assures $\nu(Nx)$ is finite, therefore N is strongly normalizable;

to (CR2): If $N \rightarrow_{\beta} N'$ and $N \in \text{COMP}_{\rho}$, let $P \in \text{COMP}_{\sigma}$. Then $NP \in \text{COMP}_{\tau}$ and $NP \rightarrow_{\beta} N'P$. By induction hypothesis (CR2) for τ we have $N'P \in \text{COMP}_{\tau}$, so $N' \in \text{COMP}_{\rho}$;

to (CR3): Assume that N is neutral and all the N' 's one step from N are computable. Let $P \in \text{COMP}_{\sigma}$. By induction hypothesis (CR1) for σ , P is strongly normalizable, so we can reason by induction on $\nu(P)$: In one β -step NP reduces to

◇ $N'P$, with $N' \in \text{COMP}_{\rho}$, so $N'P \in \text{COMP}_{\tau}$; or

◇ NP' , with P' one β -step from P . $P' \in \text{COMP}_{\sigma}$ by induction hypothesis (CR2) for σ , and $\nu(P') < \nu(P)$, so by induction hypothesis we have $NP' \in \text{COMP}_{\tau}$.

All possible cases are exhaust, since N is neutral, and so it has not the form $\lambda x.P$.

Thus the induction hypothesis (CR3) for τ gives $NP \in \text{COMP}_{\tau}$, and so $N \in \text{COMP}_{\rho}$.

Finally we state the following claims:

(*) If, for all $P \in \text{COMP}_{\sigma}$, $N[P/x] \in \text{COMP}_{\tau}$, then $\lambda x.N \in \text{COMP}_{\sigma \rightarrow \tau}$.

(**) Given any term N^{ρ} such that $FV(N) \subseteq \{x_1^{\sigma_1}, \dots, x_n^{\sigma_n}\}$,
if $Q_1 \in \text{COMP}_{\sigma_1}, \dots, Q_n \in \text{COMP}_{\sigma_n}$, then $N[Q_1/x_1, \dots, Q_n/x_n] \in \text{COMP}_{\rho}$.

Proof of the claims.

(*) is proved by an easy induction on $\nu(P) + \nu(N)$.

(**) is proved by induction on $|N|$, using (*). □

The main result now follows by (CR1) from (**) for M^{π} and $Q_i^{\sigma_i} := x_i^{\sigma_i}$. □

As usual, we can now shift to λ_{\rightarrow}^A and state the following

Corollary 2.5.3 (SN for λ_{\rightarrow}^A). *If M is a λ_{\rightarrow}^A -typable term, every β -reduction that starts at M is finite.*

Proof. It follows from Fact 2.3.2 and Theorem 2.5.2. □

We conclude noting that this result can be extended to $\beta\eta$ -reductions reasoning in a similar way: See [18, A2] for a detailed proof.

3 Curry-Howard Isomorphism

Curry-Howard isomorphism establishes a close correspondence between typed λ -calculi and logical systems, allowing to think of the former as suitable frameworks for discussing the functional objects which, according to the BHK-interpretation, lie behind proofs developed in a given calculus.

3.1 Implicational fragment of \mathbb{I}

Our exposition of this correspondence starts with a brief review of the implicational fragment of intuitionistic propositional logic \mathbb{I}^{\rightarrow} . Given in a NJ-formulation, it amounts to the following rules

$$\frac{\begin{array}{c} \Gamma_1 \\ \vdots \\ \alpha \rightarrow \beta \end{array} \quad \begin{array}{c} \Gamma_2 \\ \vdots \\ \alpha \end{array}}{\beta} \rightarrow\text{-elim} \qquad \frac{\begin{array}{c} \Gamma, [\alpha] \\ \vdots \\ \beta \end{array}}{\alpha \rightarrow \beta} \rightarrow\text{-intro: } \alpha$$

Here α, β, \dots are used for \mathbb{I}^{\rightarrow} -formulae, built up from propositional variables using implicational only; $\Gamma, \Gamma_1, \Gamma_2, \dots$ denote sets of assumptions. In \rightarrow -intro, a certain number (possibly null) of occurrences of hypothesis α is discharged, so that $\alpha \rightarrow \beta$ in the conclusion depends only on $\Gamma \setminus \{\alpha\}$. We write $\mathbb{I}^{\rightarrow} \vdash \alpha$ for ‘ α is provable in \mathbb{I}^{\rightarrow} ’, and $\Gamma \vdash_{\mathbb{I}^{\rightarrow}} \alpha$ for ‘ α is derivable in \mathbb{I}^{\rightarrow} from assumptions in Γ ’.

Curry-Howard isomorphism can be formally stated in the following way:

Theorem 3.1.1 (Curry-Howard Theorem).

- (i) *The provable formulae in \mathbb{I}^{\rightarrow} are exactly the types of closed λ_{\rightarrow}^A -terms.*
- (ii) *$\alpha_1, \dots, \alpha_n \vdash_{\mathbb{I}^{\rightarrow}} \beta$ iff there exists M and distinct x_1, \dots, x_n such that $x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash_{\lambda_{\rightarrow}^A} M : \beta$.*
- (iii) *There exists a one-to-one correspondence between λ_{\rightarrow}^A -proofs and \mathbb{I}^{\rightarrow} -deductions (modulo \equiv_{α}).*

Proof. (iii): Given a λ_{\rightarrow}^A -deduction Δ of $\Gamma \vdash M : \tau$, define a \mathbb{I}^{\rightarrow} -deduction Δ_L as follows:

- If $M \equiv x$ and $\Delta \equiv x : \tau \vdash x : \tau$, put $\Delta_L := \tau$;

- If $M \equiv M_1M_2$, $\Gamma = \Gamma_1 \cup \Gamma_2$, and

$$\Delta \equiv \frac{\frac{\Delta_1}{\Gamma_1 \vdash M_1 : \sigma \rightarrow \tau} \quad \frac{\Delta_2}{\Gamma_2 \vdash M_2 : \sigma}}{\Gamma_1 \cup \Gamma_2 \vdash M_1M_2 : \sigma}, \quad \text{put}$$

$$\Delta_L := \frac{\frac{\Delta_{1L}}{\sigma \rightarrow \tau} \quad \frac{\Delta_{2L}}{\sigma}}{\tau};$$
- If $M \equiv \lambda x.M_1$, $\tau \equiv \rho \rightarrow \sigma$, $\Gamma = \Gamma' - x$, and

$$\Delta \equiv \frac{\Gamma' \vdash M_1 : \sigma}{\Gamma \vdash \lambda x.M_1 : \rho \rightarrow \sigma},$$
 construct Δ_L by applying \rightarrow -intro to Δ'_L and discharging all occurrences of ρ in Δ'_L whose positions are the same as the positions of the free occurrences of x in M_1 .

Conversely to each \mathbb{I}^\rightarrow -deduction Π define a λ_{\rightarrow}^A -deduction Π_λ as follows:

- if $\Pi \equiv \alpha$, then $\Pi_\lambda := x : \alpha \vdash x : \alpha$;
- if $\Pi \equiv \frac{\frac{\Pi_1}{\alpha \rightarrow \beta} \quad \frac{\Pi_2}{\alpha}}{\beta}$, then Π_λ is obtained by applying \rightarrow -elim to the conclusions of $\Pi_{1\lambda}$ and $\Pi_{2\lambda}$ respectively;
- if $\Pi \equiv \frac{\Gamma}{\frac{\beta}{\alpha \rightarrow \beta} \rightarrow\text{-intro: } \alpha}$, then

$$\Pi'_\lambda := \frac{\frac{\Gamma}{\Gamma, y_1 : \alpha, \dots, y_k : \alpha \vdash M_1 : \beta}}{\vdots}, \quad \text{where } y_1, \dots, y_k \text{ are distinct and occur free in } M_1 \text{ at the same positions as the } k \text{ occurrences of } \alpha \text{ discharged by } \rightarrow\text{-intro in } \Pi. \text{ Now, if that discharge is not vacuous, replace all of } y_1, \dots, y_k \text{ by one new variable } x \text{ that does not occur in } \Pi'_\lambda, \text{ so that } \Pi'_\lambda \text{ changes to a deduction of } \Gamma, x : \alpha \vdash M_1[x/y_1, \dots, x/y_k]. \text{ Then apply } \rightarrow\text{-intro to } \Pi'_\lambda \text{ and put}$$

$$\Pi_\lambda := \frac{\frac{\Gamma, x : \alpha \vdash M_1[x/y_1, \dots, x/y_k]}{\vdots}}{\Gamma \vdash \lambda x.M_1[x/y_1, \dots, x/y_k] : \alpha \rightarrow \beta}.$$

If $k = 0$, then $\Pi'_\lambda := \frac{\Gamma}{\Gamma \vdash M_1 : \beta}$, and define Π_λ by applying \rightarrow -intro with a vacuous discharge of a new variable not in Π'_λ to deduce $\Gamma \vdash \lambda x.M_1 : \alpha \rightarrow \beta$.

Now, by a straightforward induction on $|M|$, we have that, for any λ_{\rightarrow}^A -deduction Δ , $(\Delta_L)_\lambda \equiv \Delta$ modulo \equiv_α , and we can prove, by a straightforward induction on the length of Π , that for any \mathbb{I}^\rightarrow -deduction Π , $(\Pi_\lambda)_L \equiv \Pi$, and we are done.

- (i): It follows immediately from (iii).
- (ii): ‘If’ direction follows from the definition of the function L from λ_{\rightarrow}^A -deductions to \mathbb{I}^\rightarrow -deductions. Conversely, let Π has form

$$\frac{\alpha_1, \dots, \alpha_n}{\beta} .$$

Apply \rightarrow -intro n times to obtain a derivation Π' of $\alpha_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta$, then construct Π'_λ , whose conclusion will be $\vdash M : \alpha_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta$ for some closed M .

Hence, by the subject-construction lemma, $M \equiv \lambda x_1 \dots x_n . M_1$ for some M_1 and distinct x_1, \dots, x_n , and Π'_λ must contain the formula $x_1 : \alpha_1, \dots, x_n : \alpha_n \vdash_{\lambda\rightarrow} M_1 : \beta$. □

Corollary 3.1.2. *To decide whether a formula α is provable in \mathbb{I}^\rightarrow it suffices to decide whether there exists a closed typed-term M^α .*

Proof. From Theorem 3.1.1 we have that the decidability of α is equivalent to the decidability of the existence of a closed M such that $\vdash_{\lambda\rightarrow} M : \alpha$. The result follows from Lemma 2.3.1. □

Remark 7. In [17, Ch. 8] a search algorithm is given to decide, for any type τ , whether the number of closed β -nf's that receive type τ is finite or infinite; it will compute this number in the finite case, and will list all the relevant terms in both cases. This gives, by the Curry-Howard Theorem, a test for provability in \mathbb{I}^\rightarrow . Despite a decision procedure for \mathbb{I}^\rightarrow has been known long time before these results were established,¹⁵ the search algorithm can be thought of as generating all the irreducible \mathbb{I}^\rightarrow -deductions of the formula τ , for the propositions-as-types correspondence we have just proved.

3.2 Substructural Logics

Curry-Howard isomorphism can be “extended” toward different directions, but before we explore these wider horizons, we decided to investigate the relevance of thinking of $\lambda\rightarrow$ -terms as proofs, starting with the following

Definition 3.1.

- A λ -term M is called λI -term iff, for each subterm with form $\lambda x . M_1$ in M , x occurs free in M_1 at least once.
- A λ -term M is called $BCK\lambda$ -term iff
 - (i) for each subterm $\lambda x . M_1$ of M , x occurs free in M_1 at most once;
 - (ii) each free variable of M has just one occurrence free in M .
- A λ -term M is called $BCI\lambda$ -term iff
 - (i) for each subterm $\lambda x . M_1$ of M , x occurs free exactly once in M_1 ;
 - (ii) each free variable of M has just one occurrence free in M .¹⁶

¹⁵See [13].

¹⁶Informally, we can refer to λI -terms as terms without vacuous binding, $BCK\lambda$ -terms as terms in which each subterms has at most one non-bound occurrence of each variable, and $BCI\lambda$ -terms as terms which are both λI - and $BCK\lambda$ -terms.

Recall now that a logic is said *relevant* when vacuous discharge in \rightarrow -intro of the relative calculus is not allowed, and *contraction-free* when multiple discharge is not allowed. Therefore it is pure routine to check

Fact 3.2.1. *The following hold:*

- (i) *The provable formulae in the implicational fragment of relevant intuitionistic propositional logic are exactly the types of the closed λI -terms.*
- (ii) *The provable formulae in the implicational fragment of contraction-free intuitionistic propositional logic are exactly the types of the closed $BCK\lambda$ -terms.*
- (iii) *The provable formulae in the implicational fragment of contraction-free relevant intuitionistic propositional logic are exactly the types of the closed $BCI\lambda$ -terms.*

Thus, in spite of limitations of λ_{\rightarrow}^A , reasoning inside such a basic typed system gives relevant informations about the kind of proofs we are dealing with, establishing in a neat and clear way their structural properties.

Note also that Curry-Howard isomorphism sheds a light on the PT algorithm: In virtue of this propositions-as-types correspondence, the method for constructing the condensed detachment formula in an Hilbert-style system is the same as the core of the very PT algorithm, so that the existence of M^* in the converse PT-algorithm results equivalent to the completeness of the condensed detachment rule for \mathbb{I}^{\rightarrow} , and similarly for the corresponding substructural systems defined by λI -, $BCK\lambda$ -, and $BCI\lambda$ -terms. See [26] for the first exposition of D -computation and [17, 7D] for the correspondence with λ -systems.

3.3 Intuitionistic Proofs as Programs

We focus now to some extensions of simple type theory, starting with the system obtained defining a new type, named ‘product-type’. Here and in the following sections we will work with Church’s typing only, so that all terms will be typed-terms w.r.t. a given context, which will be omitted unless necessary. We shall also avoid the formalism of the previous sections: The reader should by now be able to recognize the usual techniques and reasoning paths.

First, define inductively the class of types \mathbf{T} in the following way:

$$\tau \in \mathbf{T} := a \mid \rho \rightarrow \sigma \mid \rho \times \sigma \quad .$$

Consequently, extend the set of typed-terms to the following set:

$$M^{\tau} \in \Lambda^{\mathbf{T}} := x^{\tau} \mid (M_1^{\rho \rightarrow \sigma} M_2^{\sigma})^{\sigma} \mid (\lambda x^{\rho}. M_1^{\sigma})^{\rho \rightarrow \sigma} \mid \langle M_1^{\rho}, M_2^{\sigma} \rangle^{\rho \times \sigma} \mid (\pi^1.N^{\rho \times \sigma})^{\rho} \mid (\pi^2.N^{\rho \times \sigma})^{\sigma} \quad .$$

Next define the relation \gg of reduction among terms such that

$M \gg M'$ iff there exists a sequence $M \equiv M_0 M_1 \dots M_n \equiv M'$ such that, for $i = 0, 1, \dots, n-1$, M_{i+1} is obtained from M_i by replacing one of the following redexes by its corresponding contractum:

REDEX	CONTRACTUM
$(\lambda x^\sigma . N^\tau) P^\sigma$	$(N[P^\sigma/x^\sigma])^\tau$
$\pi^1 . \langle N, P \rangle^{\sigma \times \tau}$	N^σ
$\pi^2 . \langle N, P \rangle^{\sigma \times \tau}$	P^τ

Using standard methods it is also possible to prove

Fact 3.3.1. \gg has the Church-Rosser Property.

As a consequence, we have that a denotational calculus of \gg -reduction is consistent: Defining \gg -nf's in the natural way, the Church-Rosser property implies, for any term, the uniqueness of \gg -nf, so that denotational consistency is shown by the fact that if $x^\tau \not\equiv y^\tau$ are \gg -nf's, then if we could prove $x^\tau =_{\gg} y^\tau$, by the Church-Rosser property there would exist an M^τ such that $x^\tau \gg M^\tau$ and $y^\tau \gg M^\tau$, *contra* our assumption.

Moreover, both WN and SN Theorems can be extended to this system:

Theorem 3.3.2 (Weak Normalization Theorem). *For every $M^\tau \in \Lambda^T$, there exists a \gg -nf.*

Proof Sketch. The idea is the same as that used to prove WN for β -nf's. One only has to extend definitions of type-degree by

$$\partial(\rho \times \sigma) := \partial(\rho \rightarrow \sigma);$$

redex-degree by

$$\partial(\pi^i . \langle N^\rho, P^\sigma \rangle) := \partial(\rho \times \sigma);$$

and then consider the additional cases related to the product-type, reasoning as is done in proving Theorem 2.4.1. Details are left to the reader. □

Theorem 3.3.3 (Strong Normalization Theorem). *For every $M^\pi \in \Lambda^T$, all reductions of M^τ are finite.*

Proof Sketch. Reasoning as in proof of Theorem 2.5.2, define a new predicate $\widehat{\text{COMP}}_\rho$ by adding to the conditions defining inductively COMP_ρ the following:

- for $\rho \equiv \sigma \times \tau$, $N^\rho \in \widehat{\text{COMP}}_\rho$ iff $\pi^1 . N^\rho \in \widehat{\text{COMP}}_\sigma$ and $\pi^2 . N^\rho \in \widehat{\text{COMP}}_\tau$.

Then say a term to be *neutral* if it is not an abstraction or a pair.

Next prove by induction on ρ that (CR1)-(CR3) hold for $\widehat{\text{COMP}}_\rho$, considering the case of product-type (which is treated similarly to the arrow-type). Now it is pure routine to check that if N, P are computable, so is $\langle N, P \rangle$: Use this fact to easily prove the homologous to (*) and (**). Finally, the result follows from the latter claim in the obvious way. □

Curry-Howard isomorphism permits a functional reading of proofs in \wedge, \rightarrow -fragment of \mathbb{I} , once one recognize the following mapping:

$\Pi \equiv \tau$	\mapsto	x_i^τ ,	where i is the parcel of the hypothesis τ
$\Pi \equiv \frac{\Pi_1 \quad \Pi_2}{\tau \wedge \sigma}$	\mapsto	$\langle M^\tau, N^\sigma \rangle$,	where M^τ, N^σ correspond to Π_1 and Π_2 resp.
$\Pi \equiv \frac{\Pi'}{\tau \wedge \sigma}$	\mapsto	$\pi^1.M^{\tau \times \sigma}$,	where $M^{\tau \times \sigma}$ corresponds to Π'
$\Pi \equiv \frac{\Pi'}{\tau \wedge \sigma}$	\mapsto	$\pi^2.M^{\tau \times \sigma}$,	where $M^{\tau \times \sigma}$ corresponds to Π'
$\Pi \equiv \frac{\Pi'}{\tau \rightarrow \sigma}$	\mapsto	$\lambda x_i^\tau.M^\sigma$,	where M^σ corresponds to Π' and i is the parcel of discharged hypotheses τ
$\Pi \equiv \frac{\Pi_1 \quad \Pi_2}{\tau \rightarrow \sigma}$	\mapsto	$M^{\tau \rightarrow \sigma} N^\tau$,	where $M^{\tau \rightarrow \sigma}, N^\tau$ correspond to Π_1 and Π_2 resp.

On this perspective, normalizations results assure the terminating of any procedure to eliminate detours in natural deduction proofs of this fragment of intuitionistic propositional logic. But this holds for propositional NJ *as a whole*, once the following types and operator are added to the functional calculus.

\vee corresponds to *sum-type*:

- $(\iota^1.M^\tau)^{\tau+\sigma}$;
- $(\iota^2.N^\sigma)^{\tau+\sigma}$;
- $(\delta x^\rho.M^\tau y^\sigma.N^\tau P^{\rho+\sigma})^\tau$, where δ bounds all occurrences of x^ρ in M^τ and all occurrences of y^σ in N^τ .

Consequently, the following conversions can be read as local eliminations of detours involving \vee, \wedge and \rightarrow :

- $\delta x.My.N\iota^1.P \gg M[P/x]$
- $\delta x.My.N\iota^2.P \gg N[P/y]$
- $\pi^i.\delta x.My.NP \gg \delta x.\pi^i.My.\pi^i.NP$
- $(\delta x.My.NP)Q \gg \delta x.(MQ)y.(NQ)P$
- $\delta x'.M'y'.N'(\delta x.My.NP) \gg \delta x.(\delta x'.M'y'.N'M)y.(\delta x'.M'y'.N'N)P$.

\perp corresponds to the *empty-type Emp*:

- to this a canonical function ε_τ from *Emp* to each type τ is associated such that

$$(\varepsilon_\tau M^{Emp})^\tau,$$

which clearly corresponds to \perp -elimination.

We have the following conversions:

- $\pi^i.(\varepsilon_{\tau_1 \times \tau_2} M) \gg \varepsilon_{\tau_i} M$
- $(\varepsilon_{\tau \rightarrow \sigma} M) N^\tau \gg \varepsilon_\sigma M$
- $\varepsilon_\tau(\varepsilon_{Emp} M) \gg \varepsilon_\tau M$
- $\delta x^\tau.M y^\tau.N(\varepsilon_{\rho + \sigma} P) \gg \varepsilon_\tau P.$

Finally, considering the conversion

$$\varepsilon_\tau(\delta x.M y.N P) \gg \delta x.(\varepsilon_\tau M) y.(\varepsilon_\tau N) P,$$

we obtain the complete engine of functional calculus associated to NJ-deductions of propositional intuitionistic logic.

Normalization results can be extended to this calculus by means of the methods we used so far modulo some necessary technical adjustments concerning terms reducibility: For the details see [30].

Accordingly, one could say that propositions-as-types correspondence is indeed an isomorphism between NJ-proofs and typed-terms: The structural concepts of detour-elimination and normalization introduced independently in these system are preserved by the Curry-Howard interpretation we discussed here.

Adding specific operators for \forall - and \exists -rules, one obtains the λP_1 -calculus, which is isomorphic from a proof-theoretic point of view to first-order NJ system. Moreover, it is possible to define a correspondence between λP_1 and simply typed λ -calculus by means of a contracting map, so that λ_{\rightarrow}^A is indeed the core of all computational significance of NJ-derivations.

4 Gödel's System T

Turning now to mathematics, we can represent the computational content of first-order theories of arithmetic by means of an extension of simple type theory which has much more expressive power and corresponds to Gödel's *Dialectica* system T, conceived by its author to prove consistency of arithmetic.

To extract programs from proofs in a system of arithmetic one could define a λ -calculus in which terms correspond exactly to proofs. This can be done in a very compact way by means of a specific method (of realizability) where first-order quantification is avoided in favour of a simple typed system we shall denote by T.

4.1 Preliminary Definitions

Types of T are defined thus:

$$\tau ::= a \mid \text{Int} \mid \rho \rightarrow \sigma \mid \rho \times \sigma,$$

where Int is an atomic type constant.

Terms of \mathbb{T} are those in Λ^T with pairs and projections, and with the addition of new constants $\mathbf{0}$, \mathbf{s} , and \mathbf{R}_τ (for all types τ) with types, respectively, \mathbf{Int} , $\mathbf{Int} \rightarrow \mathbf{Int}$, and $(\mathbf{Int} \rightarrow \tau \rightarrow \tau) \rightarrow \tau \rightarrow \mathbf{Int} \rightarrow \tau$.

\gg is extended to $\gg_{\mathbb{T}}$ by means of the following reduction schemes:

$$\mathbf{R}_\tau M N \mathbf{0} \gg_{\mathbb{T}} N$$

$$\mathbf{R}_\tau M N (\mathbf{s}P) \gg_{\mathbb{T}} M P (\mathbf{R}_\tau M N P).$$

Their intended meaning is obvious. Note however that \mathbf{R}_τ could be substituted by an iterator $\mathbf{It}_\tau : (\tau \rightarrow \tau) \rightarrow \tau \rightarrow \mathbf{Int} \rightarrow \tau$ such that $\mathbf{It}_\tau M N \mathbf{s}P \gg_{\mathbb{T}} M (\mathbf{It}_\tau M N P)$ but this operator could correspond only to the induction schema

$$\forall x (\varphi(x) \rightarrow \varphi(\mathbf{s}x)) \rightarrow \varphi(\mathbf{0}) \rightarrow \forall x (\mathbf{int}x \rightarrow \varphi(x));$$

since the recursion operator \mathbf{R} corresponds to the induction schema

$$\forall x (\mathbf{int}x \rightarrow \varphi(x) \rightarrow \varphi(\mathbf{s}x)) \rightarrow \varphi(\mathbf{0}) \rightarrow \forall x (\mathbf{int}x \rightarrow \varphi(x)),$$

\mathbf{It} it is indeed weaker than \mathbf{R} , and makes many functions definable only by values.

Notational Convention 6. By now the reader should be familiar with typed-terms: For this reason, to facilitate the exposition, we shall use various (but already known) notations to indicate the type of a term. Moreover, we shall omit at all types unless compulsory.

4.2 Normalization

Despite our adding, the main result about normalization holds for \mathbb{T} :

Theorem 4.2.1 (Strong Normalization Theorem). $\gg_{\mathbb{T}}$ is strongly normalizing.

Proof Sketch. We proceed as usual extending Tait's method: First define a new notion of *neutrality* such that a term M is said neutral iff it has one of the following form

$$x \quad M_1 M_2 \quad \pi^1.M' \quad \pi^2.M'.$$

Next assume the definition of $\widehat{\text{COMP}}_\tau$, and extend it to $\overline{\text{COMP}}_\tau$ by considering \mathbf{Int} as an atom. Hence we have:

- $\mathbf{0} \in \overline{\text{COMP}}_{\mathbf{Int}}$: obvious, since terms of atomic type are computable iff strongly normalizable;
- if $M \in \overline{\text{COMP}}_{\mathbf{Int}}$, then $\mathbf{s}M \in \overline{\text{COMP}}_{\mathbf{Int}}$: $\nu(M) = \nu(\mathbf{s}M)$;
- if M, N, P are computable, then so is $\mathbf{R}MNP$: by induction on $\nu(M) + \nu(N) + \nu(P) + \backslash P \backslash$, where $\backslash P \backslash$ is the number of symbols of P -nf; a one step reduction of $\mathbf{R}MNP$ gives one of the following
 - $\mathbf{R}M'N'P'$, with M', N', P' obtained by one step reduction, so that we can reason inductively;
 - N , when $P \equiv \mathbf{0}$, which is computable by hypothesis;

- $MQ(\mathbf{RMNQ})$ with $P \equiv \mathbf{s}Q$, thus by induction hypothesis¹⁷ \mathbf{RMNQ} is computable, and so are M, Q ; hence $MQ(\mathbf{RMNQ})$ is computable by definition.

Finally the homologous of $(*)$ - $(**)$, provable in the usual way, give the result. □

Corollary 4.2.2. $\gg_{\mathbb{T}}$ has the Church-Rosser Property.

Proof. By an easy induction on length of $\gg_{\mathbb{T}}$ -reduction we can prove that $\gg_{\mathbb{T}}$ has the *Weak Church Rosser Property*:

(WCR) For any M if $M \gg_{\mathbb{T}} M'$ in one step and $M \gg_{\mathbb{T}} M''$ in one step, then there exists an M_1 such that $M' \gg_{\mathbb{T}} M_1$ and $M'' \gg_{\mathbb{T}} M_1$.

The result follows now from Theorem 4.2.1 by the following claim:

- (\star) For any relation \succ on a set X , if \succ has Weak Church-Rosser property and is strongly normalizing, then it has Church-Rosser property.

Proof of (\star). Assume $M, N_1, N_2 \in X$, $M \succ N_1$ and $M \succ N_2$ with $N_1 \not\equiv N_2$ such that they are \succ -nf. Hence Church-Rosser property does not hold for \succ . Now we prove that for any such an M there exists an M' such that it \succ -reduces to two normal forms $M \succ M'$, against the hypothesis of strong normalization.

Since $N_1 \not\equiv N_2$, there exist M_1 and M_2 such that $M \succ M_1$ in one step, $M \succ M_2$ in one step, and $M_1 \succ N_1$ and $M_2 \succ N_2$:

- If $M_1 \equiv M_2$ then put $M' := M_1 \equiv M_2$;
 - Else, by Weak Church-Rosser property, there exists an N_3 such that $M_1 \succ N_3$ and $M_2 \succ N_3$. We can assume N_3 is a \succ -nf, and then put $M' := M_1$ or $M' := M_2$.
-

Hence if Strong Normalization holds, there cannot exist such an $M \in \Lambda^T$, and Church-Rosser property also holds. □

4.3 Kreisel's Modified Realizability

Because SN holds, T can be seen as a powerful programming language where all well-typed programs terminate. In this language we can state facts about numbers, since every term of type \mathbf{Int} has $\mathbf{s}^n\mathbf{0}$ as canonical form.

But T is in fact much more expressible: We will now show its power starting with some definitions that slightly modify the notion of realizability due to Kleene.¹⁸

Definition 4.1. We first add a new fresh constant \mathbf{i} to terms of T and we assign to \mathbf{i} a fresh type-constant 1.

¹⁷ $\nu(Q) = \nu(P)$, but $\setminus Q \setminus < \setminus P \setminus$.

¹⁸See [20] for the original definition of realizability.

- For each type τ over this extended language we define $\|\tau\|$, which is $\mathbf{1}$ free or equal to $\mathbf{1}$, as the canonical form of τ w.r.t. the following reductions:

$$\begin{aligned}\tau \times \mathbf{1} &\triangleright \tau \\ \mathbf{1} \times \tau &\triangleright \tau \\ \tau \rightarrow \mathbf{1} &\triangleright \mathbf{1} \\ \mathbf{1} \rightarrow \tau &\triangleright \tau.\end{aligned}$$

- For M of type $\|\tau_1 \times \tau_2\|$ define

$$\pi_i^{\tau_1 \tau_2}(M) := \begin{cases} \pi^i.M & \text{if } \|\tau_1\|, \|\tau_2\| \neq \mathbf{1} \\ \mathbf{i} & \text{if } \|\tau_i\| = \mathbf{1} \\ M & \text{if } \|\tau_i\| \neq \mathbf{1} \text{ and } \|\tau_{3-i}\| = \mathbf{1}. \end{cases}$$

- For M of type $\|\tau_1 \rightarrow \tau_2\|$ and N of type $\|\tau_1\|$, define

$$(MN)^{\tau_1 \tau_2} := \begin{cases} MN & \text{if } \|\tau_1\|, \|\tau_2\| \neq \mathbf{1} \\ \mathbf{i} & \text{if } \|\tau_2\| = \mathbf{1} \\ M & \text{if } \|\tau_1\| = \mathbf{1} \text{ and } \|\tau_2\| \neq \mathbf{1}. \end{cases}$$

We shall omit the superscripts whenever τ_1 and τ_2 are derivable from the context.

Now we can start our extraction of programs from proofs in Heyting first-order theory of Arithmetic HA.¹⁹

Definition 4.2. A forgetful map \flat from formulas of arithmetic to types of T extended with $\mathbf{1}$ is defined by the following clauses:

$$\begin{aligned}\flat(\perp) &:= \mathbf{1} \\ \flat(s = t) &:= \mathbf{1} \\ \flat(\exists x\varphi) &:= \|\mathbf{Int} \times \flat(\varphi)\| \\ \flat(\forall x\varphi) &:= \|\mathbf{Int} \rightarrow \flat(\varphi)\| \\ \flat(\varphi \wedge \psi) &:= \|\flat(\varphi) \times \flat(\psi)\| \\ \flat(\varphi \rightarrow \psi) &:= \|\flat(\varphi) \rightarrow \flat(\psi)\|.\end{aligned}$$

Definition 4.3. The notion of m-realizability of a closed formula of HA φ by a closed term M of type $\flat(\varphi)$ is defined inductively:²¹

- $\mathbf{i} : \mathbf{1}$ m-realizes $t = s$ iff t and s rewrite to the same numeral;

¹⁹For a detailed analysis of Heyting Arithmetic see [38]. Here we will abuse the notation about variables, provability, and the structure of derivation in these system, but the context should make clear case by case whether we are talking about T or HA.

²⁰Recall that disjunction is definable in HA: That is the reason why we do not need for sum-type in T.

²¹Here we indicate by an underline the numerals of HA, while the overline is reserved for numerals of T. These are the result of the contracting map from λP_1 to simply type calculus given in [34, Ch.8]: Individuals get erased by the contraction $\mathbf{c}(\mathbf{inta}) = \mathbf{Int}$, and what is left are proofs that given individuals are natural numbers. Thus \bar{n} is *not* obtained by contracting \underline{n} , but by contracting a *proof* of being \underline{n} a legal number.

- $M : \|\mathbf{Int} \times \mathfrak{b}(\psi)\|$ m-realizes $\exists x\psi$ iff $\pi_1(M) =_{\top} \bar{n}$ for some n and $\pi_2(M)$ m-realizes $\psi[\underline{n}/x]$;²²
- $M : \|\mathbf{Int} \rightarrow \mathfrak{b}(\psi)\|$ m-realizes $\forall x\psi$ iff $M\bar{n}$ m-realizes $\psi[\underline{n}/x]$ for all n ;
- $M : \|\mathfrak{b}(\varphi) \times \mathfrak{b}(\psi)\|$ m-realizes $\varphi \wedge \psi$ iff $\pi_1(M)$ m-realizes φ and $\pi_2(M)$ m-realizes ψ ;
- $M : \|\mathfrak{b}(\varphi) \rightarrow \mathfrak{b}(\psi)\|$ m-realizes $\varphi \rightarrow \psi$ iff MN m-realizes ψ whenever N m-realizes φ ;
- No term m-realizes \perp .

A term M is said to m-realize a HA-derivation

$$\frac{\gamma_1, \dots, \gamma_n}{\varphi}$$

whenever it realizes $\gamma_1 \rightarrow \dots \rightarrow \gamma_n \rightarrow \varphi$.

A formula or a proof is m-realizable when it is m-realized by some term.

It is easy to prove (by induction on φ) the following

Fact 4.3.1.

- (i) If M m-realizes $\varphi[t/x]$, and t and s convert to the same numeral, then M m-realizes $\varphi[s/x]$.
- (ii) Let M, N be \mathbf{i} -free terms such that $M =_{\top} N$. If M m-realizes φ , then so does N .

Notational Convention 7. Despite the absence of \mathbf{i} among m-realizers, we conveniently identify

$$\begin{array}{lll} \langle \mathbf{i}, M \rangle & \text{with} & M \\ \langle M, \mathbf{i} \rangle & \text{with} & M \\ \lambda x^\sigma. \mathbf{i} & \text{with} & \mathbf{i} \\ \lambda x^\tau. M & \text{with} & M \text{ when } \|\tau\| = 1, \end{array}$$

in addition to what is stated by Definition 4.1.

Theorem 4.3.2. *Every theorem of HA is m-realizable.*

Proof. By induction on the derivation.

It is routine to check that axioms are m-realizable.

The only case that needs for care is the induction scheme: If $\|\varphi\| = 1$, then $\mathbf{1}$ m-realizes the whole scheme, so let $\|\varphi\| \neq 1$. Let P m-realizes the induction step and let Q m-realizes the basis. Clearly, for $n > 0$, the term $P\bar{n} - \bar{1}(P\bar{n} - \bar{2}(\dots(P\bar{1}(P\mathbf{0}Q))\dots))$ m-realizes $\varphi(\underline{n})$. Then the term $M := \lambda pqn. \mathbf{R}_{\mathfrak{b}(\varphi)} pqn$ m-realizes the induction scheme.

The cases of propositional rules are straightforward and are left to the reader.

Suppose now the HA-proof has form

²²Note that \mathfrak{b} does not distinguish between algebraic terms.

$$\begin{array}{c} \Gamma \\ \vdots \\ \frac{\forall x\psi}{\psi[t/x]} \end{array}$$

where $\Gamma := \{\gamma_1, \dots, \gamma_k\}$, $\|\gamma_i\| := \sigma_i$, and free variables in $\Gamma \cup \psi[t/x]$ are $z_1, \dots, z_r := \vec{z}$. Assume M m-realizes the fragment $\frac{\Gamma}{\forall x\psi}$ and $x \in FV(\psi)$.

Then all free variables in t are among \vec{z} , and t induces an integer function definable in T by a term $\lambda\vec{y}.T$.²³ But $T[\bar{n}_1/y_1, \dots, \bar{n}_r/y_r] \gg_{\mathbb{T}} \bar{m}$ whenever $t[\underline{n}_1/z_1, \dots, \underline{n}_r/z_r]$ converts to \underline{m} , so, by Fact 4.3.1(ii), $\psi[t/x]$ is m-realized by $\lambda\vec{y}\vec{w}.M\vec{y}\vec{w}T$ where \vec{w} abbreviates $w_1 : \sigma_1, \dots, w_k : \sigma_k$. If $x \notin FV(\psi)$, then $\psi[t/x] \equiv \psi$, so the HA-derivation is m-realized simply by $\lambda\vec{y}\vec{w}.M\vec{y}\vec{w}(v^{\text{Int}})$.

Now assume the proof has form

$$\begin{array}{c} \Gamma \\ \vdots \\ \frac{\psi[t/x]}{\exists x\psi} \\ \Gamma \end{array}$$

and M m-realizes the fragment $\frac{\Gamma}{\psi[t/x]}$. Let $\lambda\vec{y}.T$ define the function induced by t . Then $\lambda\vec{y}\vec{w}.\langle T, M\vec{y} \rangle$ m-realizes the given proof.

The remaining cases are similar and are left to the reader. \square

Corollary 4.3.3. HA is consistent.

Proof. By Definition 4.3 no term m-realizes \perp , so, by Theorem 4.3.2, $\text{HA} \not\vdash \perp$. \square

Note that the proof of Theorem 4.3.2 is actually constructive: We do *construct* a realizer for any theorem of HA, and if $\text{HA} \vdash \forall x\exists y\psi(x, y)$ with ψ atomic, then its realizer is a program computing a function f satisfying $\psi(n, f(n))$ for all n . Modifying the notion of m-realizability similarly to \vdash -realizability in [34, 9.6] the result is extended to arbitrary ψ : That is what computer scientists call *program extraction*, of which Curry-Howard Isomorphism can be think of as a “coarse-grained” version where the extraction process amounts to deleting all computationally irrelevant contents from the intuitionistic proof.

²³A function $f : \mathbb{N}^j \rightarrow \mathbb{N}$ is definable in T by a closed term F iff

1. F has type $\overbrace{\text{Int} \rightarrow \dots \rightarrow \text{Int}}^{j\text{-times}} \rightarrow \text{Int}$
2. $F\bar{m}_1 \dots \bar{m}_j =_{\mathbb{T}} \overline{f(m_1, \dots, m_j)}$, for all $m_1, \dots, m_j \in \mathbb{N}$.

4.4 Expressive Power

We conclude this exposition of T using the machinery of realizability to characterize the expressive power of this system.

First note that an “upper bound” can already be stated, after a preliminary

Definition 4.4.

- Consider an enumeration $\{M_k\}$ of all Turing Machines such that given an input $\vec{n} \in \mathbb{N}^j$ return an output $m \in \mathbb{N}$. Define Kleene's T predicate such that

$$T(k, \vec{n}, m) \text{ iff } M_k \text{ halts on input } \vec{n} \text{ in } r(m) \text{ steps with output } \ell(m).^{24}$$

In [21] it is proved that T is primitive recursive.

- Let f be computed by the machine M_k . Then the characteristic function of $T(k, \cdot, \cdot)$ is denoted by t_f :

$$t_f(\vec{n}, m) = 0 \text{ iff } M_k \text{ halts on input } \vec{n} \text{ in } r(m) \text{ steps with output } \ell(m).$$

- A recursive function f is said to be provably total in a theory T iff $T \vdash \forall \vec{x} \exists y (t_f(\vec{x}, y) = 0)$.

Lemma 4.4.1. *All functions definable in T are provably total in PA.*²⁵

Proof. Any term $F : \mathbf{Int} \rightarrow \mathbf{Int}$ induces a function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that

$$f(n) = m \text{ iff } F\vec{n} \gg_{\mathbf{T}} \vec{m}.$$

This f is clearly calculable by constructing a finitely branching normalization tree. Hence, for an appropriate primitive recursive t_f , proving the formula $\forall x \exists y (t_f(x, y) = 0)$ amounts to proving that all applications of the form $F\vec{n}$ are strongly normalizable. In Tait's method one has to consider only finitely many computability predicates dependent on type and then reason by induction on them, so that the proof relative to this F can be carried out in PA modulo an appropriate arithmetical coding. □

Next we have

Theorem 4.4.2. *All function provably total in first-order arithmetic²⁶ are definable in T.*

²⁴Here ℓ and r are the (primitive recursive) converse functions of the primitive recursive pairing bijection

$$p(m, n) = \frac{(m+n)(m+n+1)}{2} + m.$$

²⁵PA is the first-order theory of Peano Arithmetic: It is characterized by the usual axioms for **s** and **0**, defining axioms for functions sum and product, and the usual induction scheme, so that its axiomatization differs from that of HA only for the presence in the latter of defining axioms for *all* recursive functions (not only + and ·).

²⁶We do not have to distinguish between PA or HA, for these systems are equivalent in proving totality of algorithms: See [34, Ch.9].

Proof. Assume $\text{HA} \vdash \forall x \exists y (t_f(x, y) = 0)$. By Theorem 4.3.2 we have an M which m -realizes the formula. Thus, for any n , $M\bar{n}$ m -realizes $\exists y (t_f(\underline{n}, y) = 0)$, $\pi_1(M\bar{n}) =_{\top} \bar{m}$ for some m , and $\pi_2(M\bar{n})$ m -realizes $t_f(\underline{n}, \underline{m}) = 0$. Since the latter is an m -realized atomic formula, $t_f(n, m) = 0$ does hold, hence $f(n)$ is the left component of m . Therefore f is defined by $\lambda x. L(\pi_1(Mx))$ where L defines the function ℓ , and we are done. \square

Together with Lemma 4.4.1, this gives

Theorem 4.4.3. *The functions definable in \top are exactly the provably total functions of first-order arithmetic.*

Corollary 4.4.4. *The SN Theorem for \top is independent from PA.*

Proof. Consider an effective enumeration $\{F_n\}$ of closed terms of \top with type $\text{Int} \rightarrow \text{Int}$ and let $g(n, k)$ be the length of the longest reduction sequence beginning with $F_n \bar{k}$. If g was provably total in first-order arithmetic, there would be a term G that defines it in \top .

Now proceed by a typical diagonalization argument, defining a term

$$H := \lambda x. \mathbf{R}_{\text{Int}}(\lambda y v. \text{sy})\mathbf{0}(\mathbf{s}(Gxx)).$$

A little thought shows that H cannot occur in $\{F_n\}$, so g is not definable. But if SN was provable in PA, we could prove the totality of g . Hence the result. \square

References

- [1] S. ABRAMSKY, D.M. GABBAY, T.S.E. MAIBAUM (eds.), *Handbook of Logic in Computer Science, Vol.2*, Oxford University Press 1992.
- [2] H.P. BARENDREGT, *The Lambda Calculus*, North-Holland 1984.
- [3] H.P. BARENDREGT, *Lambda calculi with types*, in [1].
- [4] L. BORKOWSKI, *Jan Lukasiewicz Selected Works*, North-Holland 1970.
- [5] A. CHURCH, *A formulation of the simple theory of types*, in *Journal of Symbolic Logic* 5, 1940.
- [6] C. COHEN, T. COQUAND, S. HUBER, A. MÖRTBERG, *Cubical Type Theory: a constructive interpretation of the univalence axiom*, in *arXiv 1611.02108* November 2016 (Preprint).
- [7] T. COQUAND, *Inductive definitions and type theories. An introduction*, preliminary draft for August 1999 TYPES Summer School.
- [8] R. CORI, A. RAZBOROV, S. TODORCEVIC, C. WOOD (eds.), *Logic Colloquium 2000*, A.K. Peters 2001.
- [9] A.G. DRAGALIN, *The computability of primitive recursive terms of finite type, and primitive recursive realization*, in [33].

-
- [10] A.G. DRAGALIN, *Mathematical Intuitionism. Introduction to Proof Theory*, AMS 1988.
- [11] J.E. FENSTAD, *Proceedings of the Second Scandinavian Logic Symposium*, North-Holland 1971.
- [12] J.H. GALLIER, *On Girard's "candidats de reductibilité"*, in [28].
- [13] G. GENTZEN, *Untersuchungen über das logische Schliessen*, in *Math. Zeitschrift* 39 (1935); English translation in [35].
- [14] J.-Y. GIRARD, Y. LAFONT, P. TAYLOR, *Proofs and Types*, Cambridge University Press 1989.
- [15] K. GÖDEL, *Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes*, in *Dialectica* 12, 1958; English translation in *Journal of Philosophical Logic* 9, 1980.
- [16] A. GRZEGORCZYK, *Recursive objects in all finite types*, in *Fundamenta Mathematicae* LIV, 1964.
- [17] J.R. HINDLEY, *Basic Simple Type Theory*, Cambridge University Press 1997.
- [18] J.R. HINDLEY, J.P. SELDIN, *λ -Calculus and Combinators. An introduction*, Cambridge University Press 2008.
- [19] B. JACOBS, *Categorical Logic and Type Theory*, North-Holland 1999.
- [20] S.C. KLEENE, *On the interpretation of intuitionistic number theory*, in *Journal of Symbolic Logic* 10(4), 1945.
- [21] S.C. KLEENE, *Introduction to Metamathematics*, North-Holland 1952.
- [22] G. KREISEL, *On weak completeness of intuitionistic predicate logic*, in *Journal of Symbolic Logic* 27(2), 1962.
- [23] J. LAMBEK, *The mathematics of sentence structure*, in *American Mathematical Monthly* 65, 1958.
- [24] F.W. LAWVERE, *Quantifiers and Sheaves*, in *Actes du Congrès International des Mathématiciens, Nice 1970*.
- [25] J.R. LONGLEY, *Notions of computability at higher types*, in [8].
- [26] J. ŁUKASIEWICZ, *Der Äquivalenzenkalkül*, in *Collectanea Logica* 1 (1939); English translation in [4].
- [27] P. MARTIN-LÖF, *Truth of a proposition, evidence of a judgement, validity of a proof*, in *Synthese* 73, 1987.
- [28] P. ODIFREDDI (ed.), *Logic and Computer Science*, Academic Press 1990.
- [29] D. PRAWITZ, *Natural Deduction*, Almqvist and Wiskell 1965.
- [30] D. PRAWITZ, *Ideas and results in proof theory*, in [11].

-
- [31] J.A. ROBINSON, *A machine-oriented logic based on the resolution principle*, in *Journal of Association for Computing Machinery* 12, 1965.
 - [32] H. SCHWICHTENBERG, *An upper bound for reduction sequences in typed lambda-calculus*, in *Archive für Math. Logik* 30, 1991.
 - [33] A.O. SLISENKO, *Studies in Constructive Mathematics and Mathematical Logic II*, Steklov Mathematical Institute 1970.
 - [34] M.H.B. SØRENSEN, P. URZYCZYN, *Lectures on the Curry-Howard Isomorphism*, North-Holland 2006.
 - [35] M.E. SZABO (ed.), *The Collected Papers of Gerhard Gentzen*, North-Holland 1969.
 - [36] W.W. TAIT, *Intensional interpretations of functionals of finite type I*, in *Journal of Symbolic Logic* 32(2), 1967.
 - [37] W.W. TAIT, *Gödel's unpublished papers on foundations of mathematics*, in *Philosophiae Mathematica* 9, 2001.
 - [38] A.S. TROESLA, D. VAN DALEN, *Constructivism in Mathematics. An Introduction, Vol.1*, North-Holland 1988.
 - [39] THE UNIVALENT FOUNDATIONS PROGRAM, *Homotopy Type Theory. Univalent Foundations of Mathematics*, Institute for Advanced Study 2013.